Journal of Mathematics and the Arts

ISSN: 1751-3472 (Print) 1751-3480 (Online) Journal homepage: https://www.tandfonline.com/loi/tmaa20

# Composing first species counterpoint with a variable neighbourhood search algorithm

Dorien Herremans & Kenneth Sörensen

**To cite this article:** Dorien Herremans & Kenneth Sörensen (2012) Composing first species counterpoint with a variable neighbourhood search algorithm, Journal of Mathematics and the Arts, 6:4, 169-189, DOI: 10.1080/17513472.2012.738554

**To link to this article:** https://doi.org/10.1080/17513472.2012.738554

*Journal of Mathematics and the Arts*
Vol. 6, No. 4, December 2012, 169–189

Taylor & Francis
Taylor & Francis Group

# Composing first species counterpoint with a variable neighbourhood search algorithm

Dorien Herremans* and Kenneth Sörensen

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

(*Received 25 July 2011; final version received 2 October 2012*)

In this article, a variable neighbourhood search (VNS) algorithm is developed that can generate musical fragments consisting of a melody for the *cantus firmus* and the *first species counterpoint*. The objective function of the algorithm is based on a quantification of existing rules for counterpoint. The VNS algorithm developed in this article is a local search algorithm that starts from a randomly generated melody and improves it by changing one or two notes at a time. A thorough parametric analysis of the VNS reveals the significance of the algorithm's parameters on the quality of the composed fragment, as well as their optimal settings. A comparison of the VNS algorithm with a developed genetic algorithm shows that the VNS is more efficient. The VNS algorithm has been implemented in a user-friendly software environment for composition, called *Optimuse*. Optimuse allows a user to specify a number of characteristics such as length, key and mode. Based on this information, Optimuse 'composes' both cantus firmus and first species counterpoint. Alternatively, the user may specify a cantus firmus, and let Optimuse compose the accompanying first species counterpoint.

**Keywords:** variable neighbourhood search; VNS; counterpoint; metaheuristic; local search; computer aided composing

**AMS Subject Classifications:** 68W01; 68W40; 05E99

## 1. Introduction

In this article, an algorithm is developed for automatic composition of first species counterpoint. It is argued that composing music can – at least partially – be regarded as a *combinatorial optimization problem*, in which a search reveals one or more melodies that adhere to the 'rules' of their specific musical style. To this end, these rules need to be formalized and quantified, so that the algorithm can judge whether one automatically composed melody is better than another. The better a melody fits within a certain style, the higher its solution quality will be. Although every musical genre has its own rules, these have generally not been formally written down [34]. The main reason for the choice of the contrapuntal style is that in this style there do exist formal, written rules [21] that are not too difficult to quantify. This article, therefore, focuses on automatic composition of one or two melodies that adhere to a particular set of rules for counterpoint. The algorithm could in principle be adapted to compose melodies from other styles, using rules that have been data-mined from a musical database, but this is beyond the scope of this article.

Computers and music go hand in hand these days: music is stored digitally and often played by electronic instruments. *Computer-aided composition* (CAC) is a relatively new research area that takes the current relationship between music and computers one step further, allowing the computer to aid a composer or even generate an original score. The idea of automatic music generation is not new, and one of the earliest 'automatic composition' methods based on randomness is due to Mozart. In his *Musikalisches Würfelspiel* (Musical Dice Game), a number of small musical fragments are combined by chance to generate a Minuet [9]. Other, more modern composers also make use of chance in the composition of their pieces. John Cage's piece 'Reunion' (1968) is performed by playing chess on a chessboard equipped by a photo-recepter. Each move on the chessboard triggers electronic sounds and thus a different piece is performed each time a game of chess is played [19]. Another example of chance-inspired music by Cage is 'Atlas Eclipticalis' (1961), composed by superimposing musical staves on astronomical charts [10]. Natural phenomena also inspired composer Charles Dodge. His piece 'The Earth's Magnetic Field' (1970), is a musical translation of the fluctuations of the Earth's magnetic field [3].

Hiller Lejaren and Leonard Isaacson used the Illiac computer in 1957 to generate the score of a string quartet. The resulting 'Illiac Suite' is one of the first musical pieces successfully composed by a

*Corresponding author. Email: dorien.herremans@ua.ac.be

ISSN 1751–3472 print/ISSN 1751–3480 online

http://dx.doi.org/10.1080/17513472.2012.738554
http://www.tandfonline.com

computer [43]. Lejaren and Isaacson simulated the compositional process with a three-step rule-based approach [3]. In the first step, random pitches and rhythms are generated. Then, screening rules determine whether the components of the raw composition are accepted or rejected. Various techniques such as permutation and geometric transformations are applied to improve this compositional base. A set of selection rules finally determines the material to be included in the final composition [36].

Another pioneer in algorithmic composition is Iannis Xenakis. Xenakis uses stochastic methods to aid his compositional process. For example, in 'Analogique A', Markov models are used to control the order of musical sections [48]. A more extensive overview of techniques used by avant-garde composers of the mid-to-late twentieth century is given by Cope [13].

Composing music is computationally complex, especially since the number of possible melodies increases exponentially with the length (number of notes) of the melody to compose. For instance, a musical fragment consisting of 16 notes, without rhythm, in which each note can take on 14 different pitches, already has $14^{16}$ (or roughly 2.18 quintillion) possible note combinations. This makes *exact methods* like exhaustive enumeration practically impossible to use. *Heuristic* or *metaheuristic* optimization algorithms therefore present the most promising approaches to generate high-quality melodies in a reasonable amount of time. Contrary to *exact methods*, (meta)heuristics do not necessarily return the *optimal* (i.e., best possible) solution [8], but use a variety of strategies, some seemingly unrelated to optimization (such as natural selection or the cooling of a crystalline solid), to find good solutions. A large number of metaheuristics have been proposed, which can be roughly divided into three categories. *Local search* metaheuristics (e.g., tabu search, variable neighbourhood search, iterated local search) iteratively improve a single solution. *Constructive* metaheuristics (e.g., GRASP, ant colony optimization) build a solution from its constituent parts. *Population-based* metaheuristics (e.g., genetic/evolutionary algorithms, path relinking) maintain a set (usually called the population) of solutions and *combine* solutions from this set into new ones [44].

Most methods for CAC found in the literature belong to the class of population-based, more specifically *evolutionary*, algorithms [36] that find inspiration in the process of natural evolution. Their popularity is largely due to the fact that they do not rely on a specific problem structure or domain-specific knowledge [27]. The use of other types (constructive and/or local search) of metaheuristics has remained relatively unexplored.

The first published record of a genetic algorithm (GA) used for computer aided composition is by Horner and Goldberg [27]. Their GA is applied for thematic bridging, i.e., transforming an initial musical fragment to a final fragment over a specified duration. An extensive overview of the applications of GAs in CAC over the following decade is given by Burton and Vladimirova [11] and Todd and Werner [45].

Several CAC methods rely on user input. GenJam [7] uses an evolutionary algorithm to optimize monophonic jazz solos, relative to a given chord progression. The quality of a solo is calculated based on the feedback from a human mentor. This creates a bottleneck because the mentor needs to listen to each solo and take time to evaluate it. A comparable human fitness bottleneck also arises in the CONGA system, a rhythmic pattern generator based on an evolutionary algorithm. According to the authors of the system, the lack of a quantifiable fitness function slows down the composing process and places a psychological burden on the user who listens and has to determine the fitness value [46]. Horowitz developed an interactive GA to generate rhythmic patterns. Selection is done based on an objective fitness function, whose target levels are set by the user. However, each new generation of rhythms has to be judged subjectively by the user [28].

Other CAC algorithms have attempted to eliminate the human bottleneck and define an objective function that can be automatically calculated. Moroni [35] uses a GA to evolve a population of chords. Its fitness criteria are based on physical factors related to melody, harmony and vocal range. Another approach is used by Towsey et al. [47] to develop a GA that composes Western Diatonic Music based on best practices. Their fitness function is based on 21 features and was constructed from statistics gathered by the analysis of a library of melodies.

Other studies base themselves on rules of existing music theory. Geis and Middendorf [22] implement a function that indicates how well a given fragment adheres to five harmonic rules of baroque music. This function is then used as the objective function of an ant colony metaheuristic. Orchidée is a multiobjective GA that combines musical fragments (from a database) in order to orchestrate them efficiently. The goal of this approach is to find a combination of samples that, when played together, sound as close as possible to a target timbre [12].

Notwithstanding the above examples, very little research can be found on the automatic evaluation of a melody. As mentioned, formal, written-down rules for a given musical style typically do not exist. An exception is counterpoint: the formalized nature of this style is exemplified by the fact that simple rules exist for both a single melody and the harmony

between several melodies [41]. Counterpoint starts from a single melody called the *cantus firmus* ('fixed song'), composed according to a set of melodic rules and adds a second melody (the *counterpoint*). The counterpoint is composed according to a similar set of melodic rules, but also needs to adhere to a set of *harmonic* rules, that describe the relationship between the cantus firmus and the counterpoint [24].

Aguilera et al. [2] use probabilistic logic to generate counterpoint in C major, based on a given cantus firmus. This application only assesses the harmonic characteristics of counterpoint: the melodic aspect is ignored. GPmuse [40] implements a GA that optimizes florid counterpoint in C major, by using a fitness function based on the homework problems described by Fux.

David Cope's Experiments in Musical Intelligence (EMI) project focuses on understanding a composer-specific style, by extracting signatures of musical pieces using pattern matching and then generating music with a grammar-based system [37]. More recently, Cope developed Gradus, named after John Fux's *Gradus at Parnassum* [21], which can compose first species counterpoint given a cantus firmus. Gradus analyses a set of examples of first species counterpoint and learns the best setting for six general contrapuntal goals or rules. These goals are used to sequentially generate the piece [14].

Strasheela is a generic constraint programming system for composing music. Anders [4] uses the Strasheela system to compose first species counterpoint based on 6 rules. Other constraint programming languages, such as PWConstraints developed at IRCAM can be used to generate counterpoint, provided that the user inputs the correct rules [5].

Counterpoint also includes rules for four-part music. Based on these rules, Ebcioğlu [18] developed a knowledge-based expert system for the harmonization of four-part Bach chorales. This system uses a generate-and-test method with intelligent backtracking. DePrisco et al. [16] trained three neural networks in parallel, using Bach chorales, in order to harmonize a bass line. For each note in the bass line, their algorithms try to find a three or four note chord. The output of the neural network was compared to the original harmonization made by Bach and tends to coincide in 85–90% of the cases. McIntyre [32] harmonized similar four-part Bach chorales, making use of a GA. Donnelly and Sheppard [17] also developed a GA that evolves four-part harmonic music. The fitness function consists of 10 melodic, three harmonic and two rhythmic rules based on rules of voice-leading and counterpoint. A similar problem was tackled by Phon-Amnuaisuk et al. [38] using a fitness function based on a subset of the rules for four-part counterpoint. Phon-Amnuaisuk and Wiggins [39] also compared a rule-based system with the GA for harmonising four-part monophonic tonal music. They discovered that the rule-based system delivers much better output than the GA. Their conclusion is that 'The output of any system is fundamentally dependent on the overall knowledge that the system (explicitly and implicitly) possesses' [39]. This supports the previous claim that metaheuristics that make use of domain-specific knowledge might be more efficient than a black-box GA. A more extensive overview of algorithmic composition is given by Nierhaus [36]. To the authors' knowledge, a local search metaheuristic such as variable neighbourhood search has not yet been used to generate counterpoint.

Although other studies, including the aforementioned, have used Fux's rules in order to calculate the contrapuntal quality of a musical fragment, the exact way in which they are quantified is usually not mentioned in detail. Some studies only focus on a few of the most important rules instead of looking at the entire theory [4]. The next section will discuss the Fuxian rules for first species counterpoint and how they have been quantified to determine the 'quality' (i.e., adherence to the rules) of a fragment consisting of a melody for the cantus firmus and the counterpoint. This quality score will be used as the objective function of a local search metaheuristic, developed in Section 3. The practical implementation of the developed algorithm is described in Section 4. In Section 5, a thorough parametric analysis is performed and the efficiency of the algorithm is compared to those of a random search and a GA. Final conclusions and future research opportunities are discussed in Section 6.

## 2. Quantifying musical quality

This article focuses on a specific type of polyphonic classical music called first species counterpoint. A polyphonic musical fragment consists of two or more voices, also called parts or *melodies*. The term 'counterpoint' refers to the relationship between those melodies. The complexities that arise from playing different notes at the same time has given rise to a very restrictive and formalized set of rules on how to compose polyphonic music.

The rules for species counterpoint written by Johann Fux in 1725 are one of the most restrictive sets of rules for composing renaissance music. This system was originally developed as a pedagogical tool, for students learning how to compose. It consists of five *species* or levels (first, second, third, fourth and the most complex is called *florid* counterpoint) that are taught in sequence. Each species adds more complexity

to the music, e.g., more notes per part, or different rhythmical structure. The Fuxian rules for counterpoint are foundational in music pedagogy, even today [24].

First species counterpoint is the most restrictive species. It is often referred to as 'note-against-note' counterpoint because only whole notes are allowed [1]. The rules for first species counterpoint apply to a polyphonic musical fragment consisting of two parts or melodies: a *cantus firmus* and a *counterpoint*. The cantus firmus is a base melody to which the counterpoint is added. This latter melody takes into account not only the melodic transitions between notes within the melody (i.e., the *horizontal* aspect of the music), but also the harmonic interplay between the two melodies (i.e., the *vertical* aspect) [2]. The fact that Fux's species counterpoint can be reduced to a set of simple rules [41] makes it easy to use them as quantifiers of quality in an optimization context. In this research, an extensive set of rules for first species counterpoint is used to test if a VNS can be used to automatically compose music. In future research, we plan to quantify rules of other musical styles that may be more relevant for modern composers, for instance beginning with more complex counterpoint.

Contrary to existing work, the objective function developed in this paper spans Fux's complete theory, and is based on the qualitative Fuxian rules formalized by Salzer and Schachter [42]. Full details are given in Appendix A. Some examples rules are listed below.

- Each large leap should be followed by stepwise motion in the opposite direction.
- Only consonant intervals are allowed.
- The climax should be melodically consonant with the tonic.
- All perfect intervals should be approached by contrary or oblique motion.

In order to determine how well a generated musical fragment $s$, consisting of a melody for the cantus firmus (CF) and the counterpoint (CP), adheres to the style of counterpoint, the Fuxian descriptive rules are quantified. The objective functions for the CF and CP melodies are represented in Equations (1) and (2), respectively. The objective function for the entire fragment is the sum of these two, as shown in Equation (3). Both objective functions are the sum of several subscores, one per rule, wherein each subscore takes a value between 0 and 1. The lower the score, the better the fragment adheres to the rule. A 'perfect' musical fragment would therefore have an objective function value of 0. The relative importance of a subscore is determined by its weight. The weights $a_i$ (for the subscores of 'horizontal' rules) and $b_i$ (for the subscores of 'vertical' rules) can be set by the composer to calculate the total score. The score for the CF melody ($f_{\mathrm{CF}}$) consists only of a horizontal aspect, whereas that of the CP melody ($f_{\mathrm{CF}}$) also includes a vertical score that evaluates the interaction between the two melodies. The total score of the fragment (Equation (3)) is used only as a summary statistic as the CF and the CP melodies are optimised separately.

$$f_{\mathrm{CF}}(s) = \underbrace{\sum_{i=1}^{18} a_i.\mathrm{subscore}_i^H(s)}_{\text{horizontal aspect CF}} \tag{1}$$

$$f_{\mathrm{CP}}(s) = \underbrace{\sum_{i=1}^{18} a_i.\mathrm{subscore}_i^H(s)}_{\text{horizontal aspect CP}} + \underbrace{\sum_{j=1}^{15} b_j.\mathrm{subscore}_j^V(s)}_{\text{vertical aspect CP}} \tag{2}$$

$$f(s) = f_{\mathrm{CF}}(s) + f_{\mathrm{CP}}(s) \tag{3}$$

The rules used in the above objective function can be seen as 'soft' constraints. Although the algorithm developed in the next section tries to find a musical fragment that violates as few of these rules as possible (and by as little as possible), a fragment that breaks some of the rules is not necessarily 'infeasible'. For pieces of arbitrary length, it is additionally not known whether it is possible to adhere to all the rules. Music theory also imposes a set of 'hard rules' that have been implemented as constraints in the optimization problem. These rules, such as 'all notes are from the tonal set' and 'all notes are whole notes', always need to be satisfied in order to obtain a feasible fragment.

The very restrictive nature of counterpoint and the large number of rules that all need to be satisfied at the same time make it a difficult craft for human composers to master. For several of the rules, it is not easy to see or hear if they are fulfilled. For example, it takes an extremely well-trained ear or a laborious manual counting of the intervals to check whether the rule 'the beginning and the end of all motion needs to be consonant' has been satisfied. However, the quantification of the rules for counterpoint allows them to be assessed objectively by a computer, and allows a computer to judge whether one fragment is 'better' than another. This information is used in the algorithm developed in the next section.

As an example, we calculate the objective function score of the textbook example of counterpoint given by Salzer and Schachter [42], shown in Figure 1. As expected, both the horizontal (respectively, 0.44 and 0.64) and the vertical part (0.5) of the score are very good and close to the optimum ($f(s) = 0$), yet not totally optimal. This shows that experienced

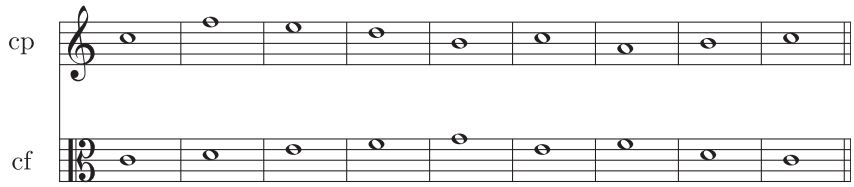


Figure 1. Example of cantus firmus and counterpoint with objective score $f(s) = 1.589$.

composers can still ascribe quality to a musical fragment even when it does not obey all Fuxian rules of good counterpoint. As a comparison, the average score of 50 random fragments of the same length is 16.39.

## 3. Variable neighbourhood search to generate counterpoint

Most of the existing research on the use of metaheuristics for CAC revolves around evolutionary algorithms. The popularity of these algorithms can partially be explained by their 'black-box' character. Essentially, an evolutionary algorithm requires only that a 'fitness' (objective) function can be calculated. Other components, such as the recombination operator or the selection operator, do not require any knowledge of the problem that is being solved. This makes it very easy to implement an evolutionary algorithm, but the resulting algorithm is not likely to be efficient, compared to other metaheuristics that exploit the specific structure of the problem.

Local search heuristics, on the other hand, are generally more focused on the specific problem and use far less randomness to search for good solutions. These methods typically operate by iteratively performing a series of small changes, called *moves*, on a *current solution* $s$. The *neighbourhood* $N(s)$ consists of all feasible solutions that can be reached in one single move from the current solution. The type of move therefore defines the neighbourhood of each solution. The local search algorithm always selects a better fragment, i.e., a solution with a better objective function value, from the neighbourhood. This solution becomes the new current solution, and the process continues until there is no better fragment in the neighbourhood. It is said that the search has arrived at a *local optimum* [26].

To escape from a local optimum, the local search metaheuristic called VNS (variable neighbourhood search) uses two mechanisms. First, instead of exploring just one neighbourhood, the algorithm switches to a different neighbourhood (defined by a different move type) if it arrives in a local optimum [33]. The underlying idea is that a local optimum is specific to a certain neighbourhood. By allowing other move types, the search can continue. A second mechanism used by VNS is a so-called *perturbation*, that randomly changes a relatively large part of the current solution.

Although VNS is a relatively recent metaheuristic, it has been successfully applied to a broad range of optimization problems. Davidović et al. [15] developed a VNS for the multiprocessor scheduling problem with communication delays. This VNS outperforms both the tabu search and genetic search algorithm developed in the same paper. Other applications of VNS include, but are not limited to, vehicle routing [29], project scheduling [20] and graph coloring [6]. Hansen and Mladenović [25] applied a VNS to five well known combinatorial optimization problems (the Travelling Salesman Problem (TSP), the p-median problem (PM), the multi-source Weber (MW) problem, the partitioning problem and the bilinear programming problem with bilinear constraints (BBLP)). Their comparison with recent heuristics showed that VNS is very efficient for large PM and MW problems. For several other problems VNS outperforms existing heuristics in an effective way, meaning that the best solutions are found in moderate computing time [25].

The effectiveness and efficiency of VNS in solving combinatorial optimization problems motivate us to apply this technique to the domain of CAC. In the following sections, a VNS is described that was implemented for composing counterpoint.

### 3.1. *Components of the VNS algorithm*

The VNS algorithm developed in this article operates in two sequential phases. First, the cantus firmus is generated, then the first species counterpoint. The algorithm used to generate both melodies is identical, and differs only in the objective function used. This two-phased design choice originated from the fact that a contrapuntal line is usually composed on top of an existing cantus firmus line. It must be noted that, by keeping the cantus firmus fixed, finding a suitable matching contrapuntal line is highly dependent on the quality of this cantus firmus.

Table 1. Neighbourhoods.

| $N_i$ | Name | Description | Neighbourhood size |
|---|---|---|---|
| $N_1$ | Swap | Swap two notes | $\binom{16\times L}{2}$ |
| $N_2$ | Change1 | Change one note | $9 \times 16 \times \mathrm{L}$ |
| $N_3$ | Change2 | Change two notes | $9 \times 9 \times (16\mathrm{L}-1)$ |

Note: $L$ is the length of the fragment expressed in units of 16 notes.

A 'solution' in our algorithm is a musical fragment consisting of a melody for the cantus firmus and the first species counterpoint. The algorithm takes as its input the key and length of the fragment that is to be composed. The length is expressed as the number of notes and can be any multiple of 16. A fragment that has the correct number of notes, all of which are in the range of allowed pitches in the specified key, is called *feasible*. A set of 10 allowed sequential pitches is defined for the cantus firmus, starting from MIDI values 48–59, depending on the specified key. A different set of 10 pitches is defined for the contrapuntal part, which starts nine semitones higher. This means that for a fragment of $n$ notes, there are $10^n$ possible note combinations per voice. The functioning of the VNS algorithm ensures that no infeasible fragments are generated.

Three types of moves have been defined, as shown in Table 1 and Figure 2.

The first neighbourhood ($N_1$) is defined by the `Swap` move type and is generated by swapping every pair of notes, starting from the current fragment. An example of a swap move is shown in Figure 2(b). $N_1$ will thus contain all possible fragments that can be obtained by swapping two notes in the current fragment.

Neighbourhoods $N_2$ and $N_3$ are, respectively, defined by move types `Change1` and `Change2`. The `Change1` move will change the pitch of any one note to any other allowed pitch. The last move, `Change2`, is an extension of the previous one whereby the pitches of two consecutive notes are changed simultaneously to any other allowed pitches. These last two moves are illustrated in Figure 2(c) and 2(d).

The sizes of the neighbourhoods can be calculated with the formulas in Table 1. As an example, the sizes of the neighbourhoods for a fragment of 64 notes are 2016, 576 and 5103.

For each of the neighbourhoods $N_1$, $N_2$ and $N_3$, the algorithm generates all possible feasible fragments $s$ by applying the corresponding move and selects the one with the best value in the objective function $f(s)$. This strategy is called a *steepest descent* and typically ensures a fast improvement of the value of the objective function.

When no improving fragments can be found in any of the neighbourhoods of the current fragment, the VNS algorithm uses a *perturbation* strategy to allow the search to continue. The perturbation is implemented by reverting back to the global best fragment and changing a predefined percentage of the notes to new, random notes from the key. The reason for performing the perturbation move from the global best fragment, and not from the current fragment, is that this strategy was found to lead to better fragments faster.

Often, the current fragment scores optimally with respect to a large majority of subscores, but performs badly with respect to some others. To correct this behaviour, the VNS uses an *adaptive weight adjustment* mechanism. This mechanism adapts the weights of the subscores of the objective function at the same time that a random perturbation is performed. The default initial setting for all of the weights is 1, making them normalized and all equally important in the calculation of the objective score. These initial weights can be changed by the user, in order to favour specific rules. The adaptive weight mechanism works by increasing the weight of the subscore that has the highest value (i.e., the subscore on which the current fragment has worst performance) by 1 immediately after the perturbation move. The algorithm then uses the score based on these new weights (called the *adaptive score* $f^a(s)$) to determine the quality of fragments in the neighbourhoods. In order to determine whether a fragment is considered as the new global best, however, the algorithm always considers the original weights. This weight adjustment strategy increases the impact of otherwise insignificant moves with little impact on the objective function. An increase of 1 ensures a constant pressure on subscores that are high during a particular run. In principle, the weights can become as high as the number of perturbations performed. However, since subscores with high weights are taken into account more during a move, their values generally decrease quickly, which causes the adaptive weights mechanism to favour other subscores. Because the weights never decrease, the probability that such a subscore will increase again is very small. An increase of 1 will usually still have an impact on the newly favoured subscore, since the value of the subscores with high weights are usually very low or 0, which partly neutralizes their weights even if they had become very high.

After the perturbation, the VNS always uses the `Swap` move first. The reason for this is that this neighbourhood does not introduce new notes, and

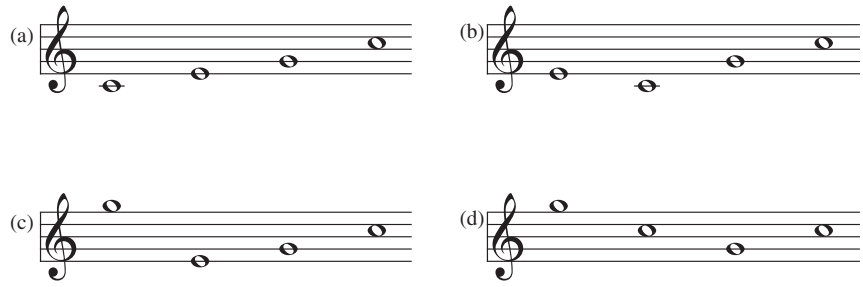


Figure 2. Moves. (a) Original. (b) Swap move. (c) Change1 move. (d) Change2 move.

therefore the search cannot immediately converge back to the previous local optimum.

To prevent the algorithm from getting trapped in cycles (i.e., revisiting the same local optimum again and again), a simple short-term memory structure is introduced. This tabu list [23] prohibits notes in certain positions from being changed. More specifically, the notes in positions that have been changed in the previous iterations by a move cannot be changed by a move of the same type. The resulting moves are referred to as *tabu active*. The number of iterations during which a move remains tabu active is called the *tabu tenure*. Each neighbourhood has its own tabu list, including its own tabu tenure. The tabu lists work by storing the recently changed notes and keeps them from being changed again by the same type of move. The tabu tenure is expressed as a fraction of the length of the melody in number of notes.

### 3.2. *General outline of the implemented VNS algorithm*

Figure 3 depicts a flow chart representation of the developed VNS algorithm. The VNS starts by generating a fragment $s$ consisting of random notes from the key. This fragment is set as the initial global best solution $s_{\text{best}}$.

For the current solution $s$, the `Swap` neighbourhood ($N_1$) is generated. Moves on the tabu list of $N_1$ are excluded and a feasibility check is performed. The best solution $s'$ of this neighbourhood is selected as the new current solution $s$, if its objective function with adaptive weights is better than that of $s$ ($f^a(s') < f^a(s)$). The move applied to obtain $s'$ is added to the tabu list of the `Swap` neighbourhood on a *first in first out* basis. This procedure is repeated as long as a better current solution is found, based on the objective function with adaptive weights. Each time a move is performed, the current solution $s$ is compared to the best global solution $s_{\text{best}}$, based on the original objective function $f(s)$. If $f(s)$ is better than $f(s_{\text{best}})$, the current solution $s$ becomes the new global best solution $s_{\text{best}}$.

If no better current solution, based on $f^a(s)$, is found in the first neighbourhood, the algorithm switches to the `Change1` neighbourhood ($N_2$) and repeats the same procedure. Again if it cannot find a better current solution in the `Change1` neighbourhood, it switches to the `Change2` neighbourhood ($N_3$).

If the algorithm goes through all three of the neighbourhoods without finding a better current solution (based on $f^a(s)$), then a perturbation is performed. A fraction $r$ of the notes of $s$ are randomly changed to form the new current solution $s$. The weight of the subscore of the best solution ($s_{\text{best}}$) that has the highest value is also increased by 1. If $f(s)$ is better than $f(s_{\text{best}})$, the current solution $s$ is recorded as the new global best solution $s_{\text{best}}$.

The algorithm continues to repeat these steps (generating the three types of neighbourhoods), until either the optimum is found or it has reached the maximum number of iterations without improving the best global solution $s_{\text{best}}$, as specified by the user through the *maxiters* parameter.

## 4. Implementation

The VNS algorithm has been implemented in `C++`. To improve the usability of the software, a plug-in for the open source music composition and notation program MuseScore has been developed in JavaScript using the QtScript engine (Figure 4). This allows for easy interaction with Optimuse (the implementation of the VNS algorithm) through a graphical user interface. A cantus firmus can either be generated from a menu item, or composed on screen by clicking on the staff. When the cantus firmus has been generated, the counterpoint can be generated from a second menu item. The user can specify the input parameters such as the key (e.g. G# minor) and the weights for each subscore in an input file (*input.txt*). Other parameters of the algorithm, such as the ones discussed in the next

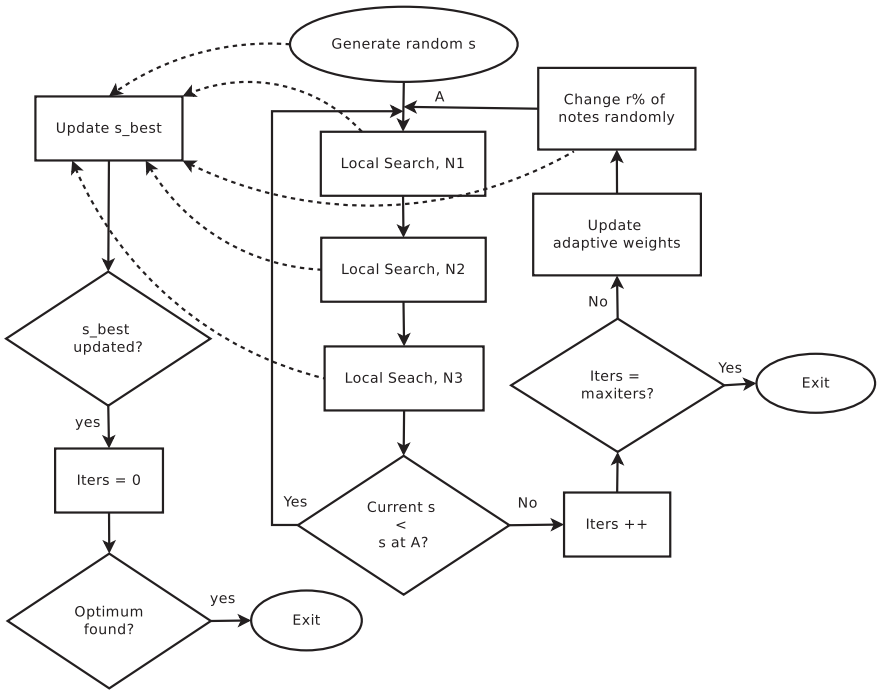


Figure 3. Flowchart of the developed VNS algorithm.

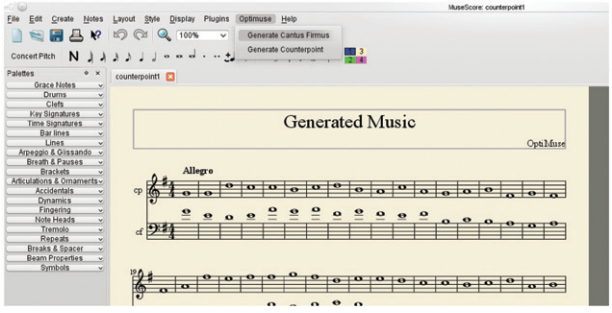


Figure 4. MuseScore.

section, can optionally be passed as command line arguments. The resulting counterpoint is displayed in score notation and can immediately be played back. MuseScore also provides easy export to MIDI, PDF, lilypond and other popular music notation formats. The fragment is also automatically exported in the MusicXML format, a relatively new format that is designed to facilitate music information retrieval. MuseScore can be downloaded at musescore.org. Optimuse is available for download at antor.ua.ac.be/optimuse.

## 5. Experiments

In this section, we describe an experiment we performed to set the parameters of the developed variable neighbourhood search algorithm to their optimal levels. The VNS is then compared to a random search and a GA. Finally, an example of generated music is presented.

### 5.1. *VNS parameter optimisation*

The VNS algorithm described in Section 3 has several components. In this section, an experiment is described that has been set up to test the effectiveness of the different parts of the developed VNS and to determine their optimal parameter settings. In this way, components of the algorithm that do not contribute to the final solution quality can be removed. Separate experiments were performed for the generation of cantus firmus and of counterpoint. The experiment for counterpoint uses the cantus firmus generated by the VNS with the same parameter settings. The different parameters that were tested are displayed in Table 2. In Section 5.1.3, a small fragment is generated with the best parameter settings. The algorithm can generate musical fragments of any length, as long as the number of notes is a multiple of 16. The experiment has been limited to fragments with lengths of 16, 32, 48 and 64 notes.

A full-factorial experiment was performed for each of the experimental groups (cantus firmus and counterpoint). This means that it includes some runs that have all three neighbourhoods deactivated. In that

Table 2. Parameters of the VNS.

| Parameter | Values | No. of levels |
|---|---|---|
| $N_1$ - `Swap` | on with $tt_1=0$, $tt_1=\frac{1}{4}$, $tt_1=\frac{1}{2}$, off | 4 |
| $N_2$ - `Change1` | on with $tt_2=0$, $tt_2=\frac{1}{4}$, $tt_2=\frac{1}{2}$, off | 4 |
| $N_3$ - `Change2` | on with $tt_3=0$, $tt_3=\frac{1}{4}$, $tt_3=\frac{1}{2}$, off | 4 |
| Random move (randsize) | $\frac{1}{4}$ changed, $\frac{1}{8}$ changed, off | 3 |
| Adaptive weights (adj. weights) | on, off | 2 |
| Max. number of iterations (iters) | 10, 50, 100 | 3 |
| Length of music (length) | 16, 32, 48, 64 notes | 4 |

Note: $tt_i$ = tabu tenure of the tabu list of neighbourhood $N_i$, expressed as a fraction of the total number of notes.

case, the algorithm simply performs perturbations, depending on the random move factor. The total number of runs for both groups is $n=4068$ ($2\times 3^2\times 4^4$). The results of these 4068 runs of the algorithm were analysed by performing a Multi-way ANOVA (Analysis of Variance). Using the open source statistical software R, a model was estimated that takes into account these basic variables (Tables 3 and 4) and analyses their impact on the musical quality of the end result, as well as on the running time of the algorithm.

#### 5.1.1. *Cantus firmus*

When including only the main effects, the $R^2$ statistic of the linear regression (Table 3) shows that approximately 26% of the total variation around the mean value of the objective function can be explained by the model. However, the value of the $R^2$ statistic and therefore also the quality of the model can be increased by including interaction effects. In order to determine which parameters have a significant influence on the quality of the generated music, a model was calculated that takes into account the interaction effects of the significant factors ($p<0.05$). This high quality model has an $R^2$ statistic of approximately 96% (Table 5), which means that it can explain 96% of the variation. The most important influential factors of the model are displayed in Table B1 in appendix B.

A similar Multi-Way ANOVA test has been performed using the computation time of the algorithm as the dependent variable. The results of this analysis are displayed in Table 6 and show that all factors, except for the tabu list tenure for $N_1$ and $N_3$, have a significant influence on the running time.

Examination of the results of the ANOVA Table with interaction effects reveals that most of the factors have a very low $p$-value. This means that they have a significant influence on the result. The large number of small $p$-values indicates that most of the factors make a contribution to the model. Only $tt_1$, the tabu tenure of $N_1$, does not seem to have a significant influence on the quality of the result. The exact effect that the different parameters settings have on the objective function of the end result is visualized in their means plots (Figure C1 in appendix C).

Table 3. Model without interactions (CF) - summary of fit.

| Measure | Value |
|---|---|
| $R^2$ | 0.2605 |
| $R^2$ Adj | 0.2577 |
| $F$-statistic | 95.1 |
| $p$-value | $<2.2\times10^{-16}$ |

Table 4. Multi-way ANOVA model without interactions (CF).

| Parameter | Df | $F$ value | Prob ($>F$) |
|---|---|---|---|
| $N_1$ | 1 | 220.3070 | $<2.2\times10^{-16}$* |
| $N_2$ | 1 | 369.0956 | $<2.2\times10^{-16}$* |
| $N_3$ | 1 | 731.9956 | $9.702\times10^{-6}$* |
| $tt_1$ | 2 | 0.0373 | 0.9634 |
| $tt_2$ | 2 | 0.6274 | 0.5340 |
| $tt_3$ | 2 | 0.6477 | 0.5233 |
| Randsize | 2 | 111.4955 | $<2.2\times10^{-16}$* |
| Adj. weights | 1 | 19.6100 | $9.716\times10^{-6}$* |
| Iters | 2 | 14.1792 | $7.261\times10^{-7}$* |
| Length | 3 | 7.2613 | $7.406\times10^{-5}$* |

Note: The asterisks indicate statistical significance $p<0.05$.

Table 5. Model with interactions (CF) - summary of fit.

| Measure | Value |
|---|---|
| $R^2$ | 0.9612 |
| $R^2$ Adj | 0.9556 |
| $F$-statistic | 171.8 |
| $p$-value | $<2.2\times10^{-16}$ |

Table 6. Multi-Way ANOVA model for Time.

| Parameter | Df | $F$ value | Prob ($>F$) |
|---|---|---|---|
| $N_1$ | 1 | 36.6801 | $1.503 \times 10^{-9}$* |
| $N_2$ | 1 | 13.3081 | 0.0002672* |
| $N_3$ | 1 | 22.4816 | $<2.186 \times 10^{-6}$* |
| $tt_1$ | 2 | 0.3025 | 0.7389918 |
| $tt_2$ | 2 | 3.8164 | 0.0220770* |
| $tt_3$ | 2 | 1.5815 | 0.2057780 |
| Randsize | 2 | 218.6638 | $<2.2 \times 10^{-16}$* |
| Adj. weights | 1 | 18.2537 | $1.973 \times 10^{-5}$* |
| Iters | 2 | 210.8771 | $<2.2 \times 10^{-16}$* |
| Length | 3 | 691.4822 | $<2.2 \times 10^{-16}$* |

Notes: $R^2$=0.3981
The asterisks indicate statistical significance $p<0.05$.

The ANOVA Table (Table B1) shows that parameters $N_1$, $N_2$ and $N_3$ all have a significant influence on the quality of the generated music ($p < 2 \times 10^{-16}$). Figures D1(a), D1(b) and D1(c) show that activating one of the neighbourhoods will have a decreasing effect on the score. However, they also show that adding a neighbourhood increases the mean running time for both $N_1$ and $N_3$. According to Table 6, this effect on the running time is significant. In deciding the optimal parameter settings, the primary objective was the musical quality; the computing time was considered a secondary objective. The three neighbourhoods will therefore be included in the optimal parameter set.

From Tables 6 and B1, it is clear that the tabu tenure for $N_1$ does not have a significant effect on the quality or the computing time ($p$-values 0.5 and 0.7). The tabu tenures for $N_2$ and $N_3$ do have a significant influence. The means plots D1(e) and D1(f) show that tabu tenures of respectively $\frac{1}{4}$ and $\frac{1}{2}$ (i.e., 25% and 50% of the length of the number of notes of the melody respectively) contribute most to a higher result quality.

With $p < 2 \times 10^{-16}$, the random perturbation size clearly has a significant effect on the solution quality. Plot D1(g) indicates that a perturbation will result in music of better quality, and that a perturbation of 12.5% offers the best results in the objective score, although this has a significant negative effect on the computing time.

Table B1 demonstrates that the adaptive weights parameter also has a significant influence on the musical quality if we take the interaction effects into account. Figure D1(h) shows that activating this parameter has a significant lowering effect on the mean computing time as well as the objective function. This means that the adaptive weights procedure makes a positive contribution to the general effectiveness of the VNS.

The maximum number of iterations is a significant factor in the algorithm ($p < 2.2 \times 10^{-16}$). The means plot D1(i) clearly shows that solution quality improves when the maximum number of iterations is higher. However, as expected, this is paired with a significant increase in computing time.

Table 7. Model with interaction effects (CP) - summary of fit.

| Measure | Value |
|---|---|
| $R^2$ | 0.9122 |
| $R^2$ Adj | 0.9062 |
| $F$-statistic | 152.4 |
| $p$-value | $<2.2 \times 10^{-16}$ |

Table 8. Multi-way ANOVA model with interaction effects (CP).

| Parameter | Df | $F$ value | Prob ($>F$) |
|---|---|---|---|
| $N_1$ | 1 | 1173.4292 | $<2.2 \times 10^{-16}$* |
| $N_2$ | 1 | 2618.9755 | $<2.2 \times 10^{-16}$* |
| $N_3$ | 1 | 6498.1957 | $<2.2 \times 10^{-16}$* |
| Randsize | 2 | 2610.1349 | $<2.2 \times 10^{-16}$* |
| Iters | 2 | 111.7095 | $<2.2 \times 10^{-16}$* |
| Length | 3 | 125.8989 | $<2.2 \times 10^{-16}$* |
| $tt_1$ | 2 | 1.3815 | 0.2513093 |
| $tt_2$ | 2 | 7.5519 | 0.0005321* |
| $tt_3$ | 2 | 188.5756 | $<2.2 \times 10^{-16}$* |
| Adj. weights | 1 | 18.5697 | $1.675 \times 10^{-05}$* |

Notes: Interactions of all significant factors are included in the model, but omitted in the table for clarity.
The asterisks indicate statistical significance $p<0.05$.

### 5.1.2. *Counterpoint*

The same full factorial experiment was run on the VNS algorithm to generate the counterpoint.

Table 7 indicates that the model with interactions can explain approximately 91% of the total variation around the mean value of the objective function. The detailed results of this model are displayed in the ANOVA Table with interactions (Table 8). The ANOVA model shows the same significant parameters as the analysis in the previous section. The mean plots for the cantus firmus (Figure D1 in appendix D) also show a strong resemblance to the ones with the results for the counterpoint (Figure C1). The conclusions of the previous section can therefore be extended to the results for counterpoint.

### 5.1.3. *Optimal parameter settings*

The means plots and ANOVA give a good indication of the significant parameters and their optimal settings.

Table 9. Best parameter settings.

| Parameter | Values |
|---|---|
| $N_1$ - Swap | on with $tt_1 = 0$ |
| $N_2$ - Change1 | on with $tt_2 = \frac{1}{4}$ |
| $N_3$ - Change2 | on with $tt_3 = \frac{1}{2}$ |
| Random move | $\frac{1}{8}$ changed |
| Adaptive weights | on |
| Max. number of iterations | 100 |
| Length of music | 64 notes |

Note: $tt_i$ = tabu tenure of the tabu list of neighbourhood $N_i$, expressed as a fraction of the total number of notes.

Interaction plots for the significant interaction effects, drawn in R, support the conclusions made in the previous section. Table 9 summarizes the optimal parameter settings. As mentioned, these parameters were set in a way that always favoured solution quality over computing time. In other words, the effect of a parameter on the computing time was only taken into account if the means plot did not show a significant effect on the quality.

The algorithm was run again with these optimal parameter settings. Figure 5 shows the evolution of the solution quality over time, which is characterized by a fast improvement of the best solution in the beginning of the algorithm's run. After several iterations, the improvements diminish in size, especially in the case of counterpoint. The random perturbations can be spotted on the graph as peaks in the objective score. They are usually followed by a steep descent which often results in a better 'best solution', confirming the importance of the random perturbation factor. The perturbation is typically performed after calculating 50–100 neighbourhoods. This number is not exact: it varies with each run and decreases after a while.

The random initial fragment serves as a starting point for the VNS. In order to examine the influence of the initial fragment on the generated music, the VNS was run two times for a cantus firmus of 64 notes with maxiters set to 10. In the end results for both runs, only 12% and 34% of the notes of the initial random fragment were unchanged. This experiment shows that the initial fragment is generally changed quite thoroughly by the algorithm. When comparing the two generated fragments, 77% of the notes in the cantus firmus are different. Similar results are seen with the counterpoint. When the VNS was run two times to generate a CP, there was a 70% difference between two generated melodies and 7% and 14% resemblance with the original random fragment. This shows that two musical fragments generated by the VNS and based on the same initial fragment have very little resemblance.

### 5.2. *VNS versus random search*

In order to determine the efficiency of the VNS algorithm, it was compared to a random search. This comparison was performed on a musical piece consisting of 64 notes. The VNS was run for one iteration, until it reached its first local optimum. No perturbation was performed. In one iteration, it performed 28 moves in $N_1$, 13 in $N_2$ and 13 in $N_3$ for the cantus firmus and 32 moves in $N_1$, 15 in $N_2$ and 12 in $N_3$ for the counterpoint. In order to evaluate the generated neighbourhoods, the algorithm had to calculate the objective score a total of respectively 147,968 and 150,912 times for cantus firmus and counterpoint. A random search generated an equal amount of solutions. The experiment for counterpoint, for both VNS and random search, started from the same initial cantus firmus that was generated by the VNS with optimum parameters. Figure 6 shows that the VNS is clearly able to find a much better solution, especially for the counterpoint. Although, the VNS moves more slowly in the very beginning of the run, it is able to find a better solution almost immediately. This conclusion is supported by Figure 7, where the best found objective score is plotted against the running time of the algorithms.

### 5.3. *VNS versus GA*

A formal comparison with a previously developed GA is not possible because the cited algorithms use a different fitness function than the developed VNS. In order to compare the effectiveness of the VNS with a GA, a GA was implemented that uses the objective function described in this article as fitness function.

The GA starts from a population of randomly generated solutions. Binary tournament selection is used to select two parents from the population. A two-point-crossover is performed on two random points to create the children. A mutation, changing a fraction $r$ of the notes randomly, is performed on one of the children with a probability $p$. The children are then reinserted in the population with reversed binary tournament selection. A child is excluded from the population if an equal population member already exists. This keeps the population from converging.

In the developed GA, a number of parameters can be set to different levels. A similar full factorial experiment to the one performed on the VNS was done to determine the best settings for the parameters represented in Table 10. The GA was run 864 times, once for each of the settings for the cantus firmus and the counterpoint. The counterpoint takes the cantus firmus generated by the GA with the same parameter settings as input.

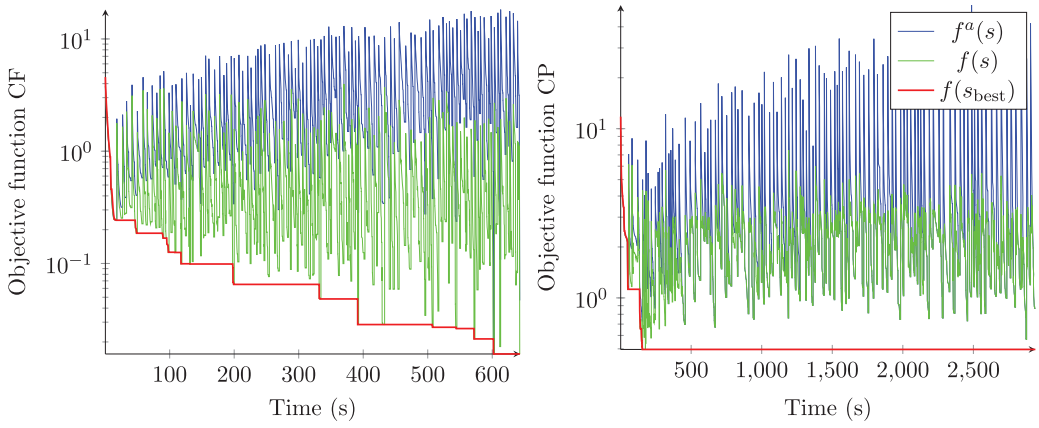


Figure 5. Evolution of the VNS with optimal parameter settings.

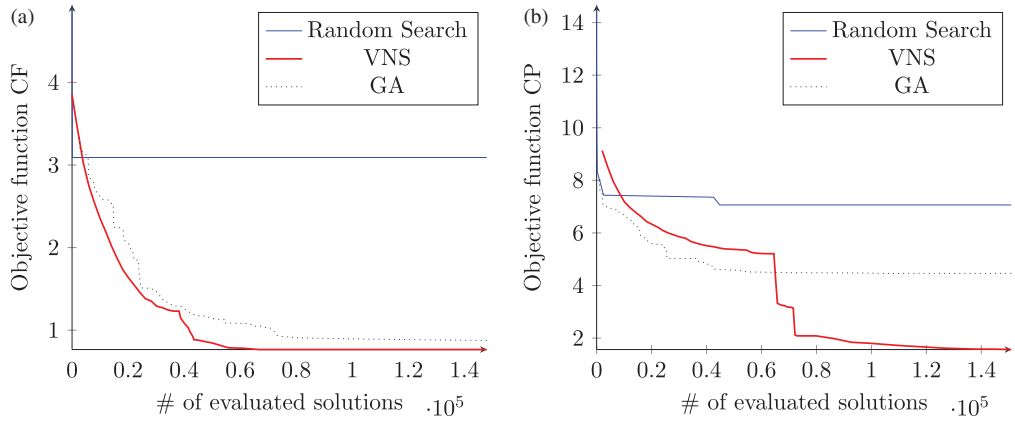


Figure 6. Comparison of VNS and random search. (a) Cantus firmus. (b) Counterpoint.

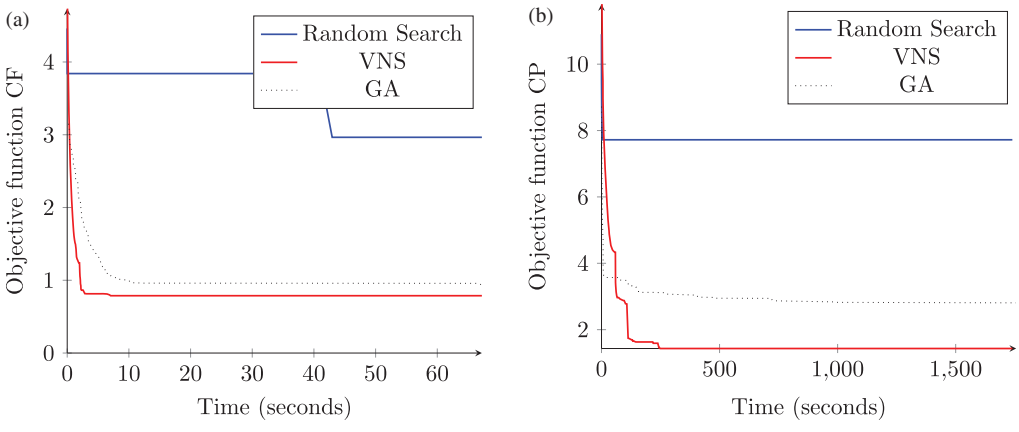


Figure 7. Comparison of VNS and GA. (a) Cantus firmus. (b) Counterpoint.

An ANOVA model was constructed from the results. The initial model, both for CF and CP, shows that almost all factors are significant. Only the number of generations and the mutation frequency do not have a significant influence on the solution quality. A new model with interaction effects between significant factors was built. The results are similar for the cantus firmus and the counterpoint, except that the

Table 10. Parameters of the GA.

| Parameter | Values | No. of levels |
|---|---|---|
| Population size | 10, 100, 1000 | 3 |
| Mutation size | 0%, 12.5%, 25% | 3 |
| Mutation frequency | 5%, 10% | 2 |
| Number of generations (gens) | 150,000, 750,000, 1,500,000 | 3 |
| Length of music (length) | 16, 32, 48, 64 notes | 4 |

Table 11. Multi-way ANOVA model with interactions (CP).

| Parameter | Df | $F$ value | Prob ($>F$) |
|---|---|---|---|
| Population size | 2 | 307.6332 | $<2.2\times10^{-16}$* |
| Mutation size | 2 | 1114.9281 | $<2.2\times10^{-16}$* |
| Length | 3 | 193.1132 | $<2.2\times10^{-16}$* |
| Mutation frequency | 1 | 2.3477 | 0.127255 |
| Gens | 2 | 10.8473 | $3.599\times10^{-05}$* |
| Population size:mutation size | 4 | 362.2764 | $<2.2\times10^{-16}$* |
| Population size:length | 6 | 3.0996 | 0.006535* |
| Mutation size:length | 6 | 8.9242 | $1.634\times10^{-08}$* |
| Population size:mutation size:length | 12 | 1.0715 | 0.386779 |

Notes: $R^2$=0.9657
The asterisks indicate statistical significance $p<0.05$.

number of generations is not significant for the CF. The ANOVA summary for the CP is displayed in Table 11. The optimal parameter settings with regards to the solution quality are displayed in Table 12. These were obtained by an analysis of the mean plots and interaction plots output by R.

In order to compare the efficiency of both algorithms another experiment was set up. The VNS and GA algorithms both generated 50 fragments of cantus firmus and counterpoint consisting of 64 measures. The 50 cantus firmus instances generated by the VNS were used by both the GA and the VNS as input to compose the counterpoint. The algorithms used the optimal parameter settings that resulted from the above analysis. The cut-off for the VNS is 25 maxiters, and for the GA, respectively, 1,500,000 and 3,000,000 generations for cantus firmus and counterpoint. This resulted in comparable running times for both algorithms, although the average running time of the GA is slightly longer than that of the VNS (Table 13). The computer used in this experiment has an Intel© Core™i7 CPU which runs at 2.93 GHz. To ensure a fair comparison, both algorithms are coded in C++ and large parts of their code base is the same. For example, both algorithms use the same code to calculate the objective function.

The findings of this experiment were analysed by a unilateral paired t-test. The resulting $p$-values of 0.01593 for the cantus firmus and $5.363\times10^{-15}$ for the counterpoint show that the VNS is able to find a better solution than the GA, despite its shorter mean running time.

Table 12. Optimal parameter settings for the GA.

| Parameter | Values |
|---|---|
| Population size | 1000 |
| Mutation size | 12.5% |
| Mutation frequency | 10% |
| Number of generations | 1,500,000 |

Table 13. A comparison of the mean running time for the VNS and GA.

| | Mean running time VNS | Mean running time GA |
|---|---|---|
| CF | 563s | 795s |
| CP | 1823s | 1992s |

Figure 7 shows the evolution over time of the best found solution for the VNS, GA and Random Search. When generating the cantus firmus, all algorithms start from a randomly generated solution. For the generation of the counterpoint all three runs are based on the same cantus firmus generated by the VNS with the optimal settings. The graph shows that although

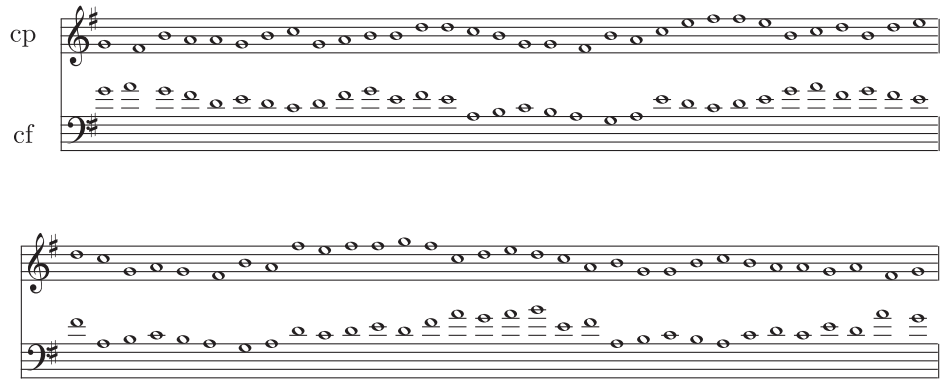


Figure 8. Generated counterpoint with score of 0.371394.

the GA finds slightly better solutions in the beginning of the run, the VNS finds a better solution than the GA, almost immediately. The GA does not converge, since duplicate population members are not allowed. This is confirmed by the fact that the algorithm is still able to make small improvements to the quality of the best solution long into the GA run, for example from 2.83698 to 2.8234 after 984 seconds. In Figure 6, the best-found solutions of the algorithms are plotted against the number of times a solution is evaluated. This graph confirms the previous conclusions. Overall, the GA and the Random search are completely outperformed by the VNS.

## 6. Conclusions and further research

In this article, an efficient VNS algorithm has been developed to automatically 'compose' musical fragments consisting of melodies for the cantus firmus and the first species counterpoint. To this end the rules of first species counterpoint have been quantified and used as an objective function in a local search algorithm. The different parameter settings of the VNS were extensively analysed by means of a full factorial experiment, which resulted in a set of optimal parameter settings. A comparison with a random search and a GA confirmed its efficiency. The resulting algorithm was then implemented in a user-friendly way. The musical output of the VNS has a good objective score. The fragments could, even at this point, be used by a composer as a starting point for a musical composition. Eventually, the authors hope to improve Optimuse to generate more complete, directed and full musical pieces that need little or no correction by a human composer. In this way, the generated compositions could for instance be used as an endless stream of royalty-free music.

An example of the music that is generated by the VNS with optimal settings is displayed in Figure 8. With an objective score as low as 0.371394, this fragment is very close to violating none of the Fuxian rules for counterpoint. Although this is a subjective interpretation, the fragment is pleasant to the ear and sounds a lot less 'random' than the fragment generated as the initial solution. This and other demo pieces are available online and can be downloaded at antor.ua.ac.be/optimuse.

The downside to using these highly restrictive rules is that they are also very limiting. In addition to the obvious limitation of using only whole notes, there are no rules enforcing a theme or coherence in the music, which causes a sense of meandering, especially in longer fragments.

An interesting extension of this work would be to evaluate different styles and types of music. A rhythmic component can be added to the music, by working with other species of counterpoint, such as florid counterpoint. The number of parts can also be increased, to allow more voices at the same time. Three part counterpoint offers a logical starting point for this expansion. It might also be interesting to evaluate music from a larger perspective and add a sense of direction or theme.

Another possible extension of the existing objective function is to add composer specific characteristics. Manaris et al. [30] developed a set of 10 composer specific metrics by scanning musical databases. These metrics, all based on Zipf's law, include the frequency distribution of pitch, duration, harmonic and melodic intervals. An artificial neural network was used to classify pieces in terms of authorship and style. While they are adequate for composer classification, these criteria alone do not seem to be sufficient for generating aesthetically pleasing music [31]. A combination of similar composer-specific simple metrics with the objective function developed in this article might offer

an interesting approach to evaluating composer-specific classical music.

## References


[1] K. Adiloglu and F.N. Alpaslan, *A machine learning approach to two-voice counterpoint composition*, Knowledge-Based Syst. 20 (2007), pp. 300–309.

[2] G. Aguilera, J. Luis Galán, R. Madrid, A.M. Martńnez, Y. Padilla, and P. Rodrńguez, *Automated generation of contrapuntal musical compositions using probabilistic logic in derive*, Math. Comput. Simulat. 80 (2010), pp. 1200–1211.

[3] A. Alpern, Techniques for algorithmic composition of music. 95 (1995), pp. 1–17, Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi = 10.1.1.23.9364

[4] T. Anders, *Composing music by composing rules: Design and usage of a generic music constraint system*, PhD thesis, Queen's University Belfast, 2007.

[5] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, *Computer-assisted composition at ircam: from patchwork to openmusic*, Comput. Music J. 23 (1999), pp. 59–72.

[6] C. Avanthay, A. Hertz, and N. Zufferey, *A variable neighborhood search for graph coloring*, Eur. J. Oper. Res. 151 (2003), pp. 379–388.

[7] J.A. Biles, *Autonomous Genjam: Eliminating the Fitness Bottleneck by Eliminating Fitness*, Proceedings of the GECCO-2001 Workshop on Non-routine Design with Evolutionary Systems, Vol. 7, Morgan Kaufmann, San Francisco, CA, USA, 2001.

[8] C. Blum and A. Roli, *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*, ACM Comput. Surv. (CSUR) 35 (2003), pp. 268–308.

[9] G. Boenn, M. Brain, M. De Vos, and J. Ffitch, *Automatic composition of melodic and harmonic music by answer set programming*, Logic Program. 5366 (2009), pp. 160–174.

[10] C. Burns, *Designing for Emergent Behavior: A John Cage Realization*, Proceedings of the ICMC, Miami, 2004.

[11] A.R. Burton and T. Vladimirova, *Generation of musical sequences with genetic techniques*, Comput. Music J. 23 (1999), pp. 59–73.

[12] G. Carpentier, G. Assayag, and E. Saint-James, *Solving the musical orchestration problem using multiobjective constrained optimization with a genetic local search approach*, J. Heurist. 16 (2010), pp. 1–34.

[13] D. Cope, *New Directions in Music*, Waveland Press, Long Grove, IL, 2000.

[14] D. Cope, *A musical learning algorithm*, Comput. Music J. 28 (2004), pp. 12–27.

[15] T. Davidović, P. Hansen, and N. Mladenović, *Permutation-based genetic, tabu, and variable neighborhood search heuristics for multiprocessor scheduling with communication delays*, Asia-Pacific J. Operat. Res. (APJOR) 22 (2005), pp. 297–326.

[16] R. DePrisco, A. Eletto, A. Torre, and R. Zaccagnino, *A neural network for bass functional harmonization*, Appl. Evol. Comput. 6025 (2010), pp. 351–360.

[17] P. Donnelly and J. Sheppard, *Evolving four-part harmony using genetic algorithms* in *EvoApplications (2)*, Vol. 6625, of Lecture Notes in Computer Science, A. Cecilia Di Chio, G. A. Brabazon, Di Caro, M. Rolf Drechsler, J. Farooq, G. Grahl, C. Greenfield, J. Prins, G. Romero, E. Squillero, A. G. B. Tarantino, N. Tettamanzi, A. Urquhart, and Şima Uyar, eds., Springer, Berlin, 2011, pp. 273–282.

[18] K. Ebcioğlu, *An expert system for harmonizing four-part chorales*, Comput. Music J. 12 (1988), pp. 43–51.

[19] W. Fetterman, *John Cage's Theatre Pieces: Notations and Performances*, Routledge, Abingdon, UK, 1996.

[20] K. Fleszar and K.S. Hindi, *Solving the resource-constrained project scheduling problem by a variable neighbourhood search*, Eur. J. Oper. Res. 155 (2004), pp. 402–413.

[21] J.J. Fux and A. Mann, *The Study of Counterpoint from Johann Joseph Fux's Gradus Ad Parnassum - 1725*, Norton, 1971.

[22] M. Geis and M. Middendorf, *An ant Colony Optimizer for Melody Creation with Baroque Harmony*, pp. 461–468, IEEE Congress on Evolutionary Computation, 2007.

[23] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, London, 1993.

[24] H. Norden, *Fundamental Counterpoint*, Crescendo Publishing Co., Boston, 1969.

[25] P. Hansen and N. Mladenović, *Variable neighborhood search: Principles and applications*, Eur. J. Operat. Res. 130 (2001), pp. 449–467.

[26] P. Hansen, N. Mladenović, and D. Perez-Britos, *Variable neighborhood decomposition search*, J. Heurist. 7 (2001), pp. 335–350.

[27] A. Horner and D.E. Goldberg, *Genetic algorithms and computer-assisted music composition*, Urbana 51 (1991), pp. 437–441.

[28] D. Horowitz. *Generating rhythms with genetic algorithms*. Proceedings of the International Computer Music Conference, pp. 142–143. International Computer Music Association, San Francisco, 1994.

[29] J. Kytöjoki, T. Nuortio, O. Bräysy, and M. Gendreau, *An efficient variable neighborhood search heuristic for very large scale vehicle routing problems*, Comput. Operat. Res. 34 (2007), pp. 2743–2757.

[30] B. Manaris, P. Machado, C. McCauley, J. Romero, and D. Krehbiel, *Developing fitness functions for pleasant music: Zipf's law and interactive evolution systems*, Appl. Evol. Comput. 3449 (2005), pp. 498–507.

[31] B. Manaris, D. Vaughan, C. Wagner, J. Romero, and R.B. Davis. *Evolutionary Music and the Zipf-Mandelbrot Law: Developing fitness functions for pleasant music*. Proceedings of the 2003 International Conference on Applications of Evolutionary Computing, pp. 522–534. Springer-Verlag, Berlin, 2003.

[32] R.A. McIntyre, *Bach in a box: The evolution of four part baroque harmony using the genetic algorithm*,

Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence, Orlando, FL, pp. 852–557, IEEE, Washington, D.C., 1994.
[33] N. Mladenovic and P. Hansen, *Variable neighborhood search*, Comput. Operat. Res. 24 (1997), pp. 1097–1100.
[34] A.F. Moore, *Categorical conventions in music discourse: Style and genre*, Music Lett. 82 (2001), pp. 432–442.
[35] A. Moroni, J. Manzolli, F.V. Zuben, and R. Gudwin, *Vox populi: An interactive evolutionary system for algorithmic music composition*, Leonardo Music J. 10 (2000), pp. 49–54.
[36] G. Nierhaus, *Algorithmic Composition: Paradigms of Automated Music Generation*, Springer, Vienna, 2009.
[37] G. Papadopoulos and G. Wiggins, *Ai Methods for Algorithmic Composition: A Survey, A Critical View and Future Prospects*, AISB Symposium on Musical Creativity, pp. 110–117, Edinburgh, 1999.
[38] S. Phon-Amnuaisuk, A. Tuson, and G. Wiggins. *Evolving Musical Harmonization*. Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Portorož, Slovenia, p. 229. Springer Verlag Wien, 1999.
[39] S. Phon-Amnuaisuk and G. Wiggins, *The Four-part Harmonization Problem: A Comparison Between Genetic Algorithms and a rule-based system*, pp. 28–34. Proceedings of the AISB'99 Symposium on Musical Creativity, 1999.
[40] J. Polito, J.M. Daida, and T.F. Bersano-Begey, *Musica ex Machina: Composing 16th-century counterpoint with genetic programming and symbiosis*, *Evolutionary Programming*, Vol. 1213, of Lecture Notes in Computer Science, pp. 113–124, Springer, Berlin, 1997.
[41] J. Rothgeb, *Strict counterpoint and tonal theory*, J. Music Theory 19 (1975), pp. 260–284.
[42] F. Salzer and C. Schachter, *Counterpoint in Composition: The Study of Voice Leading*, Columbia University Press, 1969.
[43] O. Sandred, M. Laurson, and M. Kuuskankare, *Revisiting the illiac suite – a rule-based approach to stochastic processes*, Sonic Ideas/Ideas Sonicas 2 (2009), pp. 42–46.
[44] K. Sörensen and F. Glover, *Metaheuristics*, in *Encyclopedia of Operations Research and Management Science*, 3rd ed., S.I. Gass and M.C. Fu, eds., Springer, New York, (in press), pp. 1–12.
[45] P.M. Todd and G.M. Werner, *Frankensteinian methods for evolutionary music composition*, in *Musical networks: Parallel Distributed Perception and Performance*, P. M. Todd and N. Griffith, eds., MIT Press/Bradford Books, Cambridge, MA, 1998.
[46] N. Tokui and H. Iba. *Music Composition with Interactive Evolutionary Computation*. Proc. Third Int. Conf. Generat. Art, 17 (2000), pp. 215–226.
[47] M.W. Towsey, A.R. Brown, S.K. Wright, and J. Diederich, *Towards melodic extension using genetic algorithms*, Educ. Technol. Soc. 4 (2001), pp. 54–65.
[48] I. Xenakis, *Formalized Music: Thought and Mathematics in Composition*, Number 6, Pendragon Press, Hillsdale, NJ, 1992.

## Appendix A. Detailed breakdown of objective function

### *Horizontal score*

These rules apply to a tonal cantus firmus consisting of $L \times 16$ whole notes. L is the length of the fragment expressed in units of 16 notes.

(1) Notes should not be repeated immediately. In the case of counterpoint, maximum four tied notes per L are allowed
(2) Only consonant horizontal intervals are permitted between consecutive notes. Horizontal intervals of 0, 1, 2, 3, 4, 5, 7, 8, 9, 12, 15 and 16 semitones are consonant. The largest allowed consonant interval between two *consecutive* notes is the perfect octave.
(3) Conjunct: stepwise movement should predominate, 2–4 leaps per L are optimal. Stepwise motion is an interval of 1 or 2 semitones. If the context is prevailing stepwise (90%), an exception can be made.
(4) Each large leap should be followed by stepwise motion. A *leap* is an interval of more than two semitones. A *large leap* is an interval of more than four semitones.
(5) Change direction after each large leap.
(6) There should be one climax (highest note).
(7) Climax should be melodically consonant with tonic.
(8) No more than two large leaps per L.
(9) No more than two consecutive leaps.
(10) Do not move too long, stepwise, in same direction.
(11) The direction needs to change several times (mininum three times per L).
(12) The starting note should be the tonic. In the case of counterpoint, the starting note can also be the dominant.
(13) The ending note should be tonic and in same octave as starting note.
(14) The penultimate note should be the leading tone for CP and the supertonic for CF. The leading tone is the 7th note of the scale. In the case of a minor scale it should be raised by half a tone. When the previous note is the sixth note of the scale, it is also allowed to be raised by half a tone.
(15) The beginning and the end of all motion should be consonant. A motion interval is the interval between the start and end note of an upward or downward movement.
(16) There should be a good balance between ascending and descending motion. Large motion intervals ($>$ nine semitones) should be avoided.
(17) No frequent repetition of a single note (maximum three times per L).
(18) No repetition of sequences within a 16 note interval. A sequence is a group of at least two notes.

$$subscore_1^H(s) = \begin{cases} \frac{\#\text{repeated notes}}{\text{L}} & \text{in case of CF and if } \#\text{repeated notes} \le L, \\ 1 & \text{in case of CF and if } \#\text{repeated notes} > L, \\ \frac{\#\text{2 repeated notes} - 2\times\text{L}}{\text{length - 2}\times\text{L}} & \text{in case of CP.} \end{cases} \tag{A1}$$

$$subscore_2^H(s) = 1 - \frac{\#\text{consonant intervals}}{\#\text{intervals}} \tag{A2}$$

$$subscore_3^H(s) = \begin{cases} 0 & \text{if } 2 \le \#\text{leaps} \le 4\times\text{L}, \\ \frac{\#\text{leaps} - (4\times\text{L})}{\text{length} - 1 - (4\times\text{L})} & \text{if } 4\times\text{L} < \#\text{leaps}, \\ \frac{\#\text{leaps}}{\text{length} - 1 - (4\times\text{L})} & \text{if } 2 > \#\text{leaps}. \end{cases} \tag{A3}$$

$$subscore_4^H(s) = \begin{cases} 0 & \text{if} \ge 90\% \text{ of intervals are one or two semitones,} \\ \frac{\#\text{large leaps not foll. by stepw. mot.}}{\#\text{large leaps}} & \text{if} < 90\% \text{ of intervals are one or two semitones.} \end{cases} \tag{A4}$$

$$subscore_5^H(s) = \frac{\#\text{large leaps not followed by}}{\#\text{large leaps}} \tag{A5}$$

$$subscore_6^H(s) = \frac{\#\text{occurences of highest note} - 1}{\text{length}} \tag{A6}$$

$$subscore_7^H(s) = \begin{cases} 0 & \text{if climax is consonant with tonic,} \\ 1 & \text{if climax is } \textit{not} \text{ consonant with tonic.} \end{cases} \tag{A7}$$

$$subscore_8^H(s) = \begin{cases} 0 & \text{if } \#\text{large leaps} \le 2\times\text{L}, \\ \frac{\#\text{large leaps} - (2\times\text{L})}{\text{length} - 1 - (2\times\text{L})} & \text{if } \#\text{large leaps} > 2\times\text{L}. \end{cases} \tag{A8}$$

$$subscore_9^H = \frac{\#\text{ of three consecutive leaps}}{\text{length} - 3} \tag{A9}$$

$$subscore_{10}^H(s) = \begin{cases} 0 & \text{if longest stepwise sequence} \le 5, \\ \frac{\text{length of longest stepwise sequence-5}}{\text{length} - 6} & \text{if longest stepwise sequence} > 5. \end{cases} \tag{A10}$$

$$subscore_{11}^H(s) = \begin{cases} 0 & \text{if } \#\text{ direction changes} \ge 3\times\text{L}, \\ 1 - \frac{\#\text{ direction changes}}{3\times\text{L}} & \text{if } \#\text{ direction changes} < 3\times\text{L}. \end{cases} \tag{A11}$$

$$subscore_{12}^H(s) = \begin{cases} 0 & \text{if start note is tonic (or dominant in the case of CP),} \\ 1 & \text{if start note is not tonic.} \end{cases} \tag{A12}$$

$$subscore_{13}^H(s) = \begin{cases} 0 & \text{if end note is tonic in same octave as start note,} \\ 0.5 & \text{if end note is tonic , but is not equal to start note,} \\ 1 & \text{if end note is not tonic.} \end{cases} \tag{A13}$$

$$subscore_{14}^H(s) = \begin{cases} 0 & \text{if pen. is leading note (CP) or supertonic (CF) in same oct. as end note,} \\ 0.5 & \text{if pen. is leading note (CP) or supertonic (CF) , not in same oct. as end note,} \\ 1 & \text{if pen. is not leading note (CP) or supertonic (CF).} \end{cases} \tag{A14}$$

$$subscore_{15}^H(s) = \frac{\#\text{times that start and end of motion are dissonant}}{\#\text{motion intervals}} \tag{A15}$$

$$subscore_{16}^H(s) = \frac{\#\text{motion intervals} > \textit{nine} \text{ semitones}}{\#\text{motion intervals}} \tag{A16}$$

$$subscore_{17}^H(s) = \frac{\#\text{notes repeated more than } (3\times\text{L}) \text{ times}}{\text{length}} \tag{A17}$$

$$subscore_{18}^H(s) = \frac{\#\text{notes that are part of a rep. seq. within a 16 note interval}}{\text{length}} \tag{A18}$$

*Vertical score*

(1) Only consonant vertical intervals are permitted. Vertical consonance is determined by the tonal distance between the cantus firmus (CF) and the counterpoint (CP). Vertical intervals of 0, 3, 4, 7, 8, 9, 12, 15 and 16 semitones are consonant.
(2) All perfect intervals should be approached by contrary or oblique motion. Intervals of 0, 5, 7 and 12 semitones are considered perfect. There are four types of motion:

- Contrary motion: CP and CF move in a different direction.
- Similar motion: CP and CF move in the same direction.
- Parallel motion: CP and CF move in the same direction at a constant (or virtually constant) distance. Thirds, sixths and tenths are considered parallel, even if the quality of the interval changes (minor vs. major).
- Oblique motion: one voice remains stationary and the other voice moves.

(3) Avoid the overlapping of parts. Overlap occurs when the CF is one or two semitones higher than the previous note of the CP. Or when the CP is one or two semitones lower than the previous note of the CF.
(4) The maximum allowed distance between voices is one tenth (16 semitones). With the exception of the climax.
(5) There should be no crossing. Crossing occurs when the lower voice is above the upper voice.
(6) Unison is only allowed in the beginning and end.
(7) Avoid leaps simultaneously in the cantus firmus and counterpoint. Especially leaps larger than one fourth (six semitones) in the same direction.
(8) Use all types of motion.
(9) From unison to octave (and vice versa) is forbidden.
(10) The ending should be unison or octave.
(11) The beginning should be unison, octave or fifth above cantus.
(12) Contrary motion should predominate slightly.
(13) Thirds, sixths and tenths should predominate.
(14) There can be no more than three successive parallel thirds, sixths and tenths.
(15) The climax (note with the highest pitch) of the CF and CP should not coincide.

$$subscore_1^V(s) = 1 - \frac{\#\text{consonant intervals}}{\#\text{intervals}} \tag{A19}$$

$$subscore_2^V(s) = \begin{cases} \dfrac{\#\text{perfect intervals not appr. by contr. or obl. motion}}{\#\text{perfect intervals}} \\ \quad \text{if } \#\text{perfect intervals} \neq 0, \\ 0 \\ \quad \text{if } \#\text{perfect intervals} = 0 \end{cases} \tag{A20}$$

$$subscore_3^V(s) = \frac{\#\text{overlaps}}{\#\text{length-1}} \tag{A21}$$

$$subscore_4^V(s) = \frac{\#\text{distances} > 16 \text{ semitones}}{\text{length}} \tag{A22}$$

$$subscore_5^V(s) = \frac{\#\text{crossed notes}}{\text{length-1}} \tag{A23}$$

$$subscore_6^V(s) = \frac{\#\text{unisons (excl beginning and end)}}{\text{length-2}} \tag{A24}$$

$$subscore_7^V(s) = \frac{\left\{ \begin{array}{l} (\#\text{sim leaps not in same dir}) + 2 \times (\#\text{sim leaps} \\ \quad > \text{six semitones in same dir}) \end{array} \right\}}{2 \times \#\text{intervals}} \tag{A25}$$

$$subscore_8^V(s) = 1 - \frac{\#\text{different types of motion used}}{4} \tag{A26}$$

$$subscore_9^V(s) = \frac{\#\text{motion from unison to octave and vsvs}}{\text{length}} \tag{A27}$$

$$subscore_{10}^V(s) = \begin{cases} 1 & \text{if end is not unison or octave,} \\ 0 & \text{if end is unison or octave.} \end{cases} \tag{A28}$$

$$subscore_{11}^V(s) = \begin{cases} 1 & \text{if beginning is not unison , octave or fifth above cantus,} \\ 0 & \text{if beginning is unison, octave or fifth above cantus.} \end{cases} \tag{A29}$$

$$subscore_{12}^V(s) = \begin{cases} 1 & \text{if } \#\text{ contrary motion} \leq \text{all other motion,} \\ 0 & \text{if } \#\text{ contrary motion} > \text{all other motion.} \end{cases} \tag{A30}$$

$$subscore_{13}^V(s) = \begin{cases} 0 & \text{if } \#\text{ thirds, sixths and tenths} \geq 50\%, \\ 1 & \text{if } \#\text{ thirds, sixths and tenths} < 50\%. \end{cases} \tag{A31}$$

$$subscore_{14}^V(s) = \frac{\#\text{sequences of parallel thirds , sixths and tenths} > 3}{\text{length-3}} \tag{A32}$$

$$subscore_{15}^V(s) = \begin{cases} 1 & \text{if the climax of the CF and CP coincide,} \\ 0 & \text{if the climax of the CF and CP do not coincide.} \end{cases} \tag{A33}$$

Table B1. Multi-way ANOVA model with interactions (CF).

| Parameter | Df | $F$ value | Prob ($>F$) |
|---|---|---|---|
| $N_1$ | 1 | 3685.4635 | $<2.2\times10^{-16}$* |
| $N_2$ | 1 | 6175.4618 | $<2.2\times10^{-16}$* |
| $N_3$ | 1 | 12,247.2613 | $<2.2\times10^{-16}$* |
| $tt_1$ | 2 | 0.6242 | 0.535733 |
| $tt_2$ | 2 | 10.4974 | $2.837\times10^{-05}$* |
| $tt_3$ | 2 | 10.8363 | $2.025\times10^{-05}$* |
| Randsize | 2 | 1865.4689 | $<2.2\times10^{-16}$* |
| Adj. weights | 1 | 328.102 | $<2.2\times10^{-16}$* |
| Iters | 2 | 237.2371 | $<2.2\times10^{-16}$* |
| Length | 3 | 121.4919 | $<2.2\times10^{-16}$* |
| $N_1$:$N_2$ | 1 | 7808.7883 | $<2.2\times10^{-16}$* |
| $N_1$:$N_3$ | 1 | 9563.5288 | $<2.2\times10^{-16}$* |
| $N_2$:$N_3$ | 1 | 18,704.564 | $<2.2\times10^{-16}$* |
| $N_1$:randsize | 2 | 39.3468 | $<2.2\times10^{-16}$* |
| $N_2$:randsize | 2 | 252.9978 | $<2.2\times10^{-16}$* |
| $N_3$:randsize | 2 | 872.0523 | $<2.2\times10^{-16}$* |
| $N_1$:adj. weights | 1 | 1.6866 | 0.1941202 |
| $N_2$:adj. weights | 1 | 2.4398 | 0.1183666 |
| $N_3$:adj. weights | 1 | 4.5202 | 0.033557* |
| Randsize:adj. weights | 2 | 274.9924 | $<2.2\times10^{-16}$* |
| $N_1$:iters | 2 | 41.4913 | $<2.2\times10^{-16}$* |
| $N_2$:iters | 2 | 108.369 | $<2.2\times10^{-16}$* |
| $N_3$:iters | 2 | 212.4846 | $<2.2\times10^{-16}$* |
| Randsize:iters | 4 | 8.8365 | $4.225\times10^{-07}$* |
| Adj. weights:iters | 2 | 6.0082 | 0.0024806* |
| $N_1$:length | 3 | 45.7464 | $<2.2\times10^{-16}$* |
| $N_2$:length | 3 | 6.4358 | 0.0002411* |
| $N_3$:length | 3 | 31.5985 | $<2.2\times10^{-16}$* |
| Randsize:length | 6 | 2.7547 | 0.0112953* |
| Adj. weights:length | 3 | 0.5819 | 0.6268833 |
| Iters:length | 6 | 0.3941 | 0.8832212 |
| $N_1$:$N_2$:$N_3$ | 1 | 23,470.7999 | $<2.2\times10^{-16}$* |
| $N_1$:$N_2$:randsize | 2 | 124.8853 | $<2.2\times10^{-16}$* |
| $N_1$:$N_3$:randsize | 2 | 123.0317 | $<2.2\times10^{-16}$* |
| $N_2$:$N_3$:randsize | 2 | 817.5753 | $<2.2\times10^{-16}$* |
| $N_1$:$N_2$:adj. weights | 1 | 2.4761 | 0.1156635 |
| $N_1$:$N_3$:adj. weights | 1 | 0.0013 | 0.9709671 |
| $N_2$:$N_3$:adj. weights | 1 | 0.1372 | 0.711064 |
| $N_1$:randsize:adj. weights | 2 | 6.4709 | 0.001564* |
| $N_2$:randsize:adj. weights | 2 | 0.2066 | 0.8133344 |
| $N_3$:randsize:adj. weights | 2 | 10.064 | $4.367\times10^{-05}$* |
| $N_1$:$N_2$:iters | 2 | 99.6471 | $<2.2\times10^{-16}$* |
| $N_1$:$N_3$:iters | 2 | 95.4637 | $<2.2\times10^{-16}$* |
| $N_2$:$N_3$:iters | 2 | 345.7817 | $<2.2\times10^{-16}$* |
| $N_1$:randsize:iters | 4 | 6.0981 | $6.875\times10^{-05}$* |
| $N_2$:randsize:iters | 4 | 2.7069 | 0.0287158* |
| $N_3$:randsize:iters | 4 | 7.4022 | $6.164\times10^{-06}$* |
| $N_1$:adj. weights:iters | 2 | 0.25 | 0.7787971 |
| $N_2$:adj. weights:iters | 2 | 0.4876 | 0.614123 |
| $N_3$:adj. weights:iters | 2 | 3.1004 | 0.0451376* |
| Randsize:adj. weights:iters | 4 | 5.3779 | 0.0002567* |
| $N_1$:$N_2$:length | 3 | 91.6277 | $<2.2\times10^{-16}$* |
| $N_1$:$N_3$:length | 3 | 101.2698 | $<2.2\times10^{-16}$* |
| $N_2$:$N_3$:length | 3 | 11.3937 | $1.942\times10^{-07}$* |
| $N_1$:randsize:length | 6 | 4.2001 | 0.0003224* |
| $N_2$:randsize:length | 6 | 16.7981 | $<2.2\times10^{-16}$* |

(*continued*)

Table B1. Continued.

| Parameter | Df | $F$ value | Prob ($>F$) |
|---|---|---|---|
| $N_3$:randsize:length | 6 | 21.245 | $<2.2\times10^{-16}$* |
| $N_1$:adj. weights:length | 3 | 0.2673 | 0.8490325 |
| $N_2$:adj. weights:length | 3 | 0.1774 | 0.9117525 |
| $N_3$:adj. weights:length | 3 | 0.5095 | 0.6757226 |
| Randsize:adj. weights:length | 6 | 1.6344 | 0.1333481 |
| $N_1$:iters:length | 6 | 0.7663 | 0.5963381 |
| $N_2$:iters:length | 6 | 15.7654 | $<2.2\times10^{-16}$* |
| $N_3$:iters:length | 6 | 15.8369 | $<2.2\times10^{-16}$* |
| Randsize:iters:length | 12 | 4.3792 | $5.542\times10^{-07}$* |
| Adj. weights:iters:length | 6 | 0.3186 | 0.9276215 |

Note: The asterisks indicate statistical significance $p<0.05$.

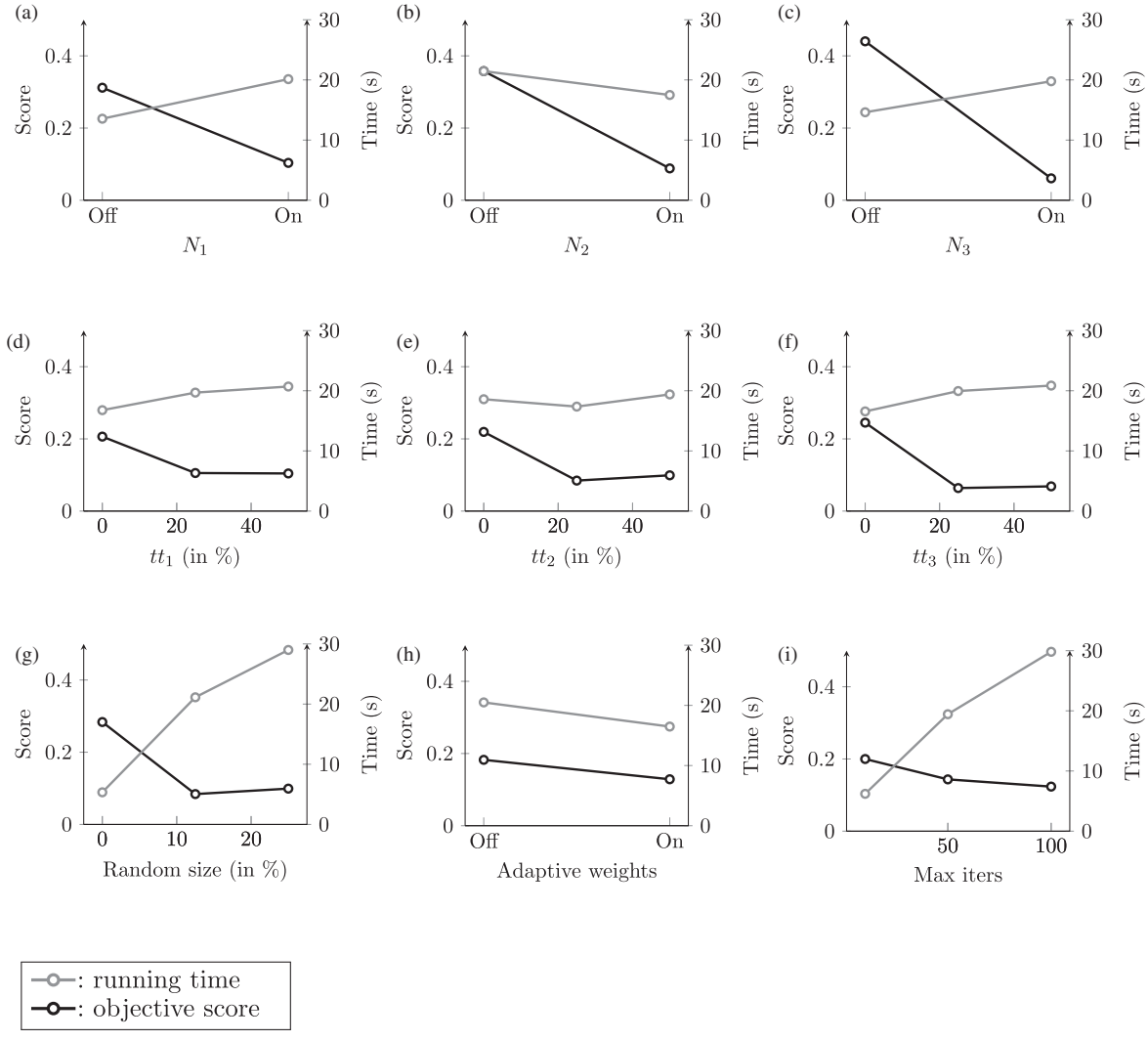


Figure C1. Means plots CF. (a) Swap neighbourhood. (b) Change1 neighbourhood. (c) Change2 neighbourhood. (d) Tabu tenure of Swap. (e) Tabu tenure of Change1. (f) Tabu tenure of Change2. (g) Size of the random perturbation. (h) Adaptive weights procedure. (i) Maximum iterations.

**Appendix D. Means plot for counterpoint.**

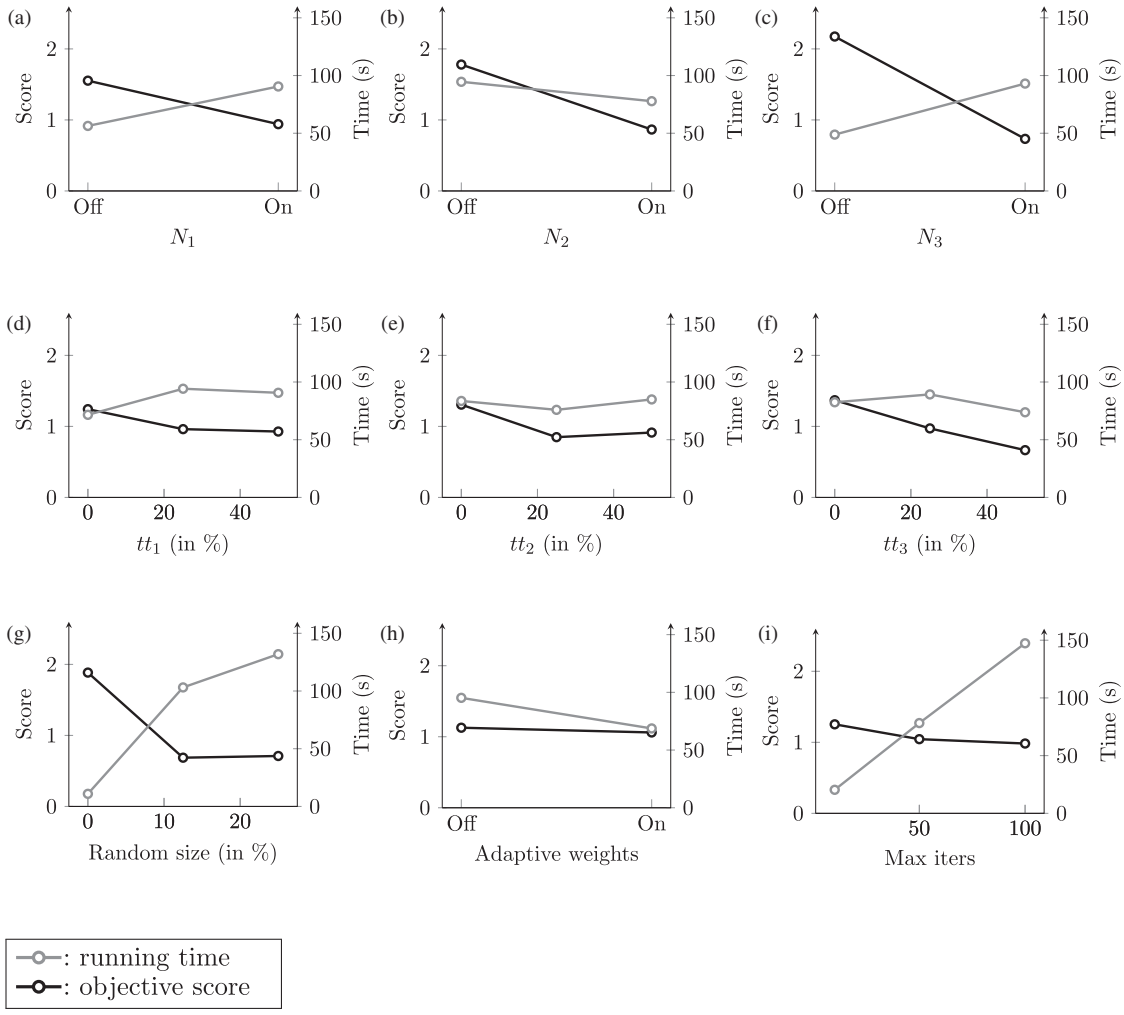


Figure D1. Mean plots CP. (a) Swap neighbourhood. (b) Change1 neighbourhood. (c) Change2 neighbourhood. (d) Tabu tenure of Swap. (e) Tabu tenure of Change1. (f) Tabu tenure of Change2. (g) Size of the random perturbation. (h) Adaptive weights procedure. (i) Maximum iterations.