On the Impact of Domain-specific Knowledge in Evolutionary Music Composition

Csaba Sulyok
Babeş-Bolyai University, Faculty of Mathematics and Computer Science
Cluj-Napoca, Romania
csaba.sulyok@gmail.com

Christopher Harte
Melodient Ltd.
London, UK
chris@melodient.com

Zalán Bodó
Babeş-Bolyai University, Faculty of Mathematics and Computer Science
Cluj-Napoca, Romania
zbodo@cs.ubbcluj.ro

ABSTRACT

In this paper we investigate the effect of embedding different levels of musical knowledge in the virtual machine (VM) architectures and phenotype representations of an algorithmic music composition system. We examine two separate instruction sets for a linear genetic programming framework that differ in their knowledge of musical structure: one a Turing-complete register machine, unaware of the nature of its output; the other a domain-specific language tailored to operations typically employed in the composition process. Our phenotype, the output of the VM, is rendered as a 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.

To derive a fitness metric, we extract musical features from a corpus of Hungarian folk songs in the form of n-grams and entropy values. Fitness is assessed by extracting those same attributes from the phenotype and finding the maximal similarity with representative corpus features.

With two different VM architectures and three pitch schemes, we present and compare results from a total of six configurations, and analyze whether the domain-specific knowledge impacts the results and the rate of convergence in a beneficial manner.

CCS CONCEPTS

• **Computing methodologies → Genetic programming**; • **Applied computing → Sound and music computing**; • **Theory of computation →** *Evolutionary algorithms*;

KEYWORDS

evolutionary music composition, genetic programming, domain-specific languages

ACM Reference Format:
Csaba Sulyok, Christopher Harte, and Zalán Bodó. 2019. On the Impact of Domain-specific Knowledge in Evolutionary Music Composition. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3321707.3321710

1 INTRODUCTION

The process of music composition may be seen as an algorithmic search for acceptable solutions. Biles [3] states that "composers search for the right chords to fit a melody or the right melody to fit a chord progression; arrangers search for the right voicings and counterpoint for constituent parts; improvisers search for the right phrases to play over a particular set of chord changes". The only difference between composition and improvisation is that in the latter case the musician has to accept the first solution he arrives to, while a composer can refine and polish their musical idea. Improvisation, however, is often considered part of the composition process, more precisely of the exploration phase. Indeed, all properties and aspects of music creation may be modeled by heuristics with different definitions of the problem space and the sub-space of possible solutions.

Composition is usually considered as a sequence of interrelated and recurrent processes. Since music interpretation and composition is highly influenced by the socio-cultural environment, it is very likely that ideas come from music the composers already know, mixing into their own new ideas. Studying individual and collaborative composing processes of students, it was observed that it is not uncommon for them to experiment with known musical pieces as a starting point, sometimes even listening to their previous works to get inspiration for new melodies [61]. This feedback of the creative self suggests evolutionary algorithms may be a good modeling tool for music creation [54]. Inspired by the sequential nature of composition, the current work uses linear genetic programming (LGP) [5] as a means of creating programs which compose instead of evolving the music itself. These genotype programs are interpreted by a virtual machine and produce outputs which can be assessed separately. Such an evolutionary system may be used to evolve any kind of output where the process of its creation may be modeled separately from the product.

Given the personal perception of music, or any art form in general, an objective definition of a "good" output is difficult to obtain [59]. The current research measures output quality as distributional similarity to existing works in a corpus of real music [54], i.e. proximity to members increases the score of each individual. Corpus pieces are not used to seed the initial population; instead, they inform the fitness tests, allowing a broader search space.

However, the question arises: "Is it helpful for the system to be informed to some extent about the nature of its own output?" Given the likelihood of benefits gained from domain-specific knowledge [4, 11], we propose a comparison between setups that differ in their awareness of the end goal of creating music. A virtual machine instruction set with music-related commands is proposed, which reduces the otherwise vast array of possible movements to ones typically found in music creation, making it a *domain-specific language* (DSL) [20]. We also propose a diatonic pitch representation, which allows the notions of *key* and *interval* to be clear to the system [53]. This allows phenotype pitch values to be seen less as representing absolute frequencies and more as positions in a scale, clarifying their purpose in the work. We first analyze the corpus offline to determine the key of each piece, then create shifted pitch representations which can be compared to phenotype pitches with the same metrics as their original version.

Restricting the building blocks (pitches, rhythms, etc.) a composer is allowed to use can hinder the invention of novel ideas [61]. Implicitly injecting knowledge about the problem at hand—by using a DSL or diatonicity—does not restrict the possible musical pieces; the sole constraint is introduced by comparing the generated music to the corpus.

The remainder of the document is structured as follows: Section 2 discusses previous research in the relevant fields of our research. An overview of the LGP system used in the experiments is outlined in Section 3, followed by detailed descriptions of the phenotype representation, its rendering from a genotype, and the quality assessment in Sections 4, 5 and 6, respectively. The experiments and results obtained are detailed in Section 7, with conclusions drawn and discussed in Section 8.

2 BACKGROUND AND RELATED WORK

The current section discusses previous related works in evolutionary music composition, n-grams in music, domain-specific languages and key detection.

2.1 Evolutionary music

Evolutionary approaches to generating music represent an active topic of research since the early 1990s [21, 22, 24]. Miranda and Biles [43] provide a comprehensive overview of evolutionary computer music, while Fernandez and Vico [18] summarize general AI approaches. Early examples of success include GenJam [3], a jazz solo improvisation software powered by a genetic algorithm, and GenDash [59], an auxiliary tool for concert music composition.

Since the search for "good" music gives way to a large and complex search space, certain studies narrow it by deploying genetic algorithms to evolve only certain musical traits, such as melodic phrases [64], harmonization [14, 39, 48], chords/chord progressions [44] or rhythmic patterns [57].

Besides standard genetic algorithms, the formal rules of music also suggest grammatical evolution methods [40, 52] as a viable approach. Indeed, formal languages for music have been proposed since the early 1970s [28, 38]. Ortega et al. [46] use a context-free grammar to automatically generate melodies for the AP440 processor. Evolutionary systems that create structures for such grammars therefore become a potential solution for including music-specific knowledge [31, 51].

An especially challenging aspect of generative art systems is quality assessment; the subjective nature of the output adds weight to the choice of an appropriate fitness function [59]. Therefore, many systems incorporate interactive evolutionary computation (IEC), where users provide ratings for output phenotypes [44, 57]. Some others use semi-automated fitness evaluation approaches informed by user feedback. For example, the GP-Music system [25] uses interactive genetic programming, also adding a neural network which learns the user preferences. Costelloe and Ryan [9] use grammatical evolution to mimic personal feedback.

Automated music fitness metrics often rely on a corpus of musical pieces for external information. Lo and Lucas [30] use a corpus to determine the fitness of an algorithmically created melody by calculating the probability that it was generated by the underlying Markov model. Wolkowicz et al. [62] evaluate fitness by taking into account the frequency and specificity of musical patterns present in the generated music and the corpus. Whorley and Conklin [60] propose an improved statistical sampling procedure to generate low cross-entropy music, based on a training set. While using statistical similarity with a corpus as musical fitness evaluation has been discussed before [17, 33], it has not been attempted in a linear genetic programming setting.

While genetic algorithms assess the same entity that is bred and/or mutated, genetic programming [26] (GP) provides a conceptual separation between the object of evolution and the object of assessment. The former is represented by the genotype, the genetic code subject to mutation and crossover, while the latter is the phenotype, the set of observable traits which influences the fitness and therefore the probability of survival. GP allows the evolution of programs which create music instead of splicing and evolving the pieces directly [23, 25].

The approach presented here uses linear genetic programming [5], a form of GP where genotypes are sequentially interpreted code segments running inside a virtual machine. The linear execution represents the thought process of a composer [54]. The result is similar to that of grammatical evolution, where the instruction set of the virtual machine determines the grammar by which all genotypes are interpreted.

2.2 N-grams in music

One of the most popular and successful representational models in information retrieval and natural language processing is *bag-of-words* whereby a natural language text is represented as a bag of terms occurring in it [34]. Despite its popularity, the drawback of the bag-of-words model is that every term is considered separately; the order of terms, which sometimes contains valuable information, is not taken into account. Generalizing words to sequences we obtain the *n-gram* model: a term sequence of length n occurring in the document. N-grams are frequently used in language modeling, estimating the probability of a word given its history; this is approximated by considering just the last few words [35], $P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-n+1}^{k-1})$, w_i^j denoting the sequence of terms $w_i, w_{i+1}, \ldots, w_j$.

Since language and music might have a common evolutionary ancestor [37], it is not unusual to apply similar models in representation and processing. Indeed, applications of n-grams can be found in music information retrieval and automatic music generation. Doraisamy and Rüger [15] use combined n-grams of pitch differences and onset interval ratios for indexing and retrieving polyphonic music. Conklin [8] discusses statistical models for generating music, including context models related to n-grams: random

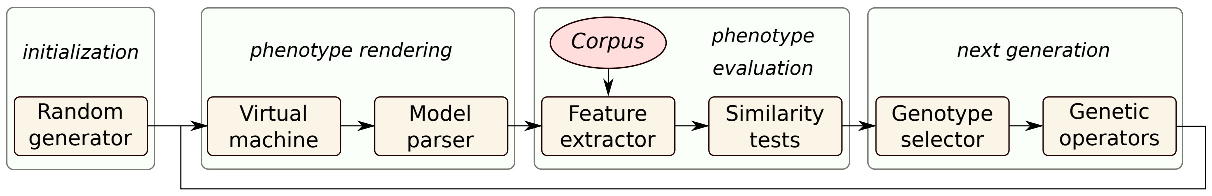


Figure 1: The general flow of our approach follows standard genetic programming conventions

walk, hidden Markov models and different sampling techniques. Lo and Lucas [30] investigate the use of n-grams in automatic music generation with evolutionary algorithms. They also assess the capability of the n-gram representational model in music categorization. Their results show that the pitch difference bigram model generates melodies with more varied notes and higher fitness scores, with the generated music also being more "interesting". Wolkowicz et al. [63] present an n-gram-based approach to music categorization, similar to [15], using n-grams of pitch difference and duration ratio pairs to represent music. In [62] they use the same representation for evolving melodies with a corpus-based fitness function. They also employ a weighting scheme to give higher scores to more useful patterns. For an exhaustive review of n-grams in music-related computational tasks, see [47].

2.3 Domain-specific languages

Domain-specific languages (DSLs), also called little or micro(scopic) languages [2], are used to cover specific application domains. Unlike general purpose languages (GPLs), DSLs need not to be computationally universal. Notable examples include the TEX typesetting system, SQL for managing data in a relational database management system, or even Lex/Flex and Yacc/Bison used for designing new programming languages. There is, however, no sharp line between the two categories: DSLs can have tailored applicability, but can also be capable of describing complex algorithms [41].[1] High-level programming languages have evolved from machine code in order to make programming more efficient, obtaining significant advancements in correctness checking, productivity, maintenance and reuse—requirements that also apply to DSLs.

Composing music is simpler using a DSL than a universal language, therefore a variety of audio programming languages have been developed, including specialized libraries written in a GPL, functional languages, markup languages or visual audio programming languages[2]. These mostly aid music representation instead of depicting the composition process; we therefore propose a new DSL as a virtual machine instruction set for music generation.

2.4 Key detection

Western tonal music generally recognizes pitches from the chromatic scale: an atonal repeating set of 12 notes. Adjacent notes are one semitone apart and obey the equal temperament tuning system, i.e. the ratio of their frequencies remains constant [53].

[1]TEX, for example, is a Turing complete language.
[2]See, for example, LilyPond (http://lilypond.org/), JSyn (http://www.softsynth.com/jsyn/), Pure Data (http://puredata.info), etc.

Diatonic scales designate a subset of possible notes as bases for musical compositions. These generally contain 7 out of the 12 chromatic notes. The group of possible notes make up the *key* of the piece, whose first degree and tonal center is known as the *tonic* or *root* note. While all pitches in a diatonic piece may be mapped to the chromatic scale, their purpose lies in their *interval*: the relationship to the tonic. Harmonically pleasing and stable intervals are named consonant, with the opposite dubbed dissonant [49].

Digital music transcription standards such as Musical Instrument Digital Interface (MIDI)[3] often encode in chromatic representations, retaining only absolute pitches and losing all key information. To analyze such data tonally, a conversion to a diatonic representation is necessary, which implies content-based key detection.

One of the earlier and most widely used key detection methods is the Krumhansl–Schmuckler algorithm [27], where key profiles are constructed with relative note importance values. Key is detected by searching for maximum correlation with these profiles. Many extensions/adaptations have been proposed for this approach, e.g. Temperley's [56] preference rule-based music cognition model uses an altered algorithm, which performs segmentation of beats and labels the segments with the appropriate keys. Madsen and Widmer [32] also extend the Krumhansl–Schmuckler method by incorporating temporal cues in the profiles.

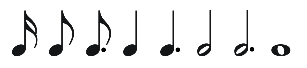
More recently, geometric models have been proposed for pitch representation in key detection algorithms. Ukkonen et al. [58] cast notes as points in a Euclidean space, where distances to reference planes become key metrics. Similarly, Chew [6] proposes the Spiral Array Model pitch representation together with the Center of Effect Generator (CEG) key-finding algorithm. This method is also used for key detection in polyphonic audio signals [7]. The current research uses the CEG algorithm to find keys in a corpus of MIDI files, allowing conversion between different pitch representations.

3 METHODOLOGY OVERVIEW

The system we present here[4] uses conventional genetic programming patterns (see Figure 1). A fixed size population is maintained throughout a run, represented by genotypes, phenotypes and fitness test scores of individuals.

Our 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

[3]`https://www.midi.org/`
[4]Project hosted open-source at `https://gitlab.com/emcrp`



Duration values	0	1	2	3	4	5	6	7
Mapped MIDI Δ_t values	1	2	3	4	6	8	12	16

Figure 2: Discrete possible durations of the musical phenotype notes together with the corresponding MIDI values, assuming 4 ticks per quarter note and a tempo of 120BPM

VM, after which the programs are executed (see Section 5) then the output bytes are collected and parsed into phenotype musical models (see Section 4).

We define quality of an output piece as its similarity to a corpus of real-world music that we import into the system from standard MIDI files. To avoid making a direct comparison between corpus members and incoming pieces (which would make overfitting likely), relevant features are extracted from the corpus offline, including n-grams and entropy values. Phenotypes output by the VM undergo the same feature extraction process, after which similarity is measured. Different weights may be assigned to the fitness metric along different features or feature categories, making the algorithm multi-objective [19] (see Section 6).

After phenotype evaluation, a new generation is constructed using genetic operators. A percentage of the new population is created through reproduction, a process representing survival of the fittest. Surviving individuals are selected based on their overall fitness and age using roulette-wheel selection [29]. The rest of the population is bred using a combination of crossover and mutation. Complementary phenotype selection [13] is used to choose parents, increasing the likelihood of pairing together individuals with different strengths. Offspring are created using homologous crossover [45]: randomly selected cut points are generated and the new genetic strings are built by swapping aligned portions from the parents. Mutation is applied to both generated offspring by re-randomizing a percentage of their genetic code.

4 THE MUSICAL PHENOTYPE

The phenotype of the proposed system is a simple piece of music containing a series of notes, each represented by duration and pitch. The representation is a byte sequence similar to the MIDI protocol (but more lightweight) and standard MIDI files may easily be imported/exported for subjective evaluation.

We choose 8 common duration values which may be mapped to MIDI temporal units as seen in Figure 2. Over 99.5% of note durations in the corpus fall into one of these values. Furthermore, no expression elements such as tempo changes or notes falling outside the simple-time metric structure (e.g. no triplet rhythms etc.) are present in the corpus, therefore the mapping between the model temporal unit and real-world time is global. We map one unit of time as a sixteenth note with a tempo of 120 quarter-note beats per minute; therefore one unit in real-world time is 0.125 seconds. The corpus contains single track vocal melodies with no rests or overlapping notes, therefore note onset and offset times are fully embedded in the duration, i.e. each note begins when the previous ends.

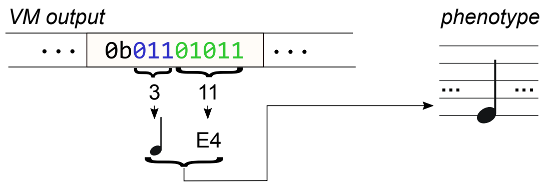


Figure 3: Building musical models from virtual machine output bytes. Each byte becomes a note whose 3 most significant bits are parsed as the duration, while the other 5 become the pitch.

Our pitch value is represented by an integer in the range from 0 to 31 with a flexible mapping to MIDI pitches (further information on our mapping in Section 4.1).

The system supports representing the velocity/volume of a note as well, however we have not used it in this study because our corpus pieces have constant velocity (this suggests the MIDI files in the corpus were generated from musical scores rather than recorded by musicians).

Creating a phenotype is equivalent to parsing the output of the virtual machine (see Section 5) which may be unaware of the nature of its output, therefore it may produce any byte array. The phenotype renderer parses the output bytes and creates a musical model.

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 3).

4.1 Diatonic pitch representation

MIDI represents pitch numerically in 7 bits, with the value 64 mapped to the note E4 and single increments/decrements for semitone steps. Analyzing the corpus reveals that 99.7% of pitches fall into a range 32 pitches wide between 53 and 84, allowing us to use an encoding in 5 bits.

As mentioned in Section 2.4, 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 phenotype, with the possibility of conversion between each (see Figure 4). 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 [49] and this representation allows pitch values to always take on the same function, e.g. the value 18 is always a perfect fifth.

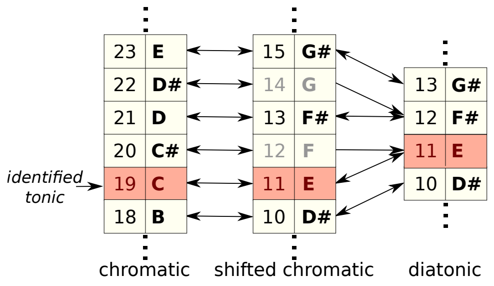


Figure 4: Mapping between the 3 representations of phenotype pitch. The detected key is C major; all notes are transposed/quantized accordingly.

- *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.

To convert from chromatic to any of the other representations, the key of the piece k is identified using the CEG algorithm [6]. Transposing all pitches with $11 - k$ results in the shifted chromatic version. Furthermore, mapping the new differences from 11 to the heptatonic scale yields the diatonic representation. The key of each piece is retained, so inverse conversions may be performed later.

The conversion to diatonic implies irreversible rounding of the pitches to the nearest harmonic neighbor, e.g. the augmented fourth (tritone) is rounded to a perfect fourth. While this results in some information loss, it guarantees harmonic correctness. Since the pitches are packed together more tightly (a delta of 7 instead of 12 represents a shift by one octave), the designated 32 values hold a wider pitch range than the other schemes.

When evolving pieces for the shifted chromatic or diatonic representations, a key may be assigned manually, therefore all models may be exported and listened to as MIDI. In our experiments, we use a key of E major, ensuring no information loss on border values when transposing.

5 VIRTUAL MACHINE

To decouple the music composition process from the output representation, we define our phenotype renderer as a virtual machine whose output is used to build the musical model. A genetic string fully represents a state of the VM.

In this section we present two VM architectures modeled after 8-bit von Neumann register machines found on classical microprocessors. 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 (see Figure 5).

An instruction set may contain many kinds of commands that manipulate data within the VM, such as data transfer, arithmetic, conditionals, etc. To produce our musical model, we define a special

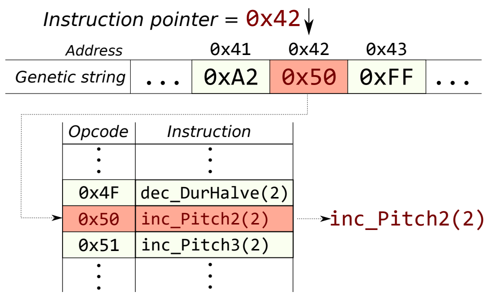


Figure 5: The virtual machine interprets genetic strings by mapping encountered 8-bit values in the RAM to instructions. Output instructions produce the bytes used to build the phenotype.

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 a given 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. The number of expected output bytes is configurable; in our experiments, we use the mean note count of the corpus (further discussed in Section 7).

5.1 Architectures and instruction sets

Our virtual machine architectures differ in their embedded knowledge of the nature of their output: one is a general-purpose von Neumann machine [54] (we will refer to this as the *GP machine*), while the second uses a domain-specific language with musical movements (the *DSL machine*)[5].

Common properties of the two machines include a 64kB random access memory (RAM) space which stores both instructions and data, allowing self-rewriting during execution. Being 8-bit machines, 256 different instructions are provided for each. The instruction pointer and an auxiliary data pointer may address any byte in memory; moving the former represents a jump. There are no actions in either of our instruction sets that generate random values, therefore interpreting the same genetic string twice using the same architecture always yields the same output—this allows reproduction and analysis of any result.

The GP instruction set is loosely based on the Intel 8080 architecture, with the addition of outputs. 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 (a) data transfer (b) arithmetic, logic, (c) branching, conditional and (d) output instructions.

The DSL is designed around knowledge of its output, therefore notes in the memory of the virtual composer are cast as 8-bit registers. The literature suggests various values for composer short-term memory capacity; these range from three or four [55] all the way

[5]Detailed descriptions of the instruction sets and memory segments are available at `https://emcrp.gitlab.io`

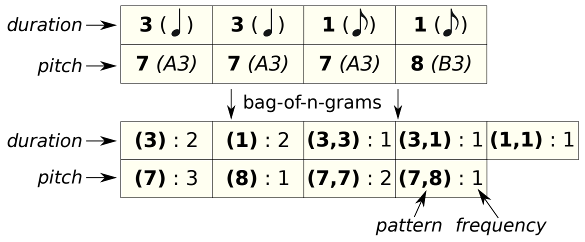


Figure 6: Demonstrating the construction of phenotype n-grams (uni- and bigrams) for comparison with corpus members.

up to seven [42] items—however these values are highly context-sensitive. In this study we have chosen to use 6 registers to model in-memory notes.

DSL instructions also include transfer, branching and conditional instructions, with the addition of data transfer acting only on the duration of a note, i.e. operating only on the most significant 3 bits in the byte represents a duration overwrite. Data manipulation instructions for the note registers also 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. The pitch representation and key are global settings of the DSL, it can therefore perform pitch change instructions keeping with the present representation. For example, shifting up a "third" means taking two steps up in the diatonic scale (since the base note is designated as the "first"). If the pitch representation is diatonic, the DSL increases the pitch value by 2; in a chromatic case however it will increment with the appropriate number of semitones to reach either the major or minor third (these are increments of 4 and 3, respectively), based on which of the two is consonant in the diatonic scale. Each note manipulation instruction has two variants: one performs just the manipulation, the other also outputs the changed note afterwards increasing the otherwise low number of available output instructions.

6 FITNESS ASSESSMENT

The quality of a phenotype is measured using statistical similarity to a corpus of real-world music. We extract relevant features from the corpus including n-grams of musical patterns and entropy measures, and these are used as the basis for our comparison.

6.1 The corpus

For our experiments in this study we use the "Cimbalom" Hungarian folk song collection[6] as our corpus. The downloaded MIDI files are all monophonic, comprising only vocal arrangements with no accompanying instruments. The online "Népdalok" sub-collection contains 4192 MIDI links, from which we discard 21 duplicates and 6 corrupt files, leaving a corpus of size 4165. This collection is chosen for its relative simplicity, stylistic homogeneity and brevity; the average length of the pieces is 25 seconds.

Before any experiments are conducted using the downloaded corpus, the following pre-processing steps are performed:

[6]Corpus downloaded from `http://www.cimbalom.nl/nepdalok.html`

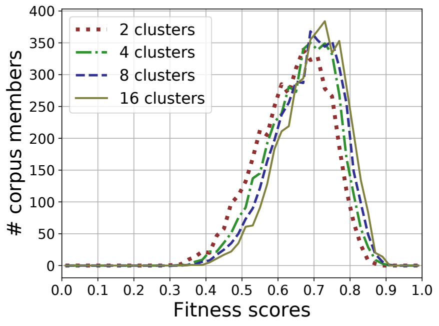


Figure 7: Fitness distribution when testing corpus members against the n-gram-based container. When increasing the number of clusters k, the fitness also rises. We use $k = 8$ in our experiments based on this.

- corpus members are converted to the internal model representation of the system (see Section 4);
- empty and corrupt files are discarded;
- notes with zero velocity are removed;
- the tracks are sub-sampled with no information discarded, to fit the tempo and time layout presented in Section 4;
- any overlapping notes are clipped.

6.2 N-gram-based tests

Since our method applies a corpus-based fitness function, n-grams offer a convenient way of representing musical patterns for our fitness tests. In order to compare two musical pieces, 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,

$$\mathrm{sim}(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1^{\mathrm{T}} \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|} \tag{1}$$

Conversely, the similarity will be minimal if their vectors are linearly independent, in this case meaning that no common n-grams can be found in their representations. In Eq. (1) $\mathbf{x}_i, i \in \{1, 2\}$ are the vector representations of the musical pieces under assessment, $\| \cdot \|$ denoting the Euclidean norm.

Our fitness function incorporates two base properties of the melodies: 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. The combined properties would likely be sufficient to model musical patterns present in the melodies; the other four components are introduced into our metric to allow generated music to diverge from the corpus: (a) and (b) enable the algorithm to consider pitch and duration patterns independently, while (d) and (e) allow variations achieved by transposition and tempo modification. The n-gram-based fitness metric therefore comprises 6 values representing similarity of these features. Figure 6 shows the construction of the uni- and bigram features for (a) and (b) of an example piece.

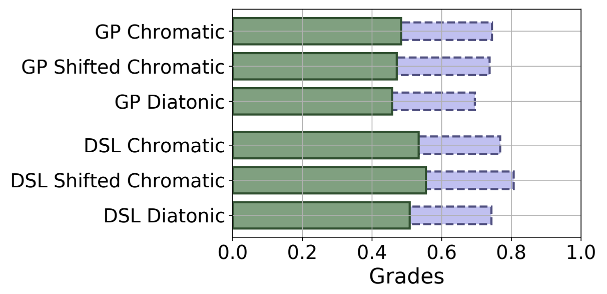


Figure 8: Last generation mean and maximum fitness values per configuration averaged over the 20 test runs

To perform the corpus-based evaluation, we compare the generated music to "representative" corpus members obtained using the k-means++ clustering algorithm [1]. We choose a number of clusters based on how well the corpus members perform in the tests. Figure 7 shows the fitness distribution for different values of k.

6.3 Entropy measures

Shannon entropy [10] measures information density by estimating the smallest number of bits necessary to represent data. 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 in a piece of music.

Analyzing the corpus entropy in terms of duration, pitch and the combination of both reveal that the values follow normal distributions. The mean μ and standard deviation σ of the entropy for the three properties is calculated for the entire corpus, and the entropy-based fitness of a phenotype is computed by measuring deviation using the Gaussian function,

$$g(X) = \exp\left(-\frac{(\mathrm{H}(X) - \mu)^2}{2\sigma^2}\right) \qquad (2)$$

where X is the respective property of the individual as a random variable, and $\mathrm{H}(X)$ denotes its entropy.

6.4 Fitness evaluation

The similarity tests concern three properties (duration, pitch, and their combination) and three transforms (n-grams of values, n-grams of differentials and entropy, as described in the previous sections) resulting in a total of 9 grades. The final fitness score is calculated by a linear combination of these sub-scores with a set of weights summing to 1, the overall score therefore lying in the interval $[0, 1]$.

7 EXPERIMENTS AND RESULTS

Our aim here is to explore if knowledge about the nature of the object of evolution impacts the results and rate of convergence of the system so we vary the following settings in the different trials:

- the virtual machine is either the GP machine or the DSL machine presented in Section 5.1;

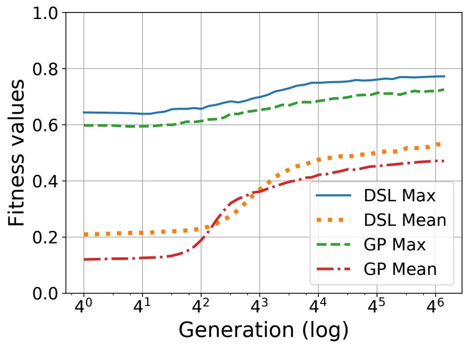


Figure 9: Progress through the generation for the 2 VM architectures. The DSL outperforms the general-purpose machine even in early generations.

- the representation of phenotype pitch is either chromatic, shifted chromatic or diatonic, as seen in Section 4.1.

The combination of these parameters results in six different sets of parameters. All other settings remain constant throughout the experiments: we use the corpus presented in Section 6 with 8 cluster centres, and a population size of 500. As commonly suggested, 8% of the population is created using reproduction [50]; the remainder are bred through a combination of homologous crossover—where the number of cut points is 0.2% the size of the genetic string—and mutation with a rate of 2%. Duration and pitch patterns are incorporated into the evaluation model using unigrams, bigrams and trigrams, as described in Section 6.2. The overall fitness of a musical piece is computed giving equal weights to all 9 sub-scores (see Section 6). For each setting, we run 20 simulations, allowing the program to reach 5000 generations each time[7]. The average length of the corpus pieces is 25 seconds, therefore all simulations are set up to create outputs of that length. Corpus analysis reveals an average of around 2.5 notes per second, therefore the stopping condition of the virtual machines is set to 63 bytes (which is equivalent in this case to the number of output notes).

Figure 8 depicts the mean and maximum grades of the population in its last generation, averaged over the 20 runs. We can conclude that in all pitch representations, the DSL machine achieves better results than the GP machine. This suggests that tailoring the instruction set to the nature of the output gives both higher mean and maximum results.

The diatonic pitch representation generally performs more poorly than the other two. This should not be caused by a general shift around the tonic, since in that case, the shifted chromatic representation should fall short as well. Instead the deficit may be due to a wider spectrum of pitches within the diatonic range. When using a chromatic scale, a delta of 12 notes represents an octave jump, while this value is 7 in the diatonic case. Since we represent pitch on 5 bits, 32 different values are defined—this covers 2.66 octaves in chromatic and 4.57 octaves in diatonic mode respectively. This

[7]Highest scoring models exported in MIDI format can be accessed at `https://emcrp.gitlab.io`

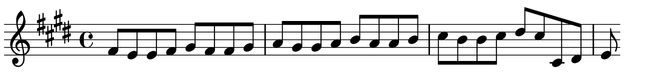


Figure 10: Portion of an example output showing variation on a small pattern. This model achieved 0.73 fitness.

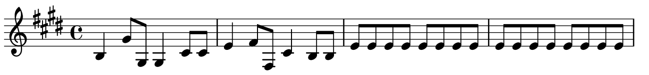


Figure 11: Portion of an example output showing a varied first part changing into a one-note loop. This model achieved 0.79 fitness.

widening of the search space in the diatonic representation potentially makes finding suitable individuals more difficult. Perhaps a constraint on diatonic pitch to values within an equivalent range would improve its performance.

The shifted chromatic representation outperforms the chromatic one only when using the DSL. This is understandable, since the DSL contains pitch manipulation 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. This explains the slight disadvantage of the chromatic representation when using the DSL.

Further analyzing the VM instruction sets, Figure 9 shows that the DSL machine outperforms the GP machine already in early generations. This demonstrates that the instruction set tailored for music increases the likelihood of even random genetic strings leading to pleasing outputs. Indeed, the mean fitness in the initial generation shows a doubling in favor of the DSL. Although it achieves higher scores, both VMs have quite high-scoring maximum individuals already in their early generations. It is likely that the informed reduction of the search space (only 8 and 32 possible values for duration and pitch, respectively) give an already favourable starting condition to the system.

In order to measure the significance of the improvements, one-tailed t-tests are run for the results shown in Figures 8 and 9. Significant improvements are obtained for all mean and maximum fitness values over the 20 runs (using significance level $\alpha = 0.05$), but we report here only the p-values (mean, maximum) obtained for the shifted chromatic representations: 4.07×10^{-12} and 1.01×10^{-17}. The improvements are also consistently significant, as the t-tests performed on the data from Figure 9 show a p-value of 0.

Exporting the entities as standard MIDI files allows subjective evaluation. This reveals that many pieces exhibit interesting repeating patterns of different lengths. For example, Figure 10 shows a portion of an output with a variation: four notes modulated up a step every iteration. This shows that longer patterns may emerge even if the maximum pattern length we investigate is shorter (in our case, trigrams).

However, many high scoring entities contain an almost random collection of notes in the beginning, later falling into a repetition of only 1 or 2 notes, or an ascending/descending scale (see Figure 11). The associated VM program likely enters a small infinite loop after outputting the first portion. This suggests the VM is able to find a "loophole" in the fitness metric: it is able to garner the expected medium entropy by having a highly random beginning and highly repetitive second half. This suggests more tests should be included which look at consistency in time rather than global metrics. Also, the instruction sets could be amended with instructions for structured loops.

8 CONCLUSION AND FUTURE WORK

In this paper we have addressed the issue of domain-specific knowledge in an evolutionary music composition algorithm. We have presented a linear genetic programming system capable of creating simple pieces of music similar to a corpus of folk songs.

We have 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 n-grams and Shannon entropy. Domain knowledge has been introduced to the system through a DSL instruction set tailored to western tonal music creation, as well as different pitch representations reliant on their knowledge of key.

We conclude that domain-specific knowledge beneficially impacts both the outcome and the rate of convergence of the genetic program. The instruction set plays an especially vital part in creating acceptable phenotypes in early generations and converging to higher fitness values in later ones. We also conclude that the shifted diatonic pitch representation only helps if the VM already understands the notion of key. The diatonic representation falls short of the other proposed schemes, suggesting further work investigating a reduced pitch range to bring its output into line with the other representations.

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. [36].

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. The quality of the evaluation process could also be measured through an online user survey—correlation between our grades and that of human evaluators would further prove the viability of the metric.

The deployed corpus consists of monophonic musical pieces, but the method could also be extended for polyphonic audio. We also propose experiments with other clustering algorithms, such as kernelized k-means, hierarchical or spectral clustering [12, 16], studying their influence on the generated musical pieces.

REFERENCES

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

[2] Jon Bentley. 1986. Programming pearls: little languages. *Commun. ACM* 29, 8 (1986), 711–721.

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

[4] Piero P. Bonissone, Raj Subbu, Neil Eklund, and Thomas R. Kiehl. 2006. Evolutionary algorithms + domain knowledge = real-world evolutionary computation. *IEEE Transactions on Evolutionary Computation* 10, 3 (2006), 256–280.

[5] Markus F. Brameier and Wolfgang Banzhaf. 2007. *Linear genetic programming* (1st ed.). Springer Science & Business Media.

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

[7] Ching-Hua Chuan and Elaine Chew. 2005. Polyphonic Audio Key Finding Using the Spiral Array CEG Algorithm. In *Proceedings of the 2005 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, Amsterdam, The Netherlands, 21–24.

[8] Darrell Conklin. 2003. Music generation from statistical models. In *Proceedings of the 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*. AISB, Aberystwyth, Wales, 30–35.

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

[10] Thomas M. Cover and Joy A. Thomas. 2012. *Elements of information theory*. John Wiley & Sons.

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

[12] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2004. Kernel k-means, spectral clustering and normalized cuts. In *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, Seattle, Washington, USA, 551–556.

[13] Brad Dolin, Maribel García Arenas, and Juan J. Merelo. 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)*. Springer-Verlag, London, UK, 142–152.

[14] Patrick Donnelly and John Sheppard. 2011. Evolving Four-Part Harmony Using Genetic Algorithms. In *Applications of Evolutionary Computation*. Vol. 6625. Springer, Berlin, Heidelberg, 273–282.

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

[16] Richard O. Duda, Peter E. Hart, and David G. Stork. 2000. *Pattern classification*. John Wiley & Sons.

[17] Arne Eigenfeldt. 2012. Corpus-based recombinant composition using a genetic algorithm. *Soft Computing* 16, 12 (2012), 2049–2056.

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

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

[20] Martin Fowler. 2010. *Domain Specific Languages* (1st ed.). Addison-Wesley Professional.

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

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

[23] David M. Hofmann. 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)*, Vol. 9027. Springer, Cham, Copenhagen, 89–100.

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

[25] Brad Johanson and Riccardo Poli. 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*. Morgan Kaufmann, University of Wisconsin, Madison, Wisconsin, USA, 181–186.

[26] John R. Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press.

[27] Carol L. Krumhansl. 2001. A key-finding algorithm based on tonal hierarchies. In *Cognitive Foundations of Musical Pitch*. Oxford University Press, 77–110.

[28] David Lidov and Jim Gabura. 1973. A Melody Writing Algorithm Using a Formal Language Model. *Computers in the Humanities* 3-4 (1973), 138–148.

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

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

[31] Róisín Loughran, James McDermott, and Michael O'Neill. 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)*. Springer, Porto, Portugal, 110–125.

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

[33] Bill Z. Manaris, Patrick Roos, Penousal Machado, Dwight Krehbiel, Luca Pellicoro, and Juan Romero. 2007. A Corpus-Based Hybrid Approach to Music Analysis and Composition. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 839–845.

[34] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge University Press.

[35] Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT press.

[36] Leonard C. Manzara, Ian H. Witten, and Mark James. 1992. On the Entropy of Music: An Experiment with Bach Chorale Melodies. *Leonardo Music Journal* 2, 1 (1992), 81.

[37] Nobuo Masataka. 2007. Music, evolution and language. *Developmental Science* 10, 1 (2007), 35–39.

[38] Jon McCormack. 1996. Grammar based music composition. *Complex systems* 96 (1996), 321–336.

[39] Ryan A. McIntyre. 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*. IEEE, Orlando, Florida, USA, 852–857.

[40] Robert I. McKay, Nguyen X. Hoai, Peter A. Whigham, Yin Shan, and Michael O'Neill. 2010. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines* 11, 3-4 (2010), 365–396.

[41] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.

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

[43] Eduardo Reck Miranda and John Al Biles. 2007. *Evolutionary Computer Music*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[44] Artemis Moroni, Jônatas Manzolli, Fernando Von Zuben, and Ricardo Gudwin. 2000. Vox populi: An interactive evolutionary system for algorithmic music composition. *Leonardo Music Journal* 10 (2000), 49–54.

[45] Peter Nordin, Wolfgang Banzhaf, and Frank D. Francone. 1999. Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. *Advances in genetic programming* 3 (1999), 275–299.

[46] Alfonso Ortega, Rafael Sánchez Alfonso, and Manuel Alfonseca. 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*. ACM, Madrid, Spain, 148–155.

[47] Marcus T. Pearce. 2005. *The construction and evaluation of statistical models of melodic structure in music perception and composition*. Ph.D. Dissertation. City University London.

[48] Somnuk Phon-Amnuaisuk, Andrew Tuson, and Geraint Wiggins. 1999. Evolving Musical Harmonisation. In *Proceedings of the 1999 International Conference on Artificial Neural Nets and Genetic Algorithms*. Springer, Portorož, Slovenia, 229–234.

[49] R. Plomp and J. M. Levelt. 1965. Tonal Consonance and Critical Bandwidth. *The Journal of the Acoustical Society of America* 38, 4 (10 1965), 548–560.

[50] Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. 2008. *A field guide to genetic programming*. Lulu. com.

[51] John Reddin, James McDermott, and Michael O'Neill. 2009. Elevated Pitch: Automated Grammatical Evolution of Short Compositions. In *Applications of Evolutionary Computing*. Lecture Notes in Computer Science, Vol. 5484. Springer Berlin Heidelberg, 579–584.

[52] Conor Ryan, John J. Collins, and Michael O'Neill. 1998. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Conference on Genetic Programming (EuroGP)*. Springer, Paris, France, 83–96.
[53] Eleanor Selfridge-Field. 2004. Music Theory for Computer Applications. (2004). http://www.ccarh.org/courses/254/MusicTheory_ComputerApps2004.htm
[54] Csaba Sulyok, Andrew McPherson, and Christopher Harte. 2019. Evolving the process of a virtual composer. *Natural Computing* 18, 1 (2019), 47–60.
[55] Joseph P. Swain. 1986. The need for limits in hierarchical theories of music. *Music Perception: An Interdisciplinary Journal* 4, 1 (1986), 121–147.
[56] David Temperley. 2001. *The cognition of basic musical structures.* MIT Press.
[57] Nao Tokui and Hitoshi Iba. 2000. Music composition with interactive evolutionary computation. In *Proceedings of the Third International Conference on Generative Art*, Vol. 17. Milan, Italy, 215–226.
[58] Esko Ukkonen, Kjell Lemström, and Veli Mäkinen. 2003. Geometric Algorithms for Transposition Invariant Content-Based Music Retrieval. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*. Johns Hopkins University Press, Maryland, USA, 193–199.
[59] Rodney Waschka II. 2007. Composing with Genetic Algorithms: GenDash. In *Evolutionary Computer Music*. Springer London, 117–136.
[60] Raymond Whorley and Darrell Conklin. 2016. Music Generation from Statistical Models of Harmony. *Journal of New Music Research* 45, 2 (2016), 160–183.
[61] Jackie Wiggins. 2007. Compositional process in music. In *International handbook of research in arts education*. Springer Netherlands, 453–476.
[62] Jacek Wolkowicz, Malcolm Heywood, and Vlado Keselj. 2009. Evolving indirectly represented melodies with corpus-based fitness evaluation. In *Workshops on Applications of Evolutionary Computation*. Springer Berlin Heidelberg, Tübingen, Germany, 603–608.
[63] Jacek Wolkowicz, Zbigniew Kulka, and Vlado Keselj. 2008. N-gram-based approach to composer recognition. *Archives of Acoustics* 33, 1 (2008), 43–55.
[64] Chia L. Wu, Chien H. Liu, and Chuan K. Ting. 2014. A novel genetic algorithm considering measures and phrases for generating melody. In *Proceedings of the 2014 Congress on Evolutionary Computation (CEC)*. IEEE, Beijing, China, 2101–2107.