DEPARTMENT OF ENGINEERING MANAGEMENT

FuX, an Android app that generates counterpoint

Dorien Herremans & Kenneth Sörensen

UNIVERSITY OF ANTWERP
Faculty of Applied Economics

Stadscampus
Prinsstraat 13, B.226
BE-2000 Antwerpen
Tel. +32 (0)3 265 40 32
Fax +32 (0)3 265 47 99
www.ua.ac.be/tew

FACULTY OF APPLIED ECONOMICS

DEPARTMENT OF ENGINEERING MANAGEMENT

FuX, an Android app that generates counterpoint

Dorien Herremans & Kenneth Sörensen

RESEARCH PAPER 2013-003
MARCH 2013

University of Antwerp, City Campus, Prinsstraat 13, B-2000 Antwerp, Belgium
Research Administration – room B.226
phone: (32) 3 265 40 32
fax: (32) 3 265 47 99
e-mail: joeri.nys@ua.ac.be

The papers can be also found at our website:
www.ua.ac.be/tew (research > working papers) &
www.repec.org/ (Research papers in economics - REPEC)

D/2013/1169/003

FuX, an Android app that generates counterpoint*

Dorien Herremans[a†] and Kenneth Sörensen

University of Antwerp
Faculty of Applied Economics
ANT/OR Operations Research Group

February 20, 2013

This paper describes the implementation of an Android application, called FuX[1], that can continuously play a stream of newly generated fifth species counterpoint. A variable neighborhood search algorithm is implemented in order to generate the music. This algorithm is a modification of an algorithm developed previously by the authors to generate musical fragments of a pre-specified length [28]. The changes in the algorithm allow the Android app to play a *continuous* stream of music. The objective function used to evaluate the quality of the fragment is based on a quantification of the extensive rules of this musical style. FuX is a user friendly application that can be installed on any Android phone of tablet.

1 Introduction

In this research an algorithm is implemented that can continuously play newly generated fifth species counterpoint on an Android device. In order to do this, the composition process is modeled as a *combinatorial optimization problem*. The objective is to find the right combination of notes so that the music can be considered as "good". In general, a musical fragment is considered to be good when it fits a certain style as well as possible. The better a fragment fits a style, the better its quality will be. The algorithm that underlies the FuX app, called Optimuse, uses fifth species counterpoint style, a type of polyphonic classical music [17]. In a previous paper, the authors have quantified the extensive rules of this style. This quantification can be used to determine the counterpoint quality of a fragment [28]. A variable neighborhood search (VNS) algorithm called Optimuse was developed and implemented. This algorithm can efficiently generate musical fragments of a pre-specified length on a pc [28]. In this research the existing VNS algorithm was modified to generate a *continuous stream* of new music. It was then ported to the Android platform. The resulting Android app, called FuX, is user friendly and can be installed on any

*Preprint accepted for IEEE Symposium Series on Computational Intelligence
†Corresponding author. Email: dorien.herremans@ua.ac.be

[1]Named after the author of the most influential book on counterpoint [17]

Android phone or tablet. Possible uses include playing an endless stream of classical music to babies. Babies often calm down and experience health benefits from listening to soothing music [51]. It can be conjectured that the highly consonant style of classical counterpoint is especially suitable for this purpose. Moreover, parents who are tired of listening to the same tune thousands of time might prefer the non-repetitiveness of the music generated by FuX. FuX also provides an endless stream of royalty-free music that could be played in elevators, lobbies and as call center waiting music. Finally, it might offer an endless source of inspiration to composers.

The idea that computers could be used to compose music was formed from the very conception of computers. Ada Lovelace, the world's first conceptual programmer [23] already hinted at using computers to automate composition around 1840:

"[The Engine's] operating mechanism might act upon other things besides numbers [...] Supposing, for instance, that the fundamental relations of pitched sounds in the signs of harmony and of musical composition were susceptible of such expressions and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent." – [8]

The research domain "computer assisted composing" or CAC was born in the mid 20th century. One of the earliest compositions made by a computer is the "Illiac Suite" from Lejaren Hiller and Leonard Isaacson in 1957 [50]. They use a rule-based approach to compose music [2]. Other applications soon followed. An extensive overview is given by Burton and Vladimirova [11] and Nierhaus [41].

1.1 Metaheuristics

Composing music is a computationally complex task, since the number of possible fragments increases exponentially with the length of the fragment. For instance, in this paper a set of 10 allowed pitches is defined depending on the selected key. This means that a small fragment of 32 notes, without taking into account rhythmic changes, already has 10^{32} possible combinations of pitches. *Heuristic* or *metaheuristic* optimization techniques are very suitable for this type of combinatorial optimization problem. Metaheuristics do not necessarily return the optimal solution like exact methods [6], but use a variety of strategies to find a good solution in a limited amount of time. There are roughly three categories of metaheuristics [54].

Population-based metaheuristics (evolutionary/genetic algorithms, path relinking,...) usually maintain a set of solutions (population) and *combine* solutions from this set into new ones. The first genetic algorithm applied in the field of music was developed in 1991 [30]. In the following years, many population based algorithms have been developed for CAC. Topics include the generation of jazz solos [5], rhythmic patterns [31, 55], counterpoint style music [14, 35, 45, 46], evolving chords [40], combining fragments for orchestration [13], and others.

Constructive metaheuristics, such as ant colony optimization and GRASP, form a second class of metaheuristics that build a solution from its constituent parts. This category is not as popular as the previous class. In 2007, the first ant colony algorithm was developed for harmonizing baroque music [18].

Local search techniques (tabu search, variable neighborhood search,...) are considered to be a third class of metaheuristics. They iteratively improve a single solution [54]. Music constraint problems have been solved at IRCAM (Institut de Recherche et Coordination Acoustique/Musique) by local search techniques [56]. To the authors knowledge, the variable neighborhood search algorithm for generating counterpoint developed by the authors is the first VNS applied to this problem [29]. In contrast to previous studies which often only optimize a very limited set of rules (such as [14]), a fairly *complete* set of counterpoint rules is used in this research.

1.2 Android

In order to make the implementation of the VNS accessible and easily useable for a large audience, an Android application (or app) called FuX is developed in this research. Android is a software toolkit that runs on a large number of mobile devices. Mobile phones and tablets have never been more popular and are getting increasingly more powerful [36].

There are a plethora of other mobile operating systems available. Symbian from Nokia, Windows Mobile from Microsoft, BlackBerry from RIM, iOS from Apple etc. According to a Survey of Oliver [42] none of these operating systems (including Android) are perfect for developers. The two most used operating systems are iOS and Android [20]. The VNS developed in this research is implemented on the Android system, which allows it to run on a multitude of devices, not only those from Apple, with many of these devices available at a relatively low cost. An added advantage is Android's "open" nature and large support community compared to iOS's lack of developer tools [42]. Google reported that more than 500 million Android devices have been activated [4].

When exploring Google Play[2], the web based platform to easily install new Android applications, the category "music" displays thousands of entries. Many applications have been developed to play music [58], recommend music based on a user profile [32] or a travel location [9], assist in browsing large music libraries [57], finding music by singing/humming [43], and many more. Park and Chung [43] give an extensive overview of music related Android applications.

Since Android 1.0 was only released in 2008 [21], the number of publications on the use of metaheuristics implemented on this platform is still limited. Added to that, the trend to invent different names for similar existing metaheuristics makes it harder to get an overview of the entire field [53]. Fajardo and Oppus [15] have implemented a genetic algorithm for mobile disaster management. Zheng et al. [59] use simulated annealing for WiFi based indoor localization on Android.

In the next sections, the objective function and the implemented VNS algorithm are described in detail. Section 4 explains the implementation (called FuX) of the VNS for the Android platform.

2 Quantifying counterpoint quality

This research focuses on generating fifth species counterpoint music. Johann Fux wrote down the rules for species counterpoint in 1725 in his Gradus at Parnassum [17]. Although every musical style has its rules, these are often not written down as formally as Fux's rules [39]. The rules of counterpoint are considered to be one of the most restrictive sets of rules for composing renaissance music. Fux's system was originally developed as a pedagogical tool for student composers. Therefore strict counterpoint consists of five *species* or levels (first, second, third, fourth and the most advanced is called *florid* counterpoint) which are all taught in sequence. With each level, more complexity is added to the music, e.g., different rhythmical structure. The rules written down by Fux are fundamental in music pedagogy, even today [24]. The fact that they are reducible to a set of simple rules [48] makes it easy to include them as quantifiers of quality in an objective function.

The fifth species counterpoint in this research consists of a cantus firmus (CF) and a counterpoint (CP) melody. The cantus firmus is the melody to which the counterpoint is composed. The algorithm described in the next section sequentially generates these two melodies. The counter-

[2] `http://play.google.com`

point rules that evaluate the cantus firmus focus on the melodic properties (i.e., the *horizontal* aspect of the music). The rules for the counterpoint also include the harmonic interplay between the two melodies (i.e., the *vertical* aspect) [1]. Example rules are "each large leap should be followed by stepwise motion in the opposite direction" and "all perfect intervals should be approached by contrary or oblique motion". All of the Fuxian rules based on Salzer and Schachter [49] were quantified and reduced to a *subscore* between 0 and 1. The lower the score, the better the fragment adheres to the rule. Therefore, the objective of the VNS is to *minimize* $f_m(s)$, whereby $m = cf$ in the case of cantus firmus and $m = cp$ for the counterpoint. A full description of the quantification of the 19 melodic and 19 harmonic subscores is given in [28].

The resulting objective function $f_m(s)$ is used to evaluate how well a fragment s fits into the counterpoint style is represented in eqs. (1) and (2).

$$f_{cf}(s) = \underbrace{\sum_{i=1}^{19} a_i \cdot \text{subscore}_i^H(s)}_{\text{horizontal aspect}} \tag{1}$$

$$f_{cp}(s) = \underbrace{\sum_{i=1}^{19} a_i \cdot \text{subscore}_i^H(s)}_{\text{horizontal aspect}} + \underbrace{\sum_{j=1}^{19} b_j \cdot \text{subscore}_j^V(s)}_{\text{vertical aspect}} \tag{2}$$

The relative importance of a subscore can be set by its weight. Weights a_i and b_i can be set by the user to emphasize the importance of certain rules. The default setting considers all scores to be equally important. Therefore all default weights are set to 1.

The rules mentioned above can be seen as "soft" rules. Although the VNS will try to minimize these subscores as well as possible, it is allowed that some rules are "broken". In fact, given the large number of rules, it is quite unlikely that a fragment can be found that can satisfy them all at the same time for a piece of arbitrary length.

These soft rules are extended with a set of "hard" rules. Hard rules are implemented as constraints and can not be violated like soft rules. When any of the hard rules is violated, the fragment is considered infeasible. Table 1 lists the implemented hard rules.

Table 1: Feasibility criteria

No	Feasability criterium
1	All notes come from the correct key.
2	Only certain rhythmic patterns are allowed for a measure.
3	No rhythmic pattern can be repeated immediately or used excessively.
4	The first measure is a half rest followed by a half note.
5	The penultimate measure is a tied quarter note, followed by two eight notes and a half note.
6	The last measure is a whole note.
7	Ties can only exist between measures and notes of the same pitch.
8	A half note can be tied to a half note or a quarter note.
9	Maximum two measures of the same note value (duration) are allowed. Variations with eight notes do not count.

In the next section, a variable neighborhood search algorithm is developed that uses $f_m(s)$ as the objective function and the hard rules as feasibility constraints.

3 Variable neighborhood search

Most of the available literature in the domain of CAC uses population-based algorithms (very often genetic/evolutionary algorithms) to generate music. The "black-box" character of these algorithms makes them appealing and easy to implement [30]. However, because they do not rely on a specific problem structure, they also fail to use it to their advantage. Phon-Amnuaisuk and Wiggins [44] compare a rule-based system with a genetic algorithm that was developed for harmonizing four-part monophonic tonal music. Their conclusion was that the rule-based system outputs superior quality compared to the genetic algorithm and conclude that "The output of any system is fundamentally dependent on the overall knowledge that the system (explicitly and implicitly) possesses" [44]. Since local search metaheuristics take into account problem-specific knowledge, the previous claim supports that they might be more efficient. Local search algorithms have been proven efficient in many different fields such as vehicle routing and scheduling [54].

3.1 VNS

Variable neighborhood search, or VNS, is a local search strategy. It starts from an initial solution s and iteratively makes small improvements (or *moves*) to the solution in order to find a better one, i.e., a solution with a better objective function value. The set of all solutions s' that can be reached from the *current* solution by making one move is called the *neighborhood* $N(s)$. The local search always selects a solution with a better objective function value than the new current solution. This process goes on until no better solution can be found in the neighborhood, at which point the search has arrived in a local optimum and the VNS strategy switches to a different type of neighborhood. This will allow the search to escape the local optimum [37]. When none of the neighborhoods are able to let the search escape from the local optimum, the VNS uses a perturbation strategy whereby a relatively large part of the current solution is randomized. This second strategy will allow the search to continue again [26].

The first implementations of variable neighborhood search stem from the late 90s. The technique has been successfully applied eversince. Its applications can be found in a wide range of combinatorial problems [25] including vehicle routing [10], project scheduling [16], finding extremal graphs [12], and graph coloring [3]. For several problems, VNS outperforms existing heuristics and is able to find the best solution in moderate computing time [27].

The VNS developed in this research operates in two phases. In the first phase, the cantus firmus is generated. After that, the counterpoint is composed on top of this cantus firmus. The algorithm used to generate the melodies in both phases is identical, it only differs in the objective function that is used. The cantus firmus is evaluated by the objective function that focuses only on melodic rules (eq. (1)). For the counterpoint melody both melodic and harmonic rules are evaluated (eq.(2)). This two-phased design originated from the fact that a counterpoint melody is usually composed "against" an existing cantus firmus and also allows a user to input her own cantus firmus, at least in the original Optimuse implementation [28].

3.2 Components

Figure 1 visualizes the developed VNS. An initial random fragment s is generated whilst taking into account the hard rules specified in the previous section to ensure feasibility.

The core of the VNS algorithm consists of a local search strategy in three neighborhoods. The three neighborhoods are defined by three types of moves (see Figure 2). The change1 move changes the pitch of one note to any other allowed pitch from the key. The change1 neighborhood

(N_1) therefore consists of the set of fragments that can be formed by changing the pitch of any one note to any other allowed pitch. The change2 move expands the previous move by changing two sequential notes to any other allowed pitch. This move is used to generate the change2 neighborhood (N_2) from the current fragment. The first two moves are illustrated in Figures 2(c) and 2(d). The swap neighborhood (N_3) consists of feasible fragments that can be created by swapping the pitch of two notes. An example of a swap move is shown in Figure 2(b).

The VNS starts by performing a local search in the change1 neighborhood. Once the neighborhood is generated, the algorithm selects the fragment s' with the best value for the objective function $f_m(s')$ as the new current fragment. This *steepest descent* strategy ensures a fast improvement of the solution quality. This process is repeated until no better solution can be found in the neighborhood, i.e., a local optimum is reached. When this happens the local search switches to the next neighborhood.

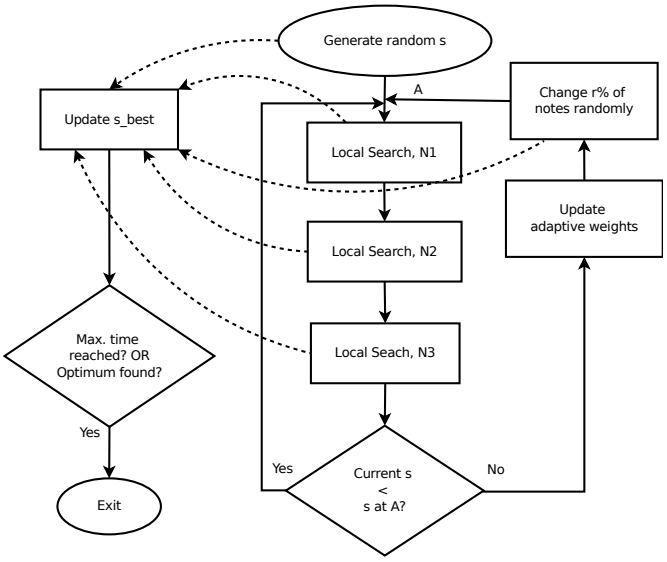


Figure 1: Overview of the developed VNS Algorithm

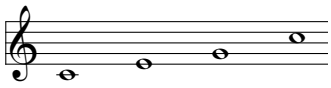
Not all fragments that can be reached by a certain move are included in the corresponding neighborhood. Fragments that violate the hard rules described in the previous section are considered *infeasible* and excluded from the neighborhood. Secondly, moves that change notes on places that are listed on the *tabu list* are also excluded from the neighborhood. A tabu list is a short term memory structure that prevents the algorithm from getting trapped in cycles (i.e., revisiting the same local optimum again and again) [19]. The tabu lists work by storing the places of notes that have been changed in previous iterations and prevents them from being changed again by the same type of move. Each neighborhood N_i has its own tabu list, with its own *tabu tenure* (tt_i). The tenure or length determines the number of iterations that a move remains *tabu active*.

A few mechanisms were added to the VNS in order to help the local search escape from local minima. When no better fragments can be found in any of the neighborhoods of the current fragment, the VNS performs a *perturbation*. This strategy is implemented by reverting back to the global best fragment and changing $r\%$ of the notes to a random pitch from the key.

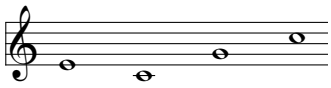
Often, the current fragment reaches the optimum value for a large majority of the subscores, but performs poorly with respect to others. To correct this, the *adaptive weights adjustment*

mechanism is set to action at the same time of the perturbation. This mechanism increases the weight of the highest (i.e., worst) subscore of the objective function by 1. The VNS uses the scores based on these new weights (called the *adaptive score* $f_m^a(s)$) to assess the quality of fragments during the local search. This weights adjustment mechanism increases the likelihood of moves (with an otherwise little impact on the original objective function) in order to improve subscores that are otherwise ignored. To determine whether a fragment should be the new global best solution, the score based on the original weights ($f_m(s)$) is used.

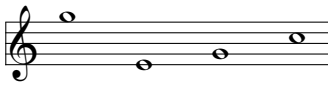
The VNS will keep improving the solution until the maximum time limit is reached or the optimal solution $f_m(s) = 0$ is reached.



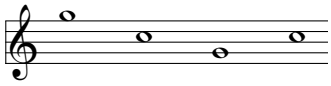
(a) Original



(b) Swap move



(c) Change1 move



(d) Change2 move

Figure 2: Moves

Table 2: neighborhoods

N_i	Name	Description	Neighbourhood size
N_1	Change1	Change one note	$16 \times 9 \times L$
N_2	Change2	Change two sequential notes	$16 \times 9 \times 9 \times L$
N_3	Swap	Swap two notes	$8 \times L \times (16 \times L - 1)$

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

The VNS algorithm described in Figure 1 has a number of parameters that need to be set, such as the size of the random jump and whether neighborhood i is used or not. In order to determine the significant factors and their corresponding optimal settings, a full factorial experiment was conducted. A Multi-Way ANOVA model with interaction effects was constructed with the 2304

instances. The R^2 statistic of the model is 0.98, which means that 98% of the variation around the mean value of the objective function can be explained by the model. The p-values of all of the factors, except the tabu tenure of the change1 neighborhood, are smaller then 0.05, which means these factors have a significant influence on the value of the objective function. The optimal settings were determined by examining the mean and interaction plots. This resulted in the optimal settings displayed in Table 3.

Table 3: Best parameters

Parameter	Values
N_1 - Change1	on with $tt_1 = \frac{1}{16}$
N_2 - Change2	on with $tt_2 = \frac{1}{16}$
N_3 - Swap	on with $tt_3 = \frac{1}{16}$
Random move	$\frac{1}{8}$ changed
Adaptive weights	on
Max. number of iterations	50

The VNS developed by the authors has been compared with a random search and a genetic algorithm. It outperforms both algorithms [29].

One example of static output from the described VNS algorithm is displayed in Figure 3. The score for this music is 0.556776, which is reasonably good, considering the fact that random initial fragment score around 10.

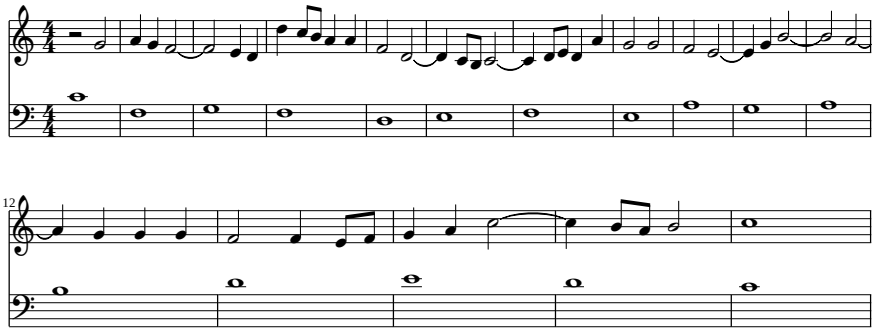


Figure 3: Generated fifth species counterpoint fragment

3.3 From Optimuse to FuX

The original VNS implementation is able to compose fragments of any length, as long as they are a multiple of 16 measures. This version the VNS needs to be able to sequentially generate new fragments that can be considered as one large fragment. The implementation was therefore slightly modified. The VNS is now able to generate 16 measures with a time limit t_1. This generated fragment is used as the starting fragment of the continuously generated piece.

The subsequent fragments that are generated consist of 8 measures and are generated with a time limit t_2. However they are evaluated by also taking into account the last 8 measures of the previous fragment. This means that the VNS always evaluates the last 16 measures. Doing so ensures that no "breaks" in the music occur.

FuX needs to be able to sequentially generate fragments that are played live. Thus, the speed of the algorithm becomes increasingly important, especially since mobile devices often have limited resources such as low-power CPUs, limited RAM and slow I/O [42]. In order to speed up the VNS, the order of the neighborhoods was changed from the previous implementation to the one listed in Table 2. The new order described in this paper favors the smaller neighborhoods because they are often able to make large improvements in the beginning of the run, which ensures that a "reasonable" quality can be obtained fairly quickly.

4 Android implementation

Android is a software toolkit for mobile phones based on the Linux platform developed by Google and the Open Handset Alliance. At the bottom of the Android software stack is the Linux operating system (Kernel 2.6), this provides all basic system functionality such as memory management and device drivers [38]. On top of the OS, there is a set of native libraries written in C/C++ that offer, for instance, audio and video support [21]. The next step in the Android stack contains the runtime engine –the Dalvik Virtual Machine (VM). Dalvik runs applications written in Android's variant of java [7].

Android developers can use the Android Software Development Kit (SDK), to get access to the same framework that is used by the core applications. These powerful libraries allow the development of a wide range of java based applications [21].

Since resources are typically limited on mobile devices, a careful consideration had to be made on how to implement the VNS. Son and Lee [52] recommend the use of Android Native Development Kit (**N**DK) for computationally expensive tasks. This is confirmed by benchmark experiments [34]. Android NDK provides a native development platform that allows embedding components that use native code. With NDK, developers can compile C/C++ code for the Android development platform [47]. Since the previously developed code for the VNS algorithm was in C++, this code could be slightly altered and integrated in the Android app. More details on the original C++ code are described in the authors' previous paper [28].

4.1 Continuous generation

The app developed in this research can continuously generate counterpoint using a VNS. This is achieved by iteratively generating small MIDI files and playing them consecutively. Since the music is played by the device as it is being generated, there should be (at least) two threads running at the same time. A "generate" thread (thread1) and a "playback" thread (thread2). This multithreading approach is described in Table 4. When the app is initialized the VNS algorithm generates the first 16 measures. These are saved as a MIDI file. Whenever the user presses "Play", thread1 generates the next 8 measures whilst thread2 plays the first MIDI file. Directly after the first MIDI file finishes playing, the second MIDI file is played. If the file is not ready yet, thread2 waits for thread1 to finish the generation process. This should be avoided, since it causes an interruption in the playback. This process is repeated until the user pauses or stops it.

The time cutoff for the VNS algorithm is currently set to 10 seconds for the initial generation. This time is divided between the generation of the cantus firmus (3 seconds) and the counterpoint (7 seconds). Because of the complexity of the counterpoint, more time was allotted to its generation. When the cantus firmus reaches an optimum before 3 seconds are passed, the remaining CF time is added to the CP time. The total time is divided by taking into account the following relationship $t_{cp} = 2 \times t_{cf} + 1$. This formula doubles the generation time for the counterpoint and

Table 4: Multithreading

Time	Generate	Playback
-1	16 measures (file 1)	
0s	8 measures (file 2)	file 1
16s	8 measures (file 3)	file 2
24s	8 measures (file 4)	file 3
...	...	...

adds one second to fully exploit the available time. Another relationship might also work, as long as t_{cp} is significantly larger then t_{cf}.

For the generation of the 8 measure fragments, 7 seconds are allocated. This time is divided with the same formula: 2 seconds for the CF and 5 seconds for the CP. The speed of the MIDI playback is set to 1 beat per second. This means that the total time available for generating the file is 8 seconds. The VNS algorithm uses 7 seconds to generate the fragment, which means that 1 second is available as a buffer for actually writing the MIDI file.

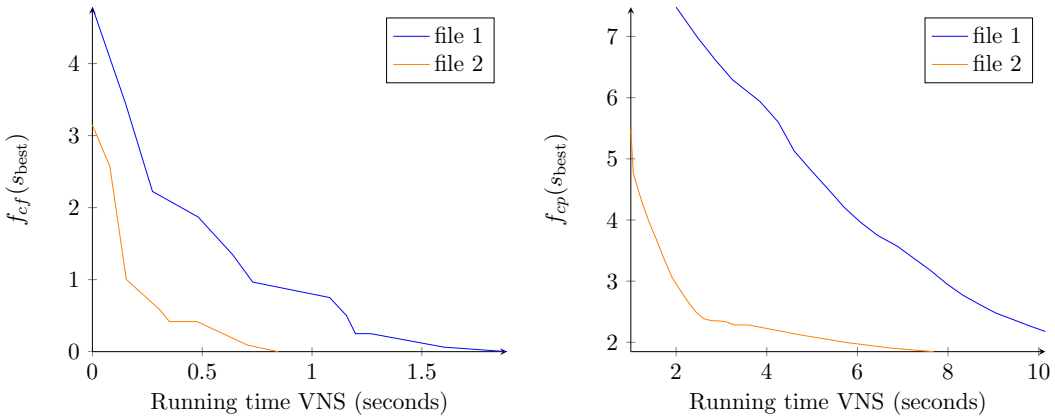
4.2 MIDI files

The VNS algorithm is executed in C++ and returns a native java array. The newly generated music is contained entirely in the *jarray*. This *jarray* is converted to a MIDI file using the library *Android-Midi-Lib* [33]. The MIDI files are stored in the cache folder of the device, so that they are automatically removed periodically. When the VNS is run to generate the next fragment, the previous *jarray* is passed as input to the VNS algorithm, so that it can take into account the previous 8 measures when evaluating the next musical fragment.

Android's *MediaPlayer* class is used to play the MIDI files. The *OnCompletionListener* of this class offers a way to easily play the next MIDI file when playback is finished. Although a small delay between the files might be heard on older Android devices, version 4.1 (Jelly Bean) advertises audio chaining as one of its features [22]. This low latency audio playback enables the files to be played continuously as if they were one big file.

4.3 Implementation and results

Figures 4(a) and 4(b) show the evolution over time of the objective function for cantus firmus and counterpoint. These results were obtained by using an Eclipse Android Virtual Device with Android 4.0.3, ARM processor and 512MB RAM. The emulator was installed on an OpenSuse system with Intel®Core™2 Duo CPU@ 2.20GHz and 3.8GB RAM. The results show a fairly steady improvement of the objective function that lessens somewhat over time. When generating the second file, the initial objective score is better than when generating file 1. This can be explained by the fact that the initial fragment is based on 16 randomly generated measures. The second file is formed by 8 "optimized" measures followed by 8 randomly generated measures, which causes the starting score to be better. This is confirmed by Figures 4(a) and 4(b). When generating file 2 a better end score can be found than with file 1 despite the lower cutoff times. The maximum cutoff time of the algorithm is respectively 2 and 3 seconds for CF and 7 and 10 seconds for CP, as described in section 4.1. Figure 4(a) shows that the algorithm is able to find an optimal cantus firmus before the maximum time is reached. This allows the generation of the counterpoint to begin sooner (after 1.5 and 2 seconds respectively), thus expanding the time that the VNS can use to generate CP.



(a) Cantus firmus (b) Counterpoint

Figure 4: Evolution of the objective function over time

The quality of the generated music depends highly on the architecture of the mobile device on which it is installed. The results described in the previous paragraph confirm that the optimal objective score is reached for the cantus firmus. There is also a significant improvement of the objective score for counterpoint. While the objective score only measures how well the generated music fits into the counterpoint style, it is the subjective opinion of the authors that music sounds pleasant to the ear even on lower-end devices. The reader is invited to install the app and listen to the results of this research.

While the music generated by FuX can be considered to largely adhere to the counterpoint rules, it would be interesting to expand the objective function in future versions. Human baroque composers often base their work on the principles of counterpoint, but a finished composition has a encompassing theme and mixes the counterpoint rules with a composer's creative freedom. It could be argued that the fact that FuX does not find the optimal solution, can be interpreted as a random creative input. Still, an interesting future improvement could be to add more complex rules to the objective function, thus endorsing for instance a recurring theme and more structure, making the generated music sound more like a complete and coherent composition.

An official .apk application package has been generated called FuX. This package is freely available through Google Play at `http://play.google.com` and can be installed on any Android phone from version 2.1 and up. The user interface for FuX version 1.0 is simple but functional (see Figure 5). This user interface is currently being redesigned to allow a user to specify more options such as: instrument, key, etc.

5 Conclusion

A user-friendly Android application was implemented that can *continuously* play a stream of new counterpoint music. The implemented app, FuX, uses a variable neighborhood search algorithm to generate the music. The VNS is based on a similar algorithm that generates musical fragments of a pre-specified length on a pc. The original algorithm was adapted to allow the *continuous* generation of music. In order to evaluate the quality of a fragment, a quantification of the extensive rules of Fux were used. This resulted in an Android app with a user-friendly interface that can generate a continuous stream of music that sounds pleasing to the ear.

Future extensions include improvements to FuX's user interface, so that the user can change the

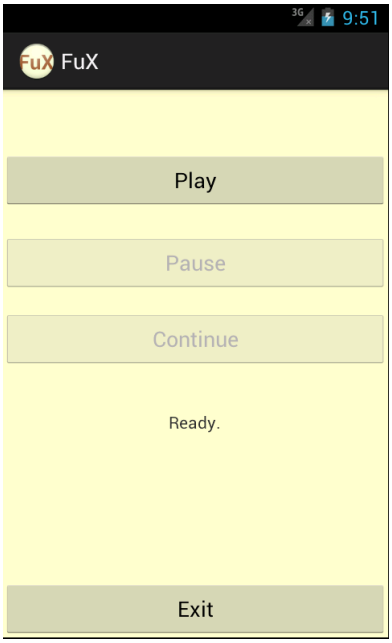


Figure 5: FuX 1.0 user interface

key, choose the instrument, etc. FuX might also be ported to the iOS platform. Other possible extensions of this research include allowing more voices at the same time, adding a recurring theme to the music or working with different musical style. The authors are currently working on composer-classification models. By including these models in the objective function FuX could be adapted to generate music with composer-specific characteristics.

References

[1] G. Aguilera, J. Luis Galán, R. Madrid, A.M. Martínez, Y. Padilla, and P. Rodríguez. Automated generation of contrapuntal musical compositions using probabilistic logic in derive. *Mathematics and Computers in Simulation*, 80(6):1200–1211, 2010.

[2] A. Alpern. Techniques for algorithmic composition of music. *On the web: http://hamp. hampshire. edu/˜ adaF92/algocomp/algocomp*, 95, 1995.

[3] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[4] Hugo Barra. 500 million. September 2012. URL `https://plus.google.com/u/0/110023707389740934545/posts/R5YdRRyeTHM`.

[5] J.A. Biles. Autonomous genjam: eliminating the fitness bottleneck by eliminating fitness. In *Proceedings of the GECCO-2001 Workshop on Non-routine Design with Evolutionary Systems*, San Francisco, California, USA, 7 2001. Morgan Kaufmann.

[6] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[7] D. Bornstein. Dalvik vm internals. In *Google I/O Developer Conference*, volume 23, pages 17–30, 2008.

[8] E.A. Bowles. Musicke's handmaiden: Of technology in the service of the arts. In *in H.B. Lincoln and Music*, page 4. Ithaca, NY: Cornelled., Ithaca, NY: Cornell ed., 1970.

[9] M. Braunhofer, M. Kaminskas, and F. Ricci. Recommending music for places of interest in a mobile travel guide. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 253–256. ACM, 2011.

[10] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[11] A.R. Burton and T. Vladimirova. Generation of musical sequences with genetic techniques. *Computer Music Journal*, 23(4):59–73, 1999.

[12] G. Caporossi and P. Hansen. Variable neighborhood search for extremal graphs: 1 the autographix system. *Discrete Mathematics*, 212(1):29–44, 2000.

[13] G. Carpentier, G. Assayag, and E. Saint-James. Solving the musical orchestration problem using multi-objective constrained optimization with a genetic local search approach. *Journal of Heuristics*, 16(5):1–34, 2010.

[14] Patrick Donnelly and John Sheppard. Evolving four-part harmony using genetic algorithms. In *EvoApplications (2)*, volume 6625 of *Lecture Notes in Computer Science*, pages 273–282. Springer, 2011.

[15] J.T.B. Fajardo and C.M. Oppus. A mobile disaster management system using the android technology. *WSEAS Transactions on Communications*, 9(6):343–353, 2010.

[16] K. Fleszar and K.S. Hindi. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research*, 155(2):402–413, 2004.

[17] J.J. Fux and A. Mann. *The study of counterpoint from Johann Joseph Fux's Gradus Ad Parnassum - 1725*. Norton, New York, 1971.

[18] M. Geis and M. Middendorf. An ant colony optimizer for melody creation with baroque harmony. In *IEEE Congress on Evolutionary Computation*, pages 461–468, 2007.

[19] F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, 1993.

[20] M.H. Goadrich and M.P. Rogers. Smart smartphone development: ios versus android. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 607–612. ACM, 2011.

[21] *Google Android SDK*. Google, November 2012. URL `http://developer.android.com/sdk/index.html`.

[22] Google. Jelly bean. Technical report, 2012. URL `http://developer.android.com/about/versions/jelly-bean.html`.

[23] D. Gürer. Pioneering women in computer science. *ACM SIGCSE Bulletin*, 34(2):175–180, 2002.

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

[25] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449–467, 2001.

[26] P. Hansen and N. Mladenović. Variable neighborhood search. *Handbook of metaheuristics*, pages 145–184, 2003.

[27] P. Hansen, N. Mladenović, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.

[28] D. Herremans and K. Sörensen. Composing fifth species counterpoint music with variable neighborhood search. *University of Antwerp, Faculty of Applied Economics, Working Papers*, (2012020), 2012.

[29] Dorien Herremans and Kenneth Sörensen. Composing first species counterpoint with a variable neighbourhood search algorithm. *Journal of Mathematics and the Arts*, 6(4):169–189, 2012.

[30] A. Horner and D.E. Goldberg. Genetic algorithms and computer-assisted music composition. *Urbana*, 51 (61801):437–441, 1991.

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

[32] M. Kaminskas and F. Ricci. Contextual music information retrieval and recommendation: State of the art and challenges. *Computer Science Review*, 2012.

[33] Alex Leffelman. *Android-Midi-Lib*, November 2012. URL `http://code.google.com/p/android-midi-lib/`.

[34] C.M. Lin, J.H. Lin, C.R. Dow, and C.M. Wen. Benchmark dalvik and native code for android system. In *Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference on*, pages 320–323. IEEE, 2011.

[35] R.A. McIntyre. Bach in a box: The evolution of four part baroque harmony using the genetic algorithm. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence*, pages 852–857. IEEE, 1994.

[36] R. Meier. *Professional Android 4 application development.* Wrox, 2012.

[37] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11): 1097–1100, 1997.

[38] B.S. Mongia and V.K. Madisetti. Reliable real-time applications on android os. *IEEE Electrical and Computer Engineering Electrical and Computer Engineering.*

[39] A.F. Moore. Categorical conventions in music discourse: Style and genre. *Music & Letters*, 82(3):432–442, 2001.

[40] A. Moroni, J. Manzolli, F.V. Zuben, and R. Gudwin. Vox populi: An interactive evolutionary system for algorithmic music composition. *Leonardo Music Journal*, 10(2000):49–54, 2000.

[41] G. Nierhaus. *Algorithmic composition: paradigms of automated music generation.* Springer, 2009.

[42] E. Oliver. A survey of platforms for mobile networks research. *ACM SIGMOBILE Mobile Computing and Communications Review*, 12(4):56–63, 2009.

[43] S. Park and K. Chung. Query by singing/hum ming (qbsh) system for polyphonic music retrieval. In *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, pages 245–246. IEEE, 2012.

[44] S. Phon-Amnuaisuk and G. Wiggins. The four-part harmonisation problem: a comparison between genetic algorithms and a rule-based system. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, pages 28–34, 1999.

[45] S. Phon-Amnuaisuk, A. Tuson, and G. Wiggins. Evolving musical harmonisation. In *Artificial neural nets and genetic algorithms: proceedings of the international conference in Portorož, Slovenia, 1999*, page 229. Springer Verlag Wien, 1999.

[46] John Polito, Jason M. Daida, and Tommaso F. Bersano-Begey. Musica ex machina: Composing 16th-century counterpoint with genetic programming and symbiosis. In *Evolutionary Programming*, volume 1213 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 1997.

[47] S. Ratabouil. *Android Ndk Beginner's Guide.* Packt Publishing Ltd, 2011.

[48] J. Rothgeb. Strict counterpoint and tonal theory. *Journal of Music Theory*, 19(2):260–284, 1975.

[49] F. Salzer and C. Schachter. *Counterpoint in composition: the study of voice leading.* Columbia University Press, 1969.

[50] O. Sandred, M. Laurson, and M. Kuuskankare. Revisiting the illiac suite—a rule-based approach to stochastic processes. *Sonic Ideas/Ideas Sonicas*, 2:42–46, 2009.

[51] F.J. Schwartz and R. Ritchie. Music listening in neonatal intensive care units. *Music Therapy & Medicine*, 2004.

[52] K.C. Son and J.Y. Lee. The method of android application speed up by using ndk. In *Awareness Science and Technology (iCAST), 2011 3rd International Conference on*, pages 382–385. IEEE, 2011.

[53] K. Sörensen. Metaheuristics – the metaphor exposed. *International Transactions on Operations Research*, Accepted for publication.

[54] K. Sörensen and F. Glover. Metaheuristics. In *Encyclopedia of Operations Research and Management Science*. S. I. Gass and M. C. Fu, eds., Springer, New York, 3rd edition, (in press).

[55] N. Tokui and H. Iba. Music composition with interactive evolutionary computation. In *Proceedings of the Third International Conference on Generative Art*, volume 17:2, pages 215–226, 2000.

[56] C. Truchet and P. Codognet. Musical constraint satisfaction problems solved with adaptive search. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 8(9):633–640, 2004.

[57] George Tzanetakis, Manjinder Singh Benning, Steven R. Ness, Darren Minifie, and Nigel Livingston. Assistive music browsing using self-organizing maps. In *Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '09, pages 3:1–3:7, New York, NY, USA, 2009. ACM.

[58] P. Yong-Cai, L. Wen-chao, and L. Xiao. Development and research of music player application based on android. In *Communications and Intelligence Information Security (ICCIIS), 2010 International Conference on*, pages 23–25. IEEE, 2010.

[59] X. Zheng, G. Bao, R. Fu, and K. Pahlavan. The performance of simulated annealing algorithms for wi-fi localization using google indoor map. 2012.