BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

# Evolutionary Music Composition

**PhD thesis summary**

PhD student: Sulyok Csaba
Scientific supervisor: Prof. Dr. Czibula Gabriela

2019

BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

# Evolutionary Music Composition

**PhD thesis summary**

PhD student: Sulyok Csaba
Scientific supervisor: Prof. Dr. Czibula Gabriela

2019

**Keywords:** algorithmic music composition; genetic programming; linear genetic programming; domain-specific knowledge; automated fitness raters

# Contents

# Thesis table of contents

# List of publications

All rankings are listed according to the 2014 classification of journals[1] and conferences[2] in Computer Science.

1. **Sulyok, C.**, McPherson, A., and Harte, C. (2015). *Corpus-taught Evolutionary Music Composition*. In Proceedings of the 13th European Conference on Artificial Life (ECAL), pages 587–594, York, UK. MIT Press. (**ISI Proceedings**)

   **Rank B, 4 points.**

2. **Sulyok, C.** and Harte, C. (2017). *On Virtual Machine Architectures for Evolutionary Music Composition*. In Proceedings of the 14th European Conference on Artificial Life (ECAL), pages 577–584 Lyon, FR. MIT Press. (**ISI Proceedings**)

   **Rank B, 4 points**

3. **Sulyok, C.** (2018). *Genetic Operators for Evolutionary Music Composition*. In Proceedings of the 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pages 253–259, Timişoara, RO. (**ISI Proceedings**)

   **Rank C, 2 points**

4. **Sulyok, C.**, McPherson, A., and Harte, C. (2019b). *Evolving the Process of a Virtual Composer*. Natural Computing, volume 18, issue 1, pages 47–60. Springer Netherlands. (**indexed WoS**)

   **Rank B, IF=0.778, 4 points.**

5. **Sulyok, C.**, Harte, C., and Bodó, Z. (2019a). *On the Impact of Domain-specific Knowledge in Evolutionary Music Composition*. In Proceedings of the 2019 Genetic and Evolutionary Computation Conference (GECCO), pages 188–197, Prague, CZ. (**ISI Proceedings**)

   **Rank A, 8 points**

[1]http://informatica-universitaria.ro/getpfile/16/CSafisat2.pdf;http://hfpop.ro/standarde/doctorat/2014-jurnale.pdf

[2]http://informatica-universitaria.ro/getpfile/16/CORE2013_Exported.xlsx;\http://hfpop.ro/standarde/doctorat/2014-conferinte.xlsx

6. **Sulyok, C.** (2019). *Towards Automated Quality Assessment Methods in Algorithmic Music Composition*. In Proceedings of the 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) *(accepted for publication)*, Timişoara, RO. (**ISI Proceedings**)

   **Rank C, 2 points**

Publications score: **24 points**.

Chapter 1

# Introduction

The process of creating and assessing music is fundamentally subjective and hard to define. The judgment of the composer as to whether a new musical idea is good or bad will be a subjective decision based on their knowledge and memory of previous pieces (see Figure 1.1). This feedback of the creative self suggests that evolutionary computation techniques may find utility in algorithmic music composition.

When designing an evolutionary system, we must first answer the question: "What is this system supposed to evolve?". Previous composition systems have generally attempted to evolve musical pieces directly, but we propose evolving the composition *process* instead. We cast the action of composing a piece of music as a process running on a Turing-complete virtual computing machine. The virtual machine (VM) has a set of instructions that will be executed in a given order depending on the initial state of its memory (i.e. its program) and a way of writing notes onto a musical "score" whenever an output instruction is encountered. The resulting system differs from previous work in the literature by incorporating these linear genetic programming (LGP) (Brameier and Banzhaf, 2007) elements.

The development of skill then becomes a genetic programming (Koza, 1992) problem; the genotype is the program string presented to the virtual machine and the phenotype its musical output. In such a system, the executing process on the virtual machine can hold internal structuring rules and information that are not visible in the final musical phenotype.

Given the personal perception of music, or any art form in general, an objective definition of a "good" output is difficult to obtain (Waschka II, 2007). Our evaluation process measures output quality as distributional similarity to existing works in a corpus of real music. As automated fitness raters are seldom deployed in the literature, we propose these two new feature extraction and assessment methods as a novelty, proposing a method for comparing them to other automated metrics as well. As a further addition to the field of evaluation, corpus pieces are not used to seed the initial population; instead they only inform the fitness tests, allowing a broader search space.

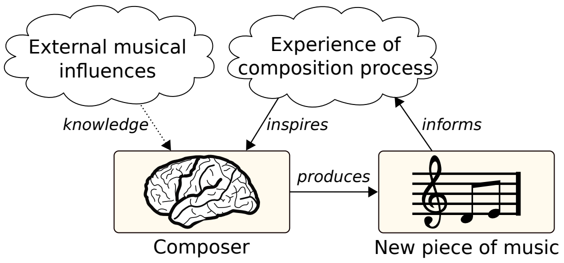
External musical influences
Experience of composition process
knowledge
inspires
informs
produces
Composer
New piece of music

Figure 1.1: The creative process of a composer is informed by both experience of music in general (external influence of others' music) and experience gained through practice of the composition process itself.

The current work also focuses on the importance of *genetic operators* (Cavicchio Jr., 1972)—the set of procedures creating subsequent generations from existing ones. We present a comparative study on dynamic adaptation of these operators. Although online hyperparameter optimization has been researched extensively for genetic algorithms and programs, it has seldom been attempted in an LGP setting.

The system described is suited for the creation of any output similar to a set of inputs, so long as the choice of extracted features is not influenced by the nature of its model. However, the question arises: "Is it helpful for the system to be informed to some extent about the nature of its own output?" We propose a comparison between setups that differ in their awareness of the end goal of creating music. A *domain-specific language* (DSL) (Fowler, 2010) with music-related commands is presented, which reduces the otherwise vast array of possible movements to ones typically found in composing. We also propose a diatonic pitch representation, which allows musical notions to be clear to the system (Selfridge-Field, 2004).

The thesis aims to present the following original contributions to the field:

- An overview of the general linear genetic programming approach to algorithmic music composition, as first proposed in Sulyok et al. (2015), as well as exploring the parameter space of such a system (see Section 3.2 and Sulyok et al. (2019b));
- A comparison of different virtual machine architectures and instruction sets in the above setting (see Sections 2.1, 3.3 and Sulyok and Harte (2017));
- An objective and fully automated quality assessment method of artificial data relying on statistical similarity to a corpus of real-world data, together with multiple proposed corpora, two feature extraction methods (see Section 2.3 and Sulyok

et al. (2019a)) and a proposal for a comparison study with other approaches in the literature (see Section 3.6 and Sulyok (2019));

- A comparison of genetic operators and their adaptive capabilities in the above setting (see Sections 1.1.3, 3.4 and Sulyok (2018));
- An exploration of the impact of domain-specific knowledge on the system in the form of a musically inspired DSL and diatonic pitch representation (see Sections 2.1, 3.5 and Sulyok et al. (2019a)).

The thesis is structured as follows.

Chapter 1 presents the foundation of our research, including a theoretical overview of genetic programming, its subcategories and associated practices. Section 1.2 gives an overview of past literature in the field of evolutionary music and fitness assessment methods. Non-evolutionary concepts deployed in the current research, such as domain-specific languages and n-grams, are also detailed here.

Chapter 2 provides a methodological overview of the building blocks of our proposed approach: we detail the machinations of the virtual machine together with its instruction sets (Section 2.1), our representations of music (Section 2.2) and the proprietary assessment methods of musical similarity (Section 2.3).

Chapter 3 presents multiple sets of experiments and results for the presented concepts: Section 3.1 presents a proof of concept for the framework as defined thus far; it details initial experiments with partially empirical parameters, to gain a starting point. Afterwards, Section 3.2 delves into hyperparameter space exploration to compare and contrast different settings for population size, number of voices (tracks) in the corpus and output pieces, as well as the deployed survival/reproduction mechanism. An in-depth comparison of different general-purpose virtual machine architectures and memory sizes is presented in Section 3.3. An exploration of adaptive genetic operator settings is shown in Section 3.4. Section 3.5 delves into the effects of embedding musical knowledge into our VM, and presents a comparison to regular representations. Finally, Section 3.6 proposes an ongoing fitness measurement metric comparison study with other automated rating approaches in the literature.

Appendix A details the supporting material attached to the current report, including the open-source repository location, its structure and how to reproduce the presented experiments. Appendix B shows a number of randomly selected musical pieces generated during the most recent experiment run.

# Chapter 2

# The proposed approach

Our system follows a conventional genetic programming structure (see Figure 2.1). A fixed size population is maintained throughout a run, represented by genotypes, phenotypes and fitness test scores of individuals.

The genotype is a fixed-size byte array fully representing a state of a virtual machine, including all memory segments, registers and flags—we name such an array a *genetic string*. Any values constitute a valid input for the VM, therefore the creation of a zeroth generation is equivalent to generating random arrays of the given size. The genetic strings provide the initial state of the VM, after which the programs are executed, then the output bytes are collected and parsed into phenotype musical models. The structure of this model is completely independent of the structure and mechanism of the VM. This two-stage approach to rendering deviates from previous musical evolutionary systems in that the genetic string is not directly used to build the phenotype, instead being interpreted by the virtual machine.

## 2.1 Virtual machine

As previously mentioned, we interpret genetic strings using a VM— the output is used to build the musical model. The byte values encountered in the memory are mapped to instructions in a predefined set. The initial position of the instruction pointer is part of the genetic string, just as the value of any other register or memory segment. The VM reads bytes one by one from the RAM and executes the instruction mapped to the encountered value. An instruction set may contain many kinds of commands that manipulate data within the VM, such as data transfer, arithmetic, conditionals, etc. To produce data for phenotype building, we define a special output instruction that outputs one or more bytes from the memory or one of the registers.

Interpretation of a genetic string continues until one of two halting conditions is met: either an expected number of output bytes is produced or a maximum number of instruction cycles is reached. The latter is present as a fail-safe mechanism to exit the program when an infinite loop containing no output instructions is encountered.

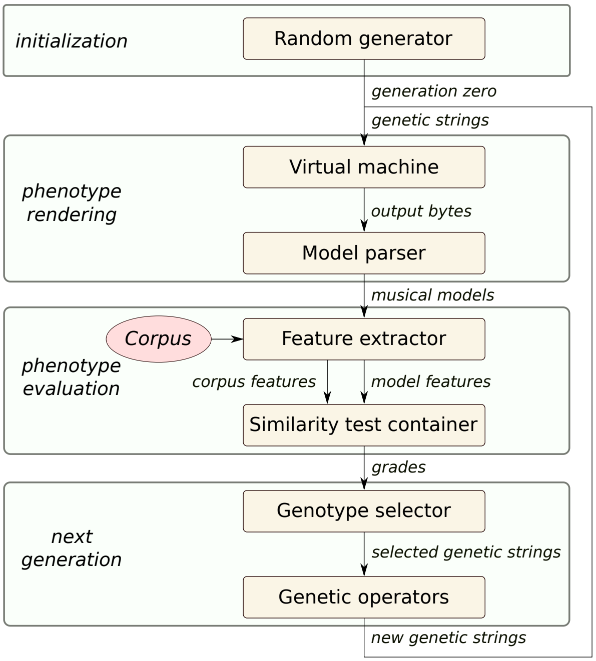

Figure 2.1: Workflow of the proposed algorithm: A population of genetic strings is interpreted by the virtual machine and the resulting bytes are parsed to build musical models. Relevant features are extracted and compared to those of the corpus, yielding a fitness. Based on these scores, genetic strings are bred and mutated to produce the next generation.

The initial experiments Sulyok et al. (2015) propose a virtual machine architecture loosely based on the Intel 8080 microprocessor, with the addition of outputs. It serves as a starting out point for further exploration and comparison in later parts of the research. The machine includes a 64kB random access memory (RAM) space which stores both instructions and data, allowing self-rewriting during execution. Being an 8-bit machine, 256 different instructions are provided. The instruction pointer and an auxiliary data pointer may address any byte in memory; moving the former represents a jump. Besides the RAM, it contains a 256 byte circular stack addressable by an 8-bit stack pointer. Other available registers include 8 general-purpose registers, one accumulator and a carry flag. It includes typical data transfer, arithmetic, logic, branching, conditional and machine control commands alongside the dedicated output commands.

The experiments presented in Sulyok et al. (2019a) compare virtual machine architectures differing in their embedded knowledge of the nature of their output. It introduces a domain-specific language designed around knowledge of its output, therefore notes in the memory of the virtual composer are cast as 8-bit registers. Data manipulation instructions for the note registers reflect domain-specific impact: the duration of any note may be doubled, halved or dotted, and pitches may be incremented/decremented within the diatonic scale.

## 2.2 Representing music

Our phenotype is a model that represents a musical composition; the model parser interprets the output byte array from the VM to produce such a model. For our experiments, we define two music representations–one is a general and permissive model based on MIDI (we refer to this as the *complex model*), while the other is a simplified view informed by the statistics of a corpus (the *reduced model*). The former is represented as a set of tracks, each consisting of a set of notes. Each note has the following properties:

1. *Inter-onset interval (IOI)* - The time period between the onset of the previous note and the current note in a particular track. For the first note in a track, it is the time interval between the beginning of the piece and the onset.
2. *Duration* - Time period between the onset and offset of the note.
3. *Pitch* - A 7-bit numeric value (between 0 and 127) representing pitch as defined in the MIDI protocol. The value 69 is associated with the 440Hz concert *A*, with an increase or decrease of one unit representing a one semitone rise or fall in pitch respectively.

The reduced model is a simplified version of the complex representation, informed by the statistics of the folk song corpus used in our experiments. The following observations are made to the corpus at hand:

- The pieces are exclusively monophonic, removing the necessity of multiple tracks.
- The pieces contain vocal melodies with no rests or overlapping notes, therefore note onset and offset times may be fully embedded in the duration, i.e. each note begins when the previous ends.
- 99.5% of note durations fall into one of 8 common duration values, therefore 3 bits are sufficient for its representation.
- Similarly, 99.7% of pitches may be mapped to 32 contiguous chromatic values, representable on 5 bits.

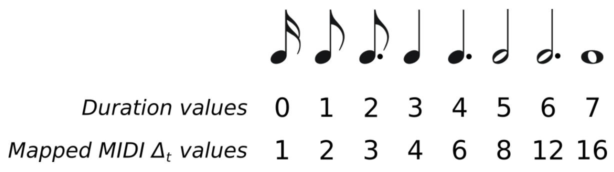

Figure 2.2: Discrete possible durations of the notes in a reduced model, together with the corresponding MIDI values, assuming 4 ticks per quarter note and a tempo of 120BPM

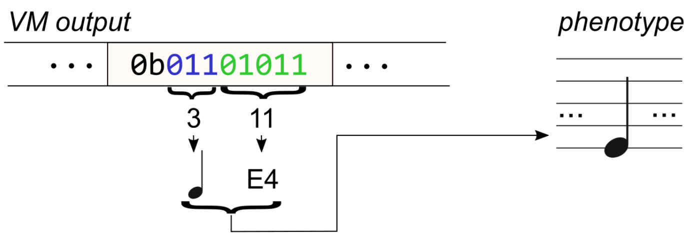

Figure 2.3: Building reduced models from virtual machine output bytes. Each byte becomes a note whose 3 most significant bits are parsed as the duration, while the other 5 become the pitch.

Therefore the reduced model becomes a piece of single-track music containing a series of notes, each represented fully by duration and pitch.

The 8 common duration values are mapped to MIDI temporal units as seen in Figure 2.2. The pitch value is represented by an integer in the range from 0 to 31 with a flexible mapping to MIDI pitches. Given the relatively small number of different values we allow for duration and pitch, we may represent a note uniquely with a single byte. To make full use of the information produced by the VM, the phenotype renderer creates 1 note per output byte, using the 3 most significant bits as duration, and the remaining 5 as the pitch (see Figure 2.3).

The role of any pitch in a piece of music is determined by its interval with respect to the tonic of the given harmonic key. To incorporate the contextual impact of pitch, we define different representations in the reduced phenotype, with the possibility of conversion between each. They are as follows:

- *standard chromatic* – On the equivalent scale as MIDI but shifted down by 53 to fit within the above mentioned range; outliers are shifted by octaves to fit within the interval and retain consonance, and key is ignored.
- *shifted chromatic* – The mean key in the corpus members is found to be 64 (E4). The shifted version of this center pitch (11) is designated as the tonic and all

notes are transposed to center around this value. Transposition does not impact the consonance or overall feel of the musical piece (Plomp and Levelt, 1965) and this representation allows pitch values to always take on the same function, e.g. the value 18 is always a perfect fifth.

- *diatonic* – Pitches are once again centered around 11, however they are diatonically quantized so that an increase represents a step within the diatonic 7-note scale. This representation prohibits the use of accidentals, i.e. chromatic "out-of-place" notes, and therefore may help the system evolve harmonically correct pieces more easily.

## 2.3 Assessing musical quality through similarity

The fitness of any rendered phenotype is determined by a series of similarity tests. All tests aim to evaluate how statistically analogous a model is to those in the corpus. To this end, certain features are extracted from the corpus; the same features of incoming models are compared to provide the fitness values. Corpus features are clustered using the k-means++ algorithm (Arthur and Vassilvitskii, 2007) to control the broadening of the search space.

Our experiments deploy two different corpora. Earlier iterations use Bach's *Inventions and Sinfonias*[1], comprising 30 keyboard exercises. This catalogue of musical pieces was chosen for their brevity, constant tempo, stylistic homogeneity, and also their usage in other papers in the literature. Two versions of this corpus is deployed in our experiments: single- and dual-track. Both contain the same pieces, each comprising the same set of notes, but the dual-track versions are separated into two voices divided by pitch range (effectively splitting the left and right hand keyboard parts).

Initial experiments presented in Sulyok et al. (2019b) reveal that the Bach keyboard exercises may be an overly complex choice of corpus. The inherent pieces deploy many key changes and complex rhythmic patterns, possibly making general rules hard to deduce for a system starting out from complete randomness. This suggests that using a simpler corpus may provide benefits. Therefore the experiments on the impact of domain-specific knowledge (Sulyok et al., 2019a) deploy the "Cimbalom" Hungarian folk song collection[2] as corpus. The files are all monophonic, comprising only vocal arrangements with no accompanying instruments.

Analyzing the entropy of both corpora in terms of duration, pitch and the combination of both reveal that the values follow normal distributions ( 45%). Entropy determines the information content of the data: random notes would yield high values, while repeatedly playing the same note results in minimal entropy; neither of these extremes sound pleasing

[1]Works BWV 772-801 downloaded from `http://www.midiworld.com/bach.htm`

[2]Népdalok collection downloaded from `http://www.cimbalom.nl/nepdalok.html`

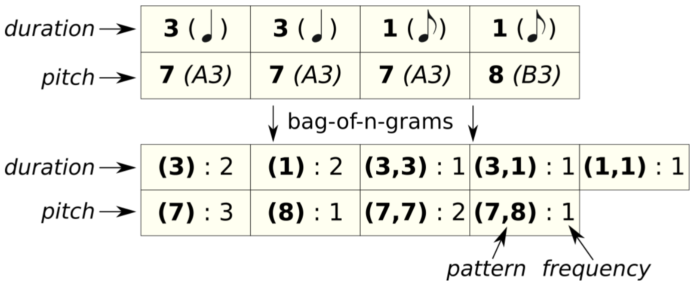

Figure 2.4: Demonstrating the construction of phenotype n-grams (uni- and bigrams) for comparison with corpus members.

in a piece of music. Both feature extraction methods presented below are paired with entropy testing.

We define a descriptor as the output of a series of transform methods applied to an input model. The descriptor-based similarity tests look at sameness between the descriptors of the input and that of the corpus. Four transforms (histograms, differential of histograms, Fourier transform and differential of Fourier transform) are applied to each of the three properties of a model, resulting in a total of twelve correlation tests. We calculate the correlation between two descriptors line-by-line using the *Pearson correlation coefficient*.

For the second analyzed similarity assessment metric, we borrow the cosine similarity measure from information retrieval: two melodies will have a maximum similarity if the angle enclosed by their bag-of-n-grams vectors is zero, i.e. their n-gram occurrences follow the same distribution. The fitness function incorporates the two base properties of the reduced models: duration and pitch. However, the n-gram-based fitness value is derived from six components. These are n-grams of: (a) durations, (b) pitches, (c) absolute pitch–duration pairs, (d) differences between consecutive durations, (e) differences between consecutive pitches, and (f) duration difference–pitch difference pairs. Figure 2.4 shows the construction of the uni- and bigram features for (a) and (b) of an example piece.

# Chapter 3

# Experiments and results

The current chapter presents all sets of experiments performed during the research, along with their results, also proposing an automated fitness rater comparison.

## 3.1 Baseline evaluation

This section outlines the initial tests employing the evolutionary music composition tool. The configurations and results set forth here are relayed as presented in Sulyok et al. (2015). We run a total of 40 experiments, each time allowing the algorithm to reach 20,000 generations; the parameters used here have been derived empirically from earlier test runs. We use a population size of 256 with a survival rate of 3% and number of clusters $k = 5$.

Figure 3.1 shows the mean and maximum grades per generation, averaged over the 40 runs. We can observe a steady rise in the maximum score, but stagnation in the mean values. This can be explained by the fragility of a genetic string when faced with crossover and mutation; even a small change can produce a completely different musical model. Figure 3.2 shows the grade distribution of the highest-scoring individual in each run. It demonstrates that the algorithm gives consistent results on different iterations. Although repetition and variation appear in the generated pieces, other musical properties such as *harmony*, *melody* or *entropy*, are somewhat lacking. This suggests the need for further fitness tests inspired by music-theory.

## 3.2 Comparing system parameters

The follow-up experiments explore the space of possible system parameters to measure their impact on the progress of the evolutionary process. The experiments relayed here are in line with Sulyok et al. (2019b). For each set of parameters, we execute 20 separate test runs, once again allowing the system to complete 20,000 generations each time. Parameters kept constant for all runs include a survival probability of 15%, maximum survival age of 3, maximum cut point ratio of 0.1%, and maximum mutation ratio of 2%.

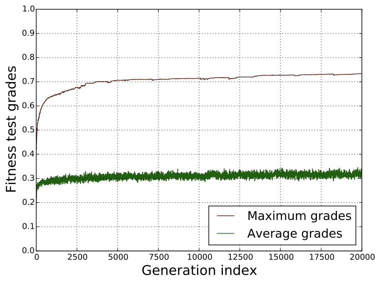

Figure 3.1: Progression of grades over 20,000 generations: the mean and maximum grades for each generation.

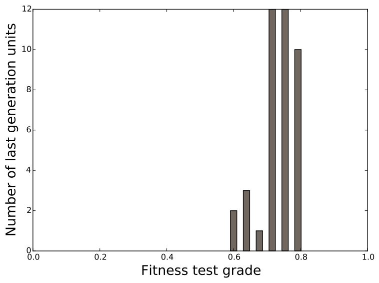

Figure 3.2: Distribution of highest scoring individual grades. Every run converges to at least 60%.

We use 5 corpus clusters and VM halting conditions of completing 60,000 fetch cycles or producing 2,600 output bytes.

Varying parameters between trials include the population size ($N \in \{2^x : 4 \leq x \leq 10\}$), the number of tracks in the corpus (either 1 or 2) as discussed in Section 2.3, and the survival mechanism: either probabilistic or deterministic. The following conclusions may be drawn:

- Larger population sizes result in both better mean and maximum grades, and fitting a curve to the available grades (see Figure 3.3) suggests that only marginal gains would be had by further increasing it.

- Using the dual-track corpus results in slightly smaller mean values and larger maxima, probably due to the inherent dimensionality rise.

- A probabilistic survival strategy prevents elitism and ensured better population diversity: although smaller maximum grades are produced, significantly better mean grades are achieved.

- Observing the changes in the rate of occurrence of different instruction types (see Figure 3.4) shows that the algorithm favors more output instructions than occur in random data. We can also observe a decline in the number of branching instructions which may be due to the detrimental effect of infinite loops.

- Subjective music evaluation gives favor to dual-track pieces for their inherent polyrhythmic nature, however harmony and overall musicality are still lacking.

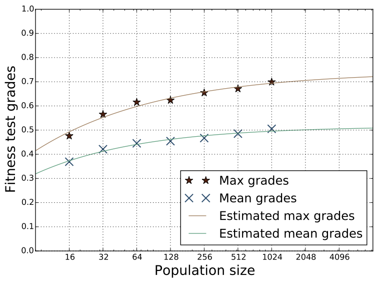

Figure 3.3: Estimated curve fit over different population sizes, suggesting that further increasing it would not give significantly better results.

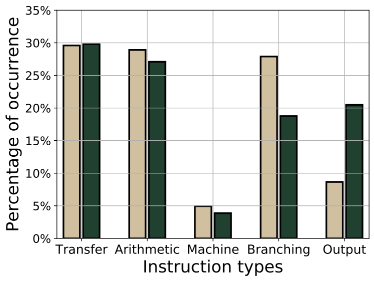

Figure 3.4: Occurrence rate of encountered instruction types: left bars show generation zero (random VM content), right are the best individuals.

## 3.3 Virtual machine architectures

The current chapter compares different settings of the virtual machine, whose only requirement is that it be able to output bytes based on a genetic string: a byte array fully representing its state. The experiments and results presented here are relayed as discussed in Sulyok and Harte (2017). Tested changing parameters include:

- *virtual machine architecture* - either von Neumann or Harvard;
- *instruction set design* - three different instruction sets are tested: the complex set used in the previous experiments, a single-instruction OISC set using `SBNZ` and a stack-based instruction set;
- *memory size* - either 256, 4096 or 65536 (addressable using 8, 12 and 16 bits, respectively).

This results in a total of 18 different configurations. 20 experiments are run for each, requesting 30 second long pieces and allowing the algorithm to reach 10,000 generations. Other parameters are chosen as the most optimal based on previous configurations. The following conclusions may be drawn:

- The results by VM architecture (see Figure 3.5) suggest the complex instruction as the best, but also the most vulnerable to change.
- The single-instruction machine, although reaching the smallest grades, performed better than expected given its complexity.
- The von Neumann architectures score slightly higher, proving the explorative usefulness of the self-rewriting capability. Only the single-instruction set achieved

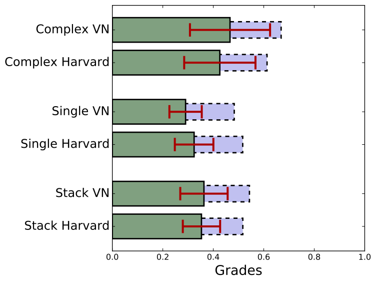

Figure 3.5: The final generation grouped by instruction set and VM architecture. The wide bars show mean and standard deviation of the entire population, while the narrow bars show the average highest-scoring entity.

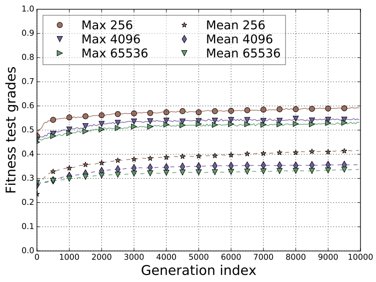

Figure 3.6: Mean and maximum grade progression over the generations, grouped by overall memory size. Virtual machines using smaller memory sizes achieved higher grades, their mean increasing even in later generations.

better results using the Harvard architecture; this may be caused by its long instructions being more fragile to self-rewriting in the von Neumann case.

- Using smaller memory sizes consistently produce higher grades (see Figure 3.6), however their resulting music files are subjectively less appealing and overly repetitive. We may conclude a deficiency in the fitness test design: they do not properly reward complexity inherent in using more memory.

## 3.4 Comparing genetic operators

In this section we investigate different approaches to hyperparameter configuration of genetic operators within our approach. We analyze the benefits of adaptively setting operator distributions and rates using heuristic hill climbing (Russell and Norvig, 2016). The experiments in this section are presented in Sulyok (2018)–they use the descriptor-based correlation tests, the corpus of Bach keyboard exercises and the complex model representation.

The operator distributions take on standard values (8% reproduction, 90% crossover and 2% mutation, as proposed by Koza (1992)) as well as adaptive values using the standard one as a starting point. The tested rates include the number of cut points $n_c$ and the number of mutated bytes $n_m$, both represented proportionally to the size of the genetic string. Changing values include:

1. constant values used in Sulyok et al. (2019b): $n_c = 0.1\%$, $n_m = 2\%$;

2. global adaptive values starting with the same values used in the first case, adapted once globally every generation;
3. individualized adaptive values starting with the same values used in the first case, inherited and adapted per individual every generation.

For each of the resulting 6 configurations we run a total of 20 iterations, allowing the algorithm to reach 10,000 generations. The results suggest that adaptive operator settings provide only marginal benefits, and always tend to increase the number of mutated entities at the cost of fewer units bred through crossover. A usual result shows large maximum grades at the cost of lower means–a surprising result since the means inform the hill climbing process. Individual rate adaptations provide no significant improvement over the baseline; this could be due to the overwhelming number of dimensions hill climbing is trying to explore.

## 3.5 Impact of domain-specific knowledge

The current section investigates the effect of embedding different levels of musical knowledge into the virtual machine (VM) architectures and phenotype representations of the system. The experiments are presented in Sulyok et al. (2019a)–they use the n-gram-based similarity tests, the corpus of Hungarian folk songs and the reduced model representation.

We examine two separate instruction sets that differ in their knowledge of musical structure: one the Turing-complete register machine used in previous research, unaware of the nature of its output; the other a domain-specific language tailored to operations typically employed in the music composition process. The phenotype is rendered as a the reduced musical model comprising a sequence of notes represented by duration and pitch. We compare three different pitch schemes with differing embedded knowledge of tonal concepts, such as key and mode.

With two different VM architectures and three pitch schemes, we present and compare results from a total of six configurations. The following conclusions may be drawn (see Figure 3.7):

- In all pitch representations, the DSL machine achieves better results than the general-purpose machine.
- The diatonic pitch representation generally performs more poorly than the other two, possibly due to its wider spectral range.
- The shifted chromatic representation outperforms the chromatic one only when using the DSL. This is understandable, since the DSL contains pitch manipulation

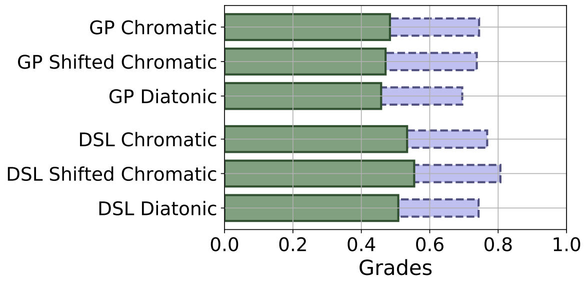

Figure 3.7: Last generation mean and maximum fitness values per configuration averaged over the 20 test runs

instructions which retain correctness in the output. In other words, it helps to avoid stray chromatic notes even if the representation would otherwise allow for them. When using chromatic pitch, the corpus members are spread across different keys, therefore consonant notes in one model may be dissonant in another, and the notions of key and harmony would need to emerge themselves rather than being inherent in the search space.

- The DSL machine outperforms the GP machine already in earlier generations, proving the effectiveness of the DSL even for random VM contents.

- A subjective evaluation shows many harmonically pleasing results (see example in Figure 3.8); however the entropy test is tricked in many cases by combining extreme

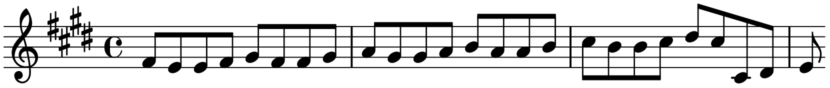

Figure 3.8: Portion of an example output showing variation on a small pattern. This model achieved 73% fitness.

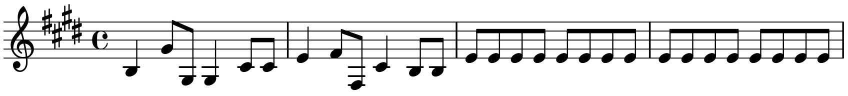

Figure 3.9: Portion of an example output showing a varied first part changing into a one-note loop. This model achieved 79% fitness.

value segments in the same piece of music (see example in Figure 3.9).

## 3.6 Fitness rater comparison proposal

The current section outlines a proposed set of experiments attempting to demonstrate the viability of the 2 proprietary fitness raters. To this end we compare the mechanisms to each other, as well as to other metrics proposed in the literature. The ongoing experiments are relayed here as submitted in Sulyok (2019).

As a first step, we plan to reproduce some of the more accepted automated raters present in the literature, and incorporate them into the current LGP system. The modular design (as seen in Figure 2.1) allows a black box-style swap of the quality assessment block for any other compatible one.

While any MIDI file may be assessed by any of the proposed metrics, a possible proof of quality would be correlation with real-world feedback from human listeners. To gather such numerical data, we propose using Amazon's crowdsourcing marketplace *Mechanical Turk* (MTurk)[1]. Given a set of chosen MIDI files, MTurk users would rate the pleasantness of each, becoming the reference quality assessment pseudo-function.

We propose selecting the musical models presented to users based on the following considerations: only generated music should be shown, evolved using all the different compared fitness functions, with a high variance in grades. The collected models are presented to MTurk users in a way that any single user would listen to all pieces and assign a numerical grade to it. The mean of the feedback values would constitute the reference grade for each model in the selection. Finally, correlation may be measured between the human-assigned and automated grades, providing numerical proof of their efficiency.

We propose the above experiment with the following values: 5 fitness functions (of which 2 are our previously presented ones) used to run a simulation with 500 individuals, reaching 1,000 generations. Of the 5 different last generations, 20 models would be chosen as presented above, resulting in 100 short pieces to present to MTurk users.

[1] Accessible at `https://www.mturk.com/`

# Chapter 4

# Conclusion and future work

Results from the literature show promise for algorithmic music composition, and many real-world musical pieces have been composed with the *assistance* of evolutionary algorithms. Indeed, Waschka II (2007) has always viewed these algorithms as auxiliary tools for composer inspiration instead of standalone virtual composers.

In the proposed research, we have successfully modeled the thought process of a virtual composer separately from the output of their work via virtual machines that produce output bytes to be parsed into musical models. These pieces have been compared to corpus members through a series of similarity tests involving statistical transforms, n-grams and Shannon entropy. We have refrained from setting favourable initial conditions to our system, such as using the corpus as the starting population.

The set of experiments show promise, demonstrating that the methodology succeeds in creating pieces of music that converge towards the properties of the chosen corpus. The output pieces exhibit certain musical qualities (repetition and variation) not specifically targeted by our fitness tests, emerging solely based on the statistical similarities. Although more complex musical properties such as harmony are lacking when viewing the system as general, these properties are also aided by injecting domain-specific knowledge into the system.

Finally, as the system has largely been tested on its own, with no in-depth comparison to other methods in the literature, we have proposed a set of comparative experiments around one of the most important aspects of the research: the automated quality assessment metric. We therefore set as our most immediate goal the execution and result analysis of these experiments.

The time representation may also be improved, since currently the data given to the tests is a function of note index. This allows loops to emerge in tracks with identical number of notes, but with a different duration, resulting in a polyrhythm.

Subjective evaluation of the results shows numerous interesting emerging patterns not present in the corpus, but having similar statistical properties. However, many results combine highly random segments with monotonous portions, suggesting that global entropy measurements are not always adequate for longer pieces. We therefore propose further tests involving instantaneous entropy measured through time, as explored by Manzara et al. (1992).

While the combination of n-grams and entropy produce interesting results, further experimentation could be performed on the fitness evaluation metrics. For example, different weighting schemes for the sub-tests or normalization mechanisms for the n-grams may represent real-world quality more closely. We also propose experiments with other clustering algorithms, such as kernelized k-means, hierarchical or spectral clustering (Duda et al., 2000; Dhillon et al., 2004), studying their influence on the generated musical pieces.

# Thesis bibliography

Adler, D. (1993). Genetic algorithms and simulated annealing: a marriage proposal. In *IEEE International Conference on Neural Networks*, pages 1104–1109. IEEE.

Alfonseca, M., Cebrian, M., and Ortega, A. (2007). A simple genetic algorithm for music generation by means of algorithmic information theory. In *IEEE Congress on Evolutionary Computation*, pages 3035–3042. IEEE.

Angeline, P. J. (1996). Two self-adaptive crossover operators for genetic programming. In *Advances in genetic programming*, pages 89–109. MIT Press.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the 18th Annual Symposium on Discrete Algorithms (ACM-SIAM)*, pages 1027–1035, New Orleans, Louisiana. Society for Industrial and Applied Mathematics.

Bäck, T. (1992). Self-adaptation in genetic algorithms. In *Proceedings of the First European Conference on Artificial Life*, pages 263–271. MIT Press.

Bentley, J. (1986). Programming pearls: little languages. *Communications of the ACM*, 29(8):711–721.

Biles, J. A. (1994). GenJam: A Genetic Algorithm for Generating Jazz Solos. In *Proceedings of the 1994 International Computer Music Conference (ICMC)*, pages 131–137, Aarhus, Denmark. Michigan Publishing.

Bonissone, P. P., Subbu, R., Eklund, N., and Kiehl, T. R. (2006). Evolutionary algorithms + domain knowledge = real-world evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 10(3):256–280.

Brameier, M. F. and Banzhaf, W. (2007). *Linear genetic programming*. Springer Science & Business Media, 1st edition.

Cavicchio Jr., D. J. (1972). Reproductive adaptive plans. In *Proceedings of the ACM annual conference*, volume 1, pages 60–70, New York, New York, USA. ACM Press.

Chen, C.-C. J. and Miikkulainen, R. (2001). Creating Melodies with Evolving Recurrent Neural Networks. In *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, pages 2241–2246, Piscataway, NJ. IEEE.

Chew, E. (2002). The Spiral Array: An Algorithm for Determining Key Boundaries. In *Proceedings of the Second International Conference on Music and Artificial Intelligence (ICMAI)*, pages 18–31, Edinburgh, Scotland. Springer Berlin Heidelberg.

Chuan, C.-H. and Chew, E. (2005). Polyphonic Audio Key Finding Using the Spiral Array CEG Algorithm. In *Proceedings of the 2005 IEEE International Conference on Multimedia and Expo (ICME)*, pages 21–24, Amsterdam, The Netherlands. IEEE.

Conklin, D. (2003). Music generation from statistical models. In *Proceedings of the 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pages 30—-35, Aberystwyth, Wales. AISB.

Costelloe, D. and Ryan, C. (2007). Towards models of user preferences in interactive musical evolution. In *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO)*, pages 2254–2254, London, UK. ACM Press.

Cover, T. M. and Thomas, J. A. (2012). *Elements of information theory*. John Wiley & Sons.

Cramer, N. L. (1985). A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

Dahlstedt, P. (2007). Autonomous Evolution of Complete Piano Pieces and Performances. In *9th European Conference on Artificial Life*.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In *proc. 3rd International conference on genetic algorithms*, pages 61–69.

De Jong, K. (1988). Learning with genetic algorithms: An overview. *Machine learning*, 3(2-3):121–138.

De Prisco, R., Zaccagnino, G., and Zaccagnino, R. (2011). A multi-objective differential evolution algorithm for 4-voice compositions. In *2011 IEEE Symposium on Differential Evolution (SDE)*, pages 1–8. IEEE.

Dhillon, I. S., Guan, Y., and Kulis, B. (2004). Kernel k-means, spectral clustering and normalized cuts. In *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 551–556, Seattle, Washington, USA. ACM Press.

Dolin, B., Arenas, M. G., and Merelo, J. J. (2002). Opposites Attract: Complementary Phenotype Selection for Crossover in Genetic Programming. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN)*, PPSN VII, pages 142–152, London, UK. Springer-Verlag.

Donnelly, P. and Sheppard, J. (2011). Evolving Four-Part Harmony Using Genetic Algorithms. In *Applications of Evolutionary Computation*, volume 6625, pages 273–282. Springer, Berlin, Heidelberg.

Doraisamy, S. and Rüger, S. M. (2004). A Polyphonic Music Retrieval System Using N-Grams. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, pages 204–209, Barcelona, Spain. Universitat Pompeu Fabra.

Dostál, M. (2012). Musically meaningful fitness and mutation for autonomous evolution of rhythm accompaniment. *Soft Computing*, 16(12):2009–2026.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern classification*. John Wiley & Sons.

Eigenfeldt, A. (2009). The Evolution of Evolutionary Software: Intelligent Rhythm Generation in Kinetic Engine. In *EvoWorkshops*, volume 5484, pages 498–507. Springer.

Esquivel, S. C., Leiva, A., and Gallard, R. H. (1997). Multiple crossover per couple in genetic algorithms. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 103–106. IEEE.

Fernández, J. D. and Vico, F. (2013). AI methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48:513–582.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA)*, pages 416–423, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Fowler, M. (2010). *Domain Specific Languages*. Addison-Wesley Professional, 1st edition.

Gibson, P. M. and Byrne, J. A. (1991). NEUROGEN, musical composition using genetic algorithms and cooperating neural networks. In *Proceedings of the Second International Conference on Artificial Neural Networks (ICANN)*, pages 309–313, Bournemouth, UK. IET.

Gilreath, W. and Laplante, P. (2004). A one instruction set architecture for genetic algorithms. In *Biocomputing*, pages 91–113. Nova Science Publishers, Inc.

Gilreath, W. F. and Laplante, P. A. (2003). *Computer architecture: A minimalist perspective*, volume 730. Springer Science & Business Media.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Goldberg, D. E. and Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. In *Proceedings of an international conference on genetic algorithms and their applications*, volume 154, pages 154–159. Lawrence Erlbaum, Hillsdale, NJ.

Gomez, J. (2004). Self adaptation of operator rates in evolutionary algorithms. In *Genetic and Evolutionary Computation Conference*, pages 1162–1173. Springer.

Grefenstette, J. (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128.

Harries, K. and Smith, P. (1997). Exploring alternative operators and search strategies in genetic programming. *Genetic Programming*, 97:147–155.

Hartmann, P. (1990). Natural Selection of Musical Identities. In *Proceedings of the 1990 International Computer Music Conference (ICMC)*, pages 234–236, Glasgow, Scotland. Michigan Publishing.

Hofmann, D. M. (2015). A Genetic Programming Approach to Generating Musical Compositions. In *Proceedings of the 4th International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMUSART)*, volume 9027, pages 89–100, Copenhagen. Springer, Cham.

Horner, A. and Goldberg, D. E. (1991). Genetic Algorithms and Computer-Assisted Music Composition. In *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA)*, pages 437–441, San Diego, CA, USA. Morgan Kaufmann.

Horowitz, D. (1994). Generating Rhythms with Genetic Algorithms. In *AAAI*, volume 94, page 1459.

Hu, T., Banzhaf, W., and Moore, J. H. (2013). Robustness and evolvability of recombination in linear genetic programming. In *European Conference on Genetic Programming*, pages 97–108. Springer.

Jacob, B. (1995). Composing with Genetic Algorithms. In *International Computer Music Association*, pages 452–455.

Johanson, B. and Poli, R. (1998). GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters. In *Proceedings of the Third Annual Conference on Genetic Programming*, pages 181–186, University of Wisconsin, Madison, Wisconsin, USA. Morgan Kaufmann.

Jones, D. W. (1988). A Minimal {CISC}. *SIGARCH Comput. Archit. News*, 16(3):56–63.

Julstrom, B. A. (1997). Adaptive operator probabilities in a genetic algorithm that applies three operators. In *Proceedings of the 1997 ACM symposium on Applied computing*, pages 233–238. ACM Press.

Kazarlis, S., Papadakis, S., Theocharis, J., and Petridis, V. (2001). Microgenetic algorithms as generalized hill-climbing operators for GA optimization. *IEEE Transactions on Evolutionary Computation*, 5(3):204–217.

Klinger, R. and Rudolph, G. (2006). Evolutionary composition of music with learned melody evaluation. In *Proceedings of the Int. Conference on Computational Intelligence, Man-machine Systems and Cybernetics (CIMMACS'06)*.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press.

Krumhansl, C. L. (2001). A key-finding algorithm based on tonal hierarchies. In *Cognitive Foundations of Musical Pitch*, pages 77–110. Oxford University Press.

Li, X., Zhou, C., Xiao, W., and Nelson, P. C. (2005). Direct evolution of hierarchical solutions with self-emergent substructures. In *Machine Learning and Applications, 2005. Proceedings. Fourth International Conference on*, pages 6—-pp. IEEE.

Lidov, D. and Gabura, J. (1973). A Melody Writing Algorithm Using a Formal Language Model. *Computers in the Humanities*, 3-4:138–148.

Lipowski, A. and Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196.

Lo, M. and Lucas, S. M. (2007). N-gram fitness function with a constraint in a musical evolutionary system. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 4246–4251, Singapore. IEEE.

López, C. L. (2008). Heuristic Hill-Climbing as a Markov Process. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 274–284. Springer Berlin Heidelberg.

Loughran, R., McDermott, J., and O'Neill, M. (2016). Grammatical Music Composition with Dissimilarity Driven Hill Climbing. In *Proceedings of the 5th International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMUSART)*, pages 110–125, Porto, Portugal. Springer.

MacCallum, R. M., Mauch, M., Burt, A., Leroi, A., and M (2012). Evolution of music by public choice. *Proceedings of the National Academy of Sciences*, 109(30):12081–12086.

Madsen, S. T. and Widmer, G. (2007). Key-finding with Interval Profiles. In *Proceedings of the 2007 International Computer Music Conference (ICMC)*, pages 212–215, Copenhagen, Denmark. Michigan Publishing.

Mahfoud, S. W. (1995). *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign.

Manaris, B., Vaughan, D., Wagner, C., Romero, J., and Davis, R. B. (2003). Evolutionary Music and the Zipf-Mandelbrot Law: Developing Fitness Functions for Pleasant Music. In *Workshops on Applications of Evolutionary Computation*, pages 522–534. Springer, Berlin, Heidelberg.

Manaris, B. Z., Roos, P., Machado, P., Krehbiel, D., Pellicoro, L., and Romero, J. (2007). A Corpus-Based Hybrid Approach to Music Analysis and Composition. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 839–845. AAAI Press.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

Manzara, L. C., Witten, I. H., and James, M. (1992). On the Entropy of Music: An Experiment with Bach Chorale Melodies. *Leonardo Music Journal*, 2(1):81.

Martins, J. M. and Miranda, E. R. (2007). Emergent Rhythmic Phrases in an A-Life Environment. In *Proceedings of ECAL 2007 Workshop on Music and Artificial Life (MusicAL 2007)*, pages 11–14.

Masataka, N. (2007). Music, evolution and language. *Developmental Science*, 10(1):35–39.

Mavaddat, F. and Parhami, B. (1988). URISC: the ultimate reduced instruction set computer. *International Journal of Electrical Engineering Education*, 25(4):327–334.

McCormack, J. (1996). Grammar based music composition. *Complex systems*, 96:321–336.

McIntyre, R. A. (1994). Bach in a box: the evolution of four part Baroque harmony using the genetic algorithm. In *Proceedings of the IEEE First World Congress on Computational Intelligence*, pages 852–857, Orlando, Florida, USA. IEEE.

McKay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., and O'Neill, M. (2010). Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81.

Miranda, E. R. and Biles, J. A. (2007). *Evolutionary Computer Music*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

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

Munroe, D. R. (2004). Genetic Programming: The ratio of Crossover to Mutation as a function of time. *Research Letters in the Information and Mathematical Sciences*, 6:83–96.

Murata, T. and Ishibuchi, H. (1995). MOGA: multi-objective genetic algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, page 289.

Nordin, P. and Banzhaf, W. (1995). Evolving Turing-Complete Programs for a Register Machine with Self-modifying Code.

Nordin, P., Banzhaf, W., and Francone, F. D. (1999). Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. *Advances in genetic programming*, 3:275–299.

Nuanáin, C. b., Herrera, P., and Jordá, S. (2015). Target-Based Rhythmic Pattern Generation and Variation with Genetic Algorithms. In *Proceedings of The 12th Sound and Music Computing Conference*, Maynooth, Ireland.

Oltean, M., Gro\csan, C., Dio\csan, L., and Mih\uail\ua, C. (2009). Genetic programming with linear representation: a survey. *International Journal on Artificial Intelligence Tools*, 18(02):197–238.

O'Reilly, U.-M. and Oppacher, F. (1995). Hybridized crossover-based search techniques for program discovery. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, volume 2, pages 573–578. IEEE.

Ortega, A., Alfonso, R. S., and Alfonseca, M. (2002). Automatic Composition of Music by Means of Grammatical Evolution. In *Proceedings of the 2002 International Conference on APL: Array Processing Languages: Lore, Problems, and Applications*, pages 148–155, Madrid, Spain. ACM Press.

Patterson, D. A. and Hennessy, J. L. (2013). *Computer organization and design: the hardware/software interface*. Newnes.

Pearce, M. T. (2005). *The construction and evaluation of statistical models of melodic structure in music perception and composition*. PhD thesis, City University London.

Perkis, T. (1994). Stack-based genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 148–153 vol.1.

Phon-Amnuaisuk, S., Law, E. H. H., and Kuan, H. C. (2007). Evolving Music Generation with SOM-Fitness Genetic Programming. In *Applications of Evolutinary Computing*, pages 557–566. Springer Berlin Heidelberg, Berlin, Heidelberg.

Phon-Amnuaisuk, S., Tuson, A., and Wiggins, G. (1999). Evolving Musical Harmonisation. In *Proceedings of the 1999 International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 229–234, Portorož, Slovenia. Springer.

Piszcz, A. and Soule, T. (2006). Genetic Programming: Analysis of Optimal Mutation Rates in a Problem with Varying Difficulty. In *FLAIRS Conference*, pages 451–456.

Plomp, R. and Levelt, J. M. (1965). Tonal Consonance and Critical Bandwidth. *The Journal of the Acoustical Society of America*, 38(4):548–560.

Poli, R. and Langdon, W. B. (1998). On the search properties of different crossover operators in genetic programming. *Genetic Programming*, pages 293–301.

Poli, R., Langdon, W. B., McPhee, N. F., and Koza, J. R. (2008). *A field guide to genetic programming*. Lulu. com.

Reddin, J., McDermott, J., and O'Neill, M. (2009). Elevated Pitch: Automated Grammatical Evolution of Short Compositions. In *Applications of Evolutionary Computing*, volume 5484 of *Lecture Notes in Computer Science*, pages 579–584. Springer Berlin Heidelberg.

Renders, J.-M. and Bersini, H. (1994). Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence.,*, pages 312–317. IEEE.

Roy, D. B., Das, P., and Mukhopadhyay, D. (2015). ECC on Your fingertips: a single instruction approach for lightweight ECC design in GF (p). In *International Conference on Selected Areas in Cryptography*, pages 161–177. Springer.

Russell, S. J. and Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Malaysia; Pearson Education Limited.

Ryan, C., Collins, J. J., and O'Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Conference on Genetic Programming (EuroGP)*, pages 83–96, Paris, France. Springer.

Selfridge-Field, E. (2004). Music Theory for Computer Applications.

Spector, L. and Robinson, A. (2002). Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

Srinivas, M. and Patnaik, L. M. (1994a). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667.

Srinivas, M. and Patnaik, L. M. (1994b). Genetic Algorithms: A Survey. *Computer*, 27(6):17–26.

Sulyok, C. (2018). Genetic Operators for Evolutionary Music Composition. In Abraham, E., Negru, V., Petcu, D., Zaharie, D., Ida, T., Jebelean, T., and Watt, S., editors, *Proceedings of the 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 253–259, Timisoara, RO.

Sulyok, C. (2019). Towards Automated Quality Assessment Methods in Algorithmic Music Composition. In *Proceedings of the 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) (accepted for publication)*, Timisoara, RO.

Sulyok, C. and Harte, C. (2017). On Virtual Machine Architectures for Evolutionary Music Composition. In *Proceedings of the 14th European Conference on Artificial Life (ECAL)*, pages 577—-584, Lyon. MIT Press.

Sulyok, C., Harte, C., and Bodó, Z. (2019a). On the Impact of Domain-specific Knowledge in Evolutionary Music Composition. In *Proceedings of the 2019 Genetic and Evolutionary Computation Conference (GECCO)*, pages 188–197, Prague, CZ. ACM Press.

Sulyok, C., McPherson, A., and Harte, C. (2015). Corpus-taught Evolutionary Music Composition. In *Proceedings of the 13th European Conference on Artificial Life (ECAL)*, pages 587–594, York, UK. The MIT Press.

Sulyok, C., McPherson, A., and Harte, C. (2019b). Evolving the process of a virtual composer. *Natural Computing*, 18(1):47—-60.

Swain, J. P. (1986). The need for limits in hierarchical theories of music. *Music Perception: An Interdisciplinary Journal*, 4(1):121–147.

Temperley, D. (2001). *The cognition of basic musical structures*. MIT Press.

Thywissen, K. (1999). GeNotator: An Environment for Exploring the Application of Evolutionary Techniques in Computer-assisted Composition. *Org. Sound*, 4(2):127–133.

Tokui, N. and Iba, H. (2000). Music composition with interactive evolutionary computation. In *Proceedings of the Third International Conference on Generative Art*, volume 17, pages 215–226, Milan, Italy.

Towsey, M., Brown, A., Wright, S., and Diederich, J. (2001). Towards Melodic Extension Using Genetic Algorithms. *Educational Technology and Society*, 4(2):54–65.

Turing, A. M. (1950). Computing Machinery and Intelligence. In *Mind*, volume 59, pages 433–460.

Tuson, A. and Ross, P. (1998). Adapting operator settings in genetic algorithms. *Evolutionary computation*, 6(2):161–184.

Ukkonen, E., Lemström, K., and Mäkinen, V. (2003). Geometric Algorithms for Transposition Invariant Content-Based Music Retrieval. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, pages 193–199, Maryland, USA. Johns Hopkins University Press.

Unemi, T. (2002). SBEAT3: A Tool for Multi-part Music Composition by Simulated Breeding. In *Proceedings of the 8th international conference on artificial life*, pages 410–413. MIT Press.

Vafaee, F. and Nelson, P. C. (2009). Self-adaptation of genetic operator probabilities using differential evolution. In *Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems. SASO'09.*, pages 274–275. IEEE.

Vafaee, F., Nelson, P. C., Zhou, C., and Xiao, W. (2008). Dynamic adaptation of genetic operators' probabilities. *Studies in Computational Intelligence*.

Waschka II, R. (2007). Composing with Genetic Algorithms: GenDash. In *Evolutionary Computer Music*, pages 117–136. Springer London.

Whorley, R. and Conklin, D. (2016). Music Generation from Statistical Models of Harmony. *Journal of New Music Research*, 45(2):160–183.

Whorley, R., Rhodes, C., Wiggins, G., and Pearce, M. (2013). Harmonising Melodies: Why do we add the bass line first? In *International Conference on Computational Creativity*, pages 79–86, Sydney.

Wiggins, J. (2007). Compositional process in music. In *International handbook of research in arts education*, pages 453–476. Springer Netherlands.

Wolkowicz, J., Heywood, M., and Keselj, V. (2009). Evolving indirectly represented melodies with corpus-based fitness evaluation. In *Workshops on Applications of Evolutionary Computation*, pages 603–608, Tübingen, Germany. Springer Berlin Heidelberg.

Wolkowicz, J., Kulka, Z., and Keselj, V. (2008). N-gram-based approach to composer recognition. *Archives of Acoustics*, 33(1):43–55.

Worth, P. and Stepney, S. (2005). Growing Music: Musical Interpretations of L-Systems. In *EvoWorkshops 2005: Applications of Evolutionary Computing*, pages 545–550. Springer, Berlin, Heidelberg.

Wu, C. L., Liu, C. H., and Ting, C. K. (2014). A novel genetic algorithm considering measures and phrases for generating melody. In *Proceedings of the 2014 Congress on Evolutionary Computation (CEC)*, pages 2101–2107, Beijing, China. IEEE.