Information Sciences 385–386 (2017) 196–212

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Splicing music composition

Clelia De Felice, Roberto De Prisco, Delfina Malandrino, Gianluca Zaccagnino, Rocco Zaccagnino*, Rosalba Zizza

Department of Computer Science, University of Salerno, Via Giovanni Paolo II, 132 I-84084 Fisciano (SA), Italy

ARTICLE INFO

Article history:
Received 15 January 2016
Revised 13 December 2016
Accepted 1 January 2017
Available online 3 January 2017

Keywords:
Splicing systems
Automatic music composition
Music formal model

ABSTRACT

Splicing systems were introduced by Tom Head (1987) as a formal model of a recombination process between DNA molecules. The existing literature on splicing systems mainly focuses on the computational power of these systems and on the properties of the generated languages; very few applications based on splicing systems have been introduced.

In this paper we show a novel application of splicing systems: we use them to build an automatic music composer. The proposed system can be seen also as a new valid bio-inspired strategy for automatic music composition. It is tailored on 4-voice chorale-like music. We define a new music representation based on words, which extends an earlier splicing approach and uses additional music information to produce music in a quick and effective way.

A performance study shows that our composer outperforms other meta-heuristics by producing better music according to a specific measure of quality evaluation. Moreover, the composition is carried out in a shorter time and using less memory with respect to a previous approach.

© 2017 Elsevier Inc. All rights reserved.

1. Introduction

Music is one of the arts that in many aspects have benefited from the use of computers: sound synthesis and design, digital signal processing, automatic composition, and so on. One of the most interesting challenges for the use of computers in the field of music, is the so-called *algorithmic music composition*, i.e., the problem of composing music by means of a computer program with no (or minimal) human intervention. In this work we face this problem.

The most meaningful examples of music composition systems start from the ILLIAC suite [24,25], and arrive to more recent efforts [7–9,17,18,35–37]. In [34], several techniques and tools to define algorithms for music composition have been discussed: evolutionary algorithms, bio-inspired algorithms, formal grammars, cellular automata, machine learning. In this work we investigate the use of a bio-inspired approach, specifically *splicing systems*, with the aim of defining efficient automatic music composers.

Splicing systems [19] represent a model of a recombination process between DNA molecules. Splicing occurs in two steps. In the first one, *restriction enzymes* (proteins) recognize a specific pattern in a DNA sequence and then cut the molecule in a specific point inside the recognized pattern. The shape of these cut points are specific to each enzyme. Since DNA is double stranded, the cutting phase may produce blunt or staggered ends. This is caused by cleavage patterns which are blunt or not.

* Corresponding author.
E-mail address: zaccagnino@dia.unisa.it (R. Zaccagnino).

http://dx.doi.org/10.1016/j.ins.2017.01.004
0020-0255/© 2017 Elsevier Inc. All rights reserved.

If no sticky ends match, no fragment previously cut may be pasted. Clearly, if such staggered ends are close to each other, the two original molecules may be re-associated, but this is not of particular interest. Nevertheless, it is also possible that two new hybrid molecules will be formed. In the second step, *ligase enzymes* paste together properly matched fragments, under chemical conditions (related to cleavage patterns structure and complementary laws).

This mechanism can be viewed as a formal language operation on words, by abstracting double strands into single strands: starting from two words, the splicing operation generates new words by concatenating a prefix of one string with a suffix of the other string. The concatenation is performed under some conditions (abstraction of the enzymes' behavior), represented as a (splicing) rule. As a consequence we can consider a system based on this operation. Given a finite alphabet $\mathcal{A}$, a splicing system $\mathcal{S}$ is specified by an initial language $\mathcal{I}$ (a set of words) and by a set $\mathcal{R}$ of splicing rules. Starting from $\mathcal{I}$, and applying the rules in $\mathcal{R}$ to the pairs of words previously generated, $\mathcal{S}$ generates a language $L(\mathcal{S})$, called the *splicing language generated* by $\mathcal{S}$. A natural research area is the investigation of the computational power of such formal systems, which mainly depends on the level of the Chomsky hierarchy that $\mathcal{I}$ and $\mathcal{R}$ belong to. For instance, all finite languages can be generated when $\mathcal{I}$ and $\mathcal{R}$ are both finite sets (finite splicing systems). The class of finite splicing systems can generate only a proper subclass of regular languages [21,48]. Splicing systems theory is still a challenging field of theoretical research, as shown by recent literature [5,6,20,30].

1.1. Contribution of this work

In this paper we (1) show a novel application of splicing system; in this regard we are extending our earlier work [10] in order to make improvements in terms of both efficiency and efficacy, and (2) we propose the music splicing approach as a new valid bio-inspired approach for automatic music composition, in alternative to other widely studied automatic composers based on meta-heuristics strategies [33]. Meta-heuristics are intelligent strategies used to design and improve very general heuristic procedures with a high performance, and have been already used for several other problems. It is worth to note that, the genetic algorithms represent the most largely used meta-heuristic strategy for automatic music composition [3,4,12,14,27–29,47].

We focus our attention on 4-voice chorale-like music and we build a system based on a splicing system, defined as follows: an initial set of Bach's chorales (initial set of words) and a set of well-established rules in classical music composition (set of splicing rules) obtained by extracting information about the notes, chords, and tonalities. The generated language contains words that represent pieces of "new" chorales in Bach's style. Similarly as in [10], our basic idea is to treat music compositions as words and to view the music compositional process as the result of operations on words. The representation proposed in [10], that in this work we name *Note representation*, only takes into account information about each single note. Our idea in this work is to extend it with a new representation, named *Tonality-degree representation*, by considering additional music information, i.e., the tonality and the degree of each single note.

The Tonality-degree representation enables to model the music composition rules (modulations, chords passages, and so on) in a more natural and effective way with respect to composers based on well-known meta-heuristics strategies, heavily used for music composition. The composition process emulated by composers based on such a representation, is closer to the human attitude, in terms of application of music rules. This holds, in particular, when reproducing the style of a specific composer, as confirmed by our experimental results.

We show that the use of the Tonality-degree representation can improve splicing systems based on the Note representation in terms of *music quality* of the produced compositions and *performance* (execution time and memory consumption). As aforementioned, there exists a broad variety of tools and techniques that can be used to design an algorithmic composer. In this paper we compare the music produced by the composer based on the Tonality-degree representation with the earlier Note representation and with music produced by several well-known meta-heuristics strategies. Results of our comparisons are discussed in detail in Section 7. An example of a comparison of a different strategy with the same meta-heuristics can be found in [1].

In general, to evaluate the produced music, automatic composition systems implement an objective function, according to either *existing rules from music theory* [2,16,22] or by *learning from a corpus of existing music* [39]. Although every musical genre has its own rules, these are usually not well-defined, and this represents a barrier to the applicability of the former approach [38]. The latter strategy can be used to overcome this problem, by defining style rules that can be learned automatically from existing music.

In this work we propose a *hybrid* approach: we define an evaluation function that, on one hand, adheres to the classical music rules, and on the other hand, extracts statistical information from a corpus of existing music, to assimilate a specific composers style. Specifically, we extracted statistical information from a large corpus of J.S. Bach's chorales, which represents a formidable example of chorale music.

The proposed composer, implemented in Java, is totally automatic since it does not require any human intervention. Finally, some chorales generated by this composer can be found online,[1] in MIDI format.

The rest of the paper is organized as follows. In Section 2 we discuss some relevant works in this field. In Section 3 we introduce important notions used through the paper. In Section 4 we briefly describe the Note representation while in

[1] http://www.di.unisa.it/~delmal/research/usability/Splicing.

Table 1
Notations used through the paper.

Notation	Description
Splicing systems	
$\mathcal{A}$	alphabet
$\mathcal{I}$	initial set of words
$\mathcal{R}$	set of splicing rules
$\mathcal{S}$	splicing system $\mathcal{S} = (\mathcal{A}, \mathcal{I}, \mathcal{R})$
ϵ	empty word
w	a word
|	special separator symbol ($\notin \mathcal{A}$)
\$	special separator symbol ($\notin \mathcal{A}$)
r	a splicing rule
site	a site of a splicing rule
$\vdash_r$	application of the rule r
$\gamma'(L)$	language obtained by applying all rules of $\mathcal{R}$ on words of L
Music splicing systems	
$\mathcal{A}_V$	voices alphabet
$\mathcal{A}_N$	notes alphabet
$\mathcal{A}_O$	octaves alphabet
$\mathcal{A}_T$	tonalities alphabet
$\mathcal{A}_Q$	qualities alphabet
$\mathcal{A}_{\text{NMSS}}$	note representation alphabet ($\mathcal{A}_V \cup \mathcal{A}_N \cup \mathcal{A}_O$)
$\mathcal{A}_{\text{TDMSS}}$	tonality-degree representation alphabet ($\mathcal{A}_V \cup \mathcal{A}_N \cup \mathcal{A}_O \cup \mathcal{A}_T \cup \mathcal{A}_Q$)
$\mathcal{I}_{\text{NMSS}}$	initial set of words for Note representation
$\mathcal{R}_{\text{NMSS}}$	set of splicing rules for Note representation
$\mathcal{S}_{\text{NMSS}}$	note music splicing system $\mathcal{S}_{\text{NMSS}} = (\mathcal{A}_{\text{NMSS}}, \mathcal{I}_{\text{NMSS}}, \mathcal{R}_{\text{NMSS}})$
$\mathcal{I}_{\text{TDMSS}}$	initial set of words for Tonality-degree representation
$\mathcal{R}_{\text{TDMSS}}$	set of splicing rules for Tonality-degree representation
$\mathcal{S}_{\text{TDMSS}}$	tonality-degree music splicing system $\mathcal{S}_{\text{TDMSS}} = (\mathcal{A}_{\text{TDMSS}}, \mathcal{I}_{\text{TDMSS}}, \mathcal{R}_{\text{TDMSS}})$
$\rightarrow$	chord passage
f_h	harmonic objective function
f_m	melodic objective function

Section 5 we introduce its enhancement, represented by the Tonality-degree representation. In Section 6 we provide details about the implementation of our system. A comparison between our approach with similar strategies is presented in Section 7. Finally, in Section 8 we conclude with some final remarks and future directions.

2. Related work

Several automatic composers of chorale music, based on different approaches, have been already proposed in previous works: rules and expert-systems [15,45], systems based on a combination of formal grammars, analysis and pattern matching techniques [7,9], cellular automata [40], neural networks [31]. In [23] it is built a system that generates bagana music, a traditional lyre from Ethiopia, based on a first order Markov model. The paper proposes a method that allows the enforcement of structure and repetition within music, thus handling long term coherence with a first order model. Also, it proposes different ways in which low order Markov models can be used to build quality assessment metrics for an optimization algorithm. Examples of automatic composers that use a Markov model for 4-voice harmonizations can be found in [46]. In a different work [39] the authors trained hidden Markov models for harmonizing Bach chorales.

Several automatic composers are based on meta-heuristics, specifically on genetic algorithms [3,4,12–14,27–29,47]. The problem of composing 4-voice music has been addressed also in [32,43,47]. Other works have been proposed for different musical genres or problems: thematic bridging [28], Jazz solos [3], interactive genetic algorithm [29], harmonization of chords progressions [27], monophonic jazz composition given a chord progression [4], harmonization of a *figured* bass [14], harmonization of an *unfigured* bass [12] and bass *functional* harmonization [11].

3. Background

We assume that the reader is familiar with the basics of music and formal languages, more specifically splicing systems. However, in this section we provide the basic notations used throughout the paper. The notation is summarized in Table 1.

3.1. Music notions

In this section we provide, briefly and in a simplified way, the basic music notions needed to understand the rest of the paper.

Our work is based on the tempered music system, conventionally used among western musicians. This system models the musical notes by using a reference sound (a given frequency, normally 440 Hz) and defining octaves as the ranges of

frequencies between those sounds obtained by doubling/halving the reference sound. To explain the organization of the musical notes we will use the piano keyboard as reference: each key of the piano corresponds to a specific pitch (note) and there are 88 notes, divided into *octaves*. The piano keyboard has roughly 7 octaves, usually numbered from 1 to 7, which contain the 88 keys ($12 * 7 = 84$ keys in 7 octaves plus 4 extra keys); notes outside this range are hardly used because their frequencies are too low or too high to be pleasantly perceived by our ear. Each octave is divided in 12 equally spaced notes, which constitute the chromatic scale. These notes are denoted by the letters *A*, *A#* or *B♭*, *B C*, *C#* or *D♭*, *D*, *D#* or *E♭*, *E*, *F*, *F#* or *G♭*, *G*, *G#* or *A♭*.

This system is based on *tonalities*. Western music considers two kinds of tonalities: *major and minor*. For each note we can build a major tonality and a minor tonality, for a total of 12 major tonalities and 12 minor tonalities. A tonality is associated to a sequence of notes, called *scale*. A scale is an ordered set of musical notes included in the range of an octave. For example, the scale of the C major tonality is *C, D, E, F, G, A, B, C*, while the scale of the tonality D Major is *D, E, F♯, G, A, B, C♯, D*. Given a tonality and the corresponding scale, the notes that do not belong to such a scale are called *non-harmonic tones*. Non-harmonic tones can be classified in: *passing tones, auxiliary tones, appoggiatura tones* and *suspension tones* (see, for example, [41] for more details). Each music composition has its own *main tonality*.

Music comes in a huge variety of forms. In this work we focus on a specific form: 4-voice chorale music. In a chorale there are 4 voices called *bass, tenor, alto* and *soprano*. Each voice has an admissible range of notes, and overall the 4 voices cover from octave number 2 (partially) to octave number 6, with reference to the organization of the octaves of the piano. Formally, a chorale is a sequence of *measures*. In each measure there are *beats* whose number may vary, but usually is fixed for the entire composition according to a fixed *meter* of the composition. Although the following is a simplification of the reality, we assume that in each beat, each voice plays a single note (at the end of Section 6 we will describe how to overcome this simplification and add extra notes). The 4 notes played by the 4 voices in a given beat create a *chord*. Chords are built on the degrees of the scale of the tonality used. To abstract from the specific tonality used, the degree of the scale is indicated with roman numerals: *I, II, III, IV, V, VI, VII*. Commonly, major chords are indicated with capital letters and minor chords are indicated with small letters. As an example, given a scale, if the chord on the first degree is major then we use the notation *I*, on the contrary, if the chord on the first degree is minor then we use the notation *i* (see [41] for details). We remark that the chord built on the seventh degree of a major or minor scale, and on the second degree of a minor scale, is a special minor chord, which is named diminished chord and is denoted with *vii*°.

The sequence of chords used to compose music is obviously decided by the composer. However there are rules established by the theory of harmony. According to these rules, there are some chords combinations that work better than others, and there are specific chord sequences (e.g., cadences) that have specific musical functions. For our system, we consider the following musical cadences: $V \to I$, $II \to V$, $VI \to II$, $V \to VI$, $IV \to V$,$IV \to I$ and $III \to VI$. For each cadence we will define several splicing rules, detailed in Sections 4 and 5. Beside rules about chord sequences, there are also rules about melodic lines.

Classical music theory established strict rules about the movement of melodic lines. These rules concern both the movement of a single line (jump) and the relationship between the movements of two different lines. Such rules are based on *intervals*, i.e., the distance between two notes of two different melodic lines. The interval between two notes is the number of halftones between the notes. For example, between C3 and G3 there is a fifth interval (7 halftones), between C3 and C4 there is an octave interval (12 halftones) and between C3 and C3 an unison interval (0 halftones).

A single line should move by using the notes of the tonality of the composition, and each jump from note to note creates an interval. Since there are intervals preferred to others, there are jumps preferred to others. As an example, jumps greater than one octave are less preferable than to those within an octave. Hence, since the movements of a single line are quite straightforward to check, we focus our attention on the rules about the relationship between the movements of two different lines. According to what dictated by music theory rules, for any given pair of lines, some specific patterns should be avoided: (1) two lines that move by creating two consecutive unisons; (2) two lines that create two consecutive octaves or fifths; (3) two melodic lines that intersect.

3.2. Splicing systems

In his seminal work [19], Head described the biochemical phenomenon of splicing as an operation on words. Subsequently, some variants of this operation were introduced [42,44]. Here we will use the splicing operation as introduced by Păun and described below.

Let $\mathcal{A}$ be a finite alphabet. A word (or string) over $\mathcal{A}$ is a finite sequence of symbols from $\mathcal{A}$. We denote by $\mathcal{A}^*$ the set of words over $\mathcal{A}$ and by ϵ the empty word, i.e., the word containing no symbols.

A rule r is a word of the form $u_1|u_2\ \$\ u_3|u_4$, where u_1, u_2, u_3, u_4 are words over $\mathcal{A}$ and |,$ are two symbols which are not in $\mathcal{A}$. The rule specifies two positions where we cut two input strings and how we can glue these fragments.

More formally, let x and y be two words and let $r = u_1|u_2\ \$\ u_3|u_4$ be a rule. If x contains u_1u_2, i.e., $x = x_1u_1u_2x_2$, and y contains u_3u_4, i.e. $y = y_1u_3u_4y_2$, the rule r applies to x and y and produces the words $w' = x_1u_1u_4y_2$ and $w'' = y_1u_3u_2x_2$ as a result. This operation is often denoted by $(x, y) \vdash_r (w', w'')$. The words u_1u_2 and u_3u_4 are called *sites* of r. Clearly, if a site occurs more than once, the splicing operation is applied for each occurrence of it. The words x_1, x_2, y_1, y_2 can be the empty word.

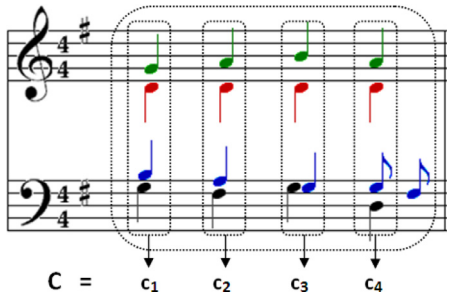


Fig. 1. Note representation. Fragment of the Bach's Chorale BWV 32.6.

Example 1. Let $r = a|b\ \$\ c|d$ and consider $x = fabf$ and $y = ggcdh$. Then *fadh* and *ggcbf* are generated by splicing. If we consider $x = fabfgabf$ and $y = ggcdh$, then *fadh, ggcbfgabf, fabfgadh* and *ggcbf* are generated.

As mentioned in Section 1, splicing systems are formal models for generating languages, i.e., sets of words. We start with an initial set of words and we apply to these words the splicing operation by using rules in a given set. The set of generated words is joined to the initial set and the process is iterated on this new set until no new word is produced. The language generated by a splicing system is the collection of all these words.

Formally, a *splicing system* is a triple $\mathcal{S} = (\mathcal{A}, \mathcal{I}, \mathcal{R})$, where $\mathcal{A}$ is a finite alphabet, $\mathcal{I} \subseteq \mathcal{A}^*$ is the initial language and $\mathcal{R} \subseteq \mathcal{A}^*|\mathcal{A}^*\ \$\ \mathcal{A}^*|\mathcal{A}^*$ is the set of rules, where $|, \$ \notin \mathcal{A}$. A splicing system $\mathcal{S}$ is finite when $\mathcal{I}$ and $\mathcal{R}$ are both finite sets. Let $L \subseteq \mathcal{A}^*$. We set $\gamma'(L) = \{w', w'' \in \mathcal{A}^* \mid (x, y) \vdash_r (w', w''),\ x, y \in L, r \in \mathcal{R}\}$. The definition of the splicing operation is extended to languages as follows:

$$\begin{aligned} \gamma^0(L) &= L, \\ \gamma^{i+1}(L) &= \gamma^i(L) \cup \gamma'(\gamma^i(L)),\ i \geq 0, \end{aligned}$$

$$\gamma^*(L) = \bigcup_{i \geq 0} \gamma^i(L). \tag{1}$$

Definition 1. Let $\mathcal{S} = (\mathcal{A}, \mathcal{I}, \mathcal{R})$ be a splicing system. We denote by $L(\mathcal{S}) = \gamma^*(\mathcal{I})$ the *splicing language generated* by $\mathcal{S}$. We say that L is *a splicing language* if there exists a splicing system $\mathcal{S}$ such that $L = L(\mathcal{S})$.

The interested reader can refer to [26] for a detailed study of the theory of formal languages, and to [19,44,48] for Supplementary material on the theory of splicing systems.

4. Note representation

In this section we briefly describe the representation defined in [10], called here *Note representation*. The enhancements made on this representation will be discussed in the next section.

We define a music splicing system using the Note representation as a *Note Music Splicing System* $\mathcal{S}_{\text{NMSS}}$. Such a system is defined by an initial ground data set of Bach's chorales, represented as words (initial set of words $\mathcal{I}_{\text{NMSS}}$ on the alphabet $\mathcal{A}_{\text{NMSS}}$), and a set of well-established rules in classical music composition (set of splicing rules $\mathcal{R}_{\text{NMSS}}$) obtained by extracting information about the notes. The language generated contains words that represent pieces of "new" chorales in the Bach's style. Formally, a Note Music Splicing System is a triple $\mathcal{S}_{\text{NMSS}} = (\mathcal{A}_{\text{NMSS}}, \mathcal{I}_{\text{NMSS}}, \mathcal{R}_{\text{NMSS}})$. Among the generated words we choose the best solution according to the evaluation function that we will define in Section 6.1.

The alphabet $\mathcal{A}_{\text{NMSS}}$ is used to specify the notes. We set $\mathcal{A}_{\text{NMSS}} = \mathcal{A}_V \cup \mathcal{A}_N \cup \mathcal{A}_O$, where $\mathcal{A}_V = \{\beta, \tau, \alpha, \sigma\}$ is the *voices alphabet* (β, τ, α, σ refer to bass, tenor, alto, and soprano); $\mathcal{A}_N = \{C, C\#, D\flat, D, D\#, E\flat, E, F, F\#, G\flat, G, G\#, A\flat, A, A\#, B\flat, B\}$ is the *notes alphabet* and $\mathcal{A}_O = \{2, 3, 4, 5, 6\}$ is the *octaves alphabet*. Using $\mathcal{A}_{\text{NMSS}}$ we can represent 4-voice chorale-like music as words. Given a 4-voice composition $C = (c_1, \ldots, c_n)$, each c_i is a chord which can be represented as a word w_i over $\mathcal{A}_{\text{NMSS}}$, for each $1 \leq i \leq n$. Specifically $w_i = \beta x_i \tau y_i \alpha v_i \sigma z_i$, where $x_i, y_i, v_i, z_i \in \mathcal{A}_N \mathcal{A}_O$, for each $1 \leq i \leq n$. So, the composition $C = (c_1, \ldots, c_n)$ is represented as $w(C) = w_1, w_2, \ldots, w_n$.

Example 2. Let us consider the chorale fragment C (from Bach's Chorale BWV 32.6) shown in Fig. 1. The fragment consists of 4 chords: $C = (c_1, c_2, c_3, c_4)$. We have: $w_1 = \beta G3\tau B3\alpha D4\sigma G4$, $w_2 = \beta F\#3\tau A3\alpha D3\sigma A4$, $w_3 = \beta G3\tau G3\alpha D4\sigma B4$, $w_4 = \beta D3\tau G3\alpha D4\sigma A4$, therefore:

$$w = \beta G3\tau B3\alpha D4\sigma G4\beta F\#3\tau A3\alpha D4\sigma A4\beta G3\tau G3\alpha D4\sigma B4\beta D3\tau G3\alpha D4\sigma A4.$$

To define $\mathcal{I}_{\text{NMSS}}$, we consider the *ground set* $\mathcal{G}$ containing 10 Bach's chorales[2]. Each one of these 10 chorales was transposed in every tonality. For each obtained chorale $C = (c_1, \ldots, c_n)$, given the corresponding word $W(C) = w_1, \ldots, w_n$, we inserted w_i in $\mathcal{I}_{\text{NMSS}}$, for each $1 \leq i \leq n$.

The aim is to define rules that generate good music patterns. So the chorales in $\mathcal{G}$ have been first analyzed, and then single chords have been extracted so that we can generate music that is closer to that in $\mathcal{G}$. During the chord extraction, we also attached information about the degree of the scale on which the chord is built to the extracted chord. This information will be crucial in re-arranging the chords, by means of splicing rules, so that specific sequences of chords (e.g., cadences) will be produced. As done for the initial set, we have transposed each extracted chord in all 12 tonalities. We call Chords($\mathcal{G}$) the set of these chords. For each extracted chord $c \in \text{Chords}(\mathcal{G})$, we denote with Degree($c$) the degree of c. The set of words associated with Chords($\mathcal{G}$) is $\mathcal{W}(\text{Chords}(\mathcal{G}))$.

We decided to define the splicing rules exploiting the most used musical cadences; additionally, we also impose that a composition starts and ends with a chord built on the I degree of the scale, since this is what normally happens. Thus we can partition all the rules in three groups:

Group 1 (forcing I *to start).* For each $c_1, c_4 \in \text{Chords}(\mathcal{G}))$, such that $\text{Degree}(c_1) = I$, we define a rule $r = w_1|\epsilon \ \$ \ \epsilon|w_4$, where w_1 is the word associated to c_1 and w_4 is the word associated to c_4. Moreover we also insert w_1 in $\mathcal{I}_{\text{NMSS}}$.

Group 2 (forcing cadences). For each $c_1, c_2, c_3, c_4 \in \text{Chords}(\mathcal{G})$, such that $\text{Degree}(c_1) \rightarrow \text{Degree}(c_4)$ and $\text{Degree}(c_3) \rightarrow \text{Degree}(c_2)$ are cadences, we define $r = w_1|w_2 \ \$ \ w_3|w_4$ where w_i is the word associated to c_i, for $i = 1, 2, 3, 4$.

Group 3 (forcing I *as ending).* For each chords $c_1, c_4 \in \text{Chords}(\mathcal{G})$, such that $\text{Degree}(c_4) = I$, we define $r = w_1|\epsilon \ \$ \ \epsilon|w_4$ where w_1 is the word associated to c_1 and w_4 is the word associated to c_4. The word w_4 is also inserted into $\mathcal{I}_{\text{NMSS}}$.

Using the 10 chorales in $\mathcal{G}$ we have obtained 2160 chords. The above set of rules, applied to the ground set $\mathcal{G}$, yields a total of 1,658,880 rules in $\mathcal{R}_{\text{NMSS}}$.

4.1. Limitations

The Note representation suffers from some limitations; on one hand, the quality of the music is lower compared to the Tonality-degree representation because the former uses less information about the music; moreover it is also not efficient with respect to both space and time complexity. Indeed an empirical evaluation has shown that the system based on the Note representation is quite slow and uses a lot of memory. From an analytical point of view, this can be justified by the fact that in such a representation, there is no information about the tonality and degree of the chords. This means that, in order to define the rules, we have first to transpose all the chords extracted in all the tonalities, and then we have to build the rules by considering all possible combinations. This involves for a lot of memory usage and consequently time consumption. Formally, given an initial set of chorales $\mathcal{G}$, the time and space complexities of the Note music splicing system are quadratic in $|\text{Chords}(\mathcal{G})|$, i.e., $\mathcal{O}(|\text{Chords}(\mathcal{G})|^2)$.

In Section 7, we report the results of some experiments in which execution time and memory consumption, for very large initial sets and executions with many iterations, can significantly increase.

5. Tonality-degree representation

In order to overcome the limitations described in Section 4.1, we introduce the *Tonality-degree representation* and then we build a *Tonality-degree music splicing system* ($\mathcal{S}_{\text{TDMSS}}$ for short). The basic idea is to increase the musical information in the Note representation by also considering music degree information for each beat of the composition. The use of more information allows the automatic composer to produce better quality music using less time and space.

Similar to what defined in Section 4, we build an initial set of words starting from a ground data set of Bach's chorales. Then, we apply the splicing rules on the set of initial words repeating it many times; among the generated words we choose the best solution according to an evaluation function that we will define in Section 6.1.

The splicing system $\mathcal{S}_{\text{TDMSS}}$ consists of three components: $\mathcal{A}_{\text{TDMSS}}$ is the alphabet of the symbols, $\mathcal{I}_{\text{TDMSS}}$ is the initial of words and $\mathcal{R}_{\text{TDMSS}}$ is the set of rules. In the following we will describe such components in order to define the system $\mathcal{S}_{\text{TDMSS}} = (\mathcal{A}_{\text{TDMSS}}, \mathcal{I}_{\text{TDMSS}}, \mathcal{R}_{\text{TDMSS}})$.

5.1. The alphabet $\mathcal{A}_{\text{TDMSS}}$

For the Tonality-degree representation we use the voice alphabet $\mathcal{A}_{\text{TDMSS}}$. We set $\mathcal{A}_{\text{TDMSS}} = \mathcal{A}_V \cup \mathcal{A}_N \cup \mathcal{A}_O \cup \mathcal{A}_T \cup \mathcal{A}_Q$ where $\mathcal{A}_V$ is the *voices alphabet*, $\mathcal{A}_N$ is the *notes alphabet* and $\mathcal{A}_O$ is the octaves alphabet, as defined in Section 4. Additionally, we introduce the *tonalities alphabet*$\mathcal{A}_T = \{C, C\#, D\flat, D, D\#, E\flat, E, F, F\#, G\flat, G, G\#, A\flat, A, A\#, B\flat, B\}$, the *qualities alphabet* $\mathcal{A}_Q = \{M, m\}$ (where M stands for major tonality and m for minor tonality), and the degree alphabet $\mathcal{A}_D = \{1, 2, 3, 4, 5, 6, 7\}$.

[2] The 10 chorales in the ground set $\mathcal{G}$ are BWV 3.6, BWV 10.7, BWV 11.6, BWV 12.7, BWV 13.6, BWV 14.5, BWV 20.7, BWV 20.11, BWV 31.9 and BWV 32.6.

As explained in Section 4 an entire composition $C = (c_1, \ldots, c_n)$ is represented as $w(C) = w_1, w_2, \ldots, w_n$. Given a set of chorale-like compositions $\mathcal{C} = \{C_1, \ldots, C_k\}$, the set of resulting words is $\mathcal{W}(\mathcal{C}) = \{w(C_1), \ldots, w(C_k)\}$. We will also refer to $\mathcal{C}$ as the set of 4-voice compositions associated to $\mathcal{W}(\mathcal{C})$.

Example 3. Let us consider again the music fragment C (from the Bach's Chorale BWV 32.6) shown in Fig. 1. To use the Tonality-degree representation, we need to observe more information about the composition. In particular, the fragment is in G major tonality, that is denoted with **GM**. There are 4 chords and the sequence of degrees is $I - V - I - V$. The I degree is denoted with **1** and the V degree is denoted with **5**. Such information are encapsulated in the new representation as follows:

$w_1 =$ **GM1** $\beta G3\tau B3\alpha D4\sigma G4$ **GM1**, $w_2 =$ **GM5**$\beta F\#3\tau A3\alpha D4\sigma A4$ **GM5**,
$w_3 =$ **GM1** $\beta G3\tau G3\alpha D4\sigma B4$ **GM1**, $w_4 =$ **GM5**$\beta D3\tau G3\alpha D4\sigma A4$**GM5**, so $w = w_1w_2w_3w_4$.

5.2. Initial set and rules definition

The initial set was described in Section 4. We want to emphasize that the definition of the rules is crucial because they determine the language being generated. Similar to the Note representation, we start from the ground data set $\mathcal{G}$ obtained by analyzing a set of Bach's chorales, and we model the set of splicing rules by using the theory of music harmony. However, in the Note representation system we built the rules by considering all the combinations of the extracted chords. With the Tonality-degree representation system, instead, we directly integrate the degree and the tonality of the chord in the word representation of each chord. The advantage is that we do not need to transpose each chord in all 12 tonalities since to define a rule we only need to use extracted chords having specific tonalities and degrees. This guarantees a considerably smaller number of rules. In addition, with the Tonality-degree representation we directly extract the rules for the modulations.

As a consequence, given an initial set of chorales $\mathcal{G}$, and the set $\text{Chords}(\mathcal{G})$ of chords extracted by $\mathcal{G}$, the time and space complexities of the Tonality-degree music splicing system are linear in $|\text{Chords}(\mathcal{G})|$, i.e., $\mathcal{O}(|\text{Chords}(\mathcal{G})|)$.

As proposed in [10] we define splicing rules on the basis of the classical harmonic rules (see Section 3.1). Furthermore, we impose that a composition starts and ends with a I degree of the scale, since this is what usually happens in chorales music. Hence, we partition all the rules in four sets:

Group 1 (forcing I *to start).* For $c_1, c_4 \in \text{Chords}(\mathcal{G})$, satisfying $\text{Degree}(c_1) = I$ and $\text{Tonality}(c_1) = \text{Tonality}(c_4)$, we construct the rule $r = w_1|\epsilon\ \$\ \epsilon|w_4$ where, w_1 and w_4 are the word associated to c_1 and c_4, respectively. We also add w_1 to $\mathcal{I}_{\text{TDMSS}}$.

Group 2 (forcing cadences). For $c_1, c_2, c_3, c_4 \in \text{Chords}(\mathcal{G})$, if $\text{Degree}(c_1) \to \text{Degree}(c_4)$, with $\text{Tonality}(c_1) = \text{Tonality}(c_4)$ is a cadence, and $\text{Degree}(c_3) \to \text{Degree}(c_2)$, $\text{Tonality}(c_3) = \text{Tonality}(c_2)$, is also a cadence, we define $r = w_1|w_2\ \$\ w_3|w_4$ where w_i is the word associated to c_i, for $i = 1, 2, 3, 4$.

Group 3 (forcing I *as ending).* For $c_1, c_4 \in \text{Chords}(\mathcal{G})$, satisfying $\text{Degree}(c_4) = I$ and $\text{Tonality}(c_1) = \text{Tonality}(c_4)$, we define $r = w_1|\epsilon\ \$\ \epsilon|w_4$ where w_1 and w_4 are the words associated to c_1 and c_4, respectively. The word w_4 is also inserted into $\mathcal{I}_{\text{TDMSS}}$.

Group 4 (forcing modulations). Let $c_i, c_{i+1} \in \text{Chords}(\mathcal{G})$ be consecutive chords such that $\text{Tonality}(c_i) \neq \text{Tonality}(c_{i+1})$. Let $c_j, c_{j+1} \in \text{Chords}(\mathcal{G})$ be two other consecutive chords, such that $\text{Tonality}(c_j) \neq \text{Tonality}(c_{j+1})$. Then we define $r = w_i|w_{i+1}\ \$\ w_j|w_{j+1}$ where w_k is the word associated with c_k, for $k = i, i+1, j, j+1$.

The ground set $\mathcal{G}$, containing 10 Bach's chorales, produced 138 chords. The above set of rules, applied to these 138 chords yields a total of 8,598 rules in $\mathcal{R}_{\text{NMSS}}$. We want to emphasize that both the number of chords and the number of rules are significantly reduced compared with those of the Note representation.

6. Implementation details

We now give details about the implementation of the automatic composer based on the Tonality-degree representation.

In general, given a splicing system $\mathcal{S} = (\mathcal{A}, \mathcal{I}, \mathcal{R})$, the corresponding generated language $L(\mathcal{S})$ is an infinite set of words. Moreover, we recall that $L(\mathcal{S}) = \gamma^*(\mathcal{I}) = \bigcup_{i \geq 0} \gamma^i(\mathcal{I})$ (Definition 1 and Eq. (1)). In this equation the number i of iterations of the splicing operation is unbounded.

Of course, for practical reasons, when we consider the splicing language $L(\mathcal{S}_{\text{TDMSS}})$ generated by the music splicing system $\mathcal{S}_{\text{TDMSS}}$ we need to fix bounds for both these parameters (cardinality of the set and number of iterations). Thus, we fix a number k of iterations and a maximal cardinality p_{max}. We also define k languages as follows. We set $L_0 = \mathcal{I}_{\text{TDMSS}} = \gamma^0(\mathcal{I}_{\text{TDMSS}})$. For any i, $1 \leq i \leq k$, we consider $L'_i = L_{i-1} \cup \gamma'(L_{i-1})$, which corresponds to enlarge L_{i-1} by an application of all the rules in $\mathcal{R}_{\text{TDMSS}}$ to all possible pairs of words in L_{i-1}. If $\text{Card}(L'_i) \leq p_{max}$, then $L_i = L'_i$. Otherwise, L_i is obtained from L'_i by erasing the $\text{Card}(L'_i) - p_{max}$ words in L'_i that are the worst with respect to two functions, namely the *harmonic function*, i.e., the quality of the sequence of chords in the chorale, and the *melodic function*, i.e., the appropriate placement of the notes within each chord. Therefore, to measure the quality of the compositions and choose the better solutions we consider

Table 2
Typical harmonic chord passages in a major tonality (see [41]). The symbol - means that for a given degree there is no chord in the corresponding class.

Major	Cadence			
Degree	*Often*	*Sometimes*	*Seldom*	*Never*
I →	*I, IV, V*	*vi*	*ii, iii*	*vii°*
ii →	*ii, V*	*IV, vi*	*I, iii*	*vii°*
iii →	*iii, vi*	*IV*	*I, ii, V*	*vii°*
IV →	*IV, V*	*I, ii*	*iii, vi*	*vii°*
V →	*I, V*	*IV, vi*	*ii, iii*	*vii°*
vi →	*ii, V, vi*	*iii, IV*	*I*	*vii°*
vii° →	*I, iii, vii°*	*vi*	*ii, IV, V*	–

a multi-objective function. Notice that since we use a multi-objective function, the best solutions are given by those in the Pareto front, that represents the set of all non-dominated solutions. Finally, we define $L(k, p_{max}) = \cup_{1 \leq i \leq k} L_i$ as the (k, p_{max})-language generated by $\mathcal{S}_{\text{TDMSS}}$. We remark that $L(k, p_{max})$ is the language considered during the experiments described in Section 7.

In the following section we describe the multi-objective function.

6.1. The multi-objective evaluation function

As said in Section 1, in this work we define an evaluation function that, on one hand reflects the classical music rules, and on the other hand extracts statistical information from a corpus of existing music. Specifically we extracted statistical information from a set of Bach's chorales.

The multi-objective evaluation function is composed of two objective functions: an harmonic function f_h and a melodic function f_m. Both functions have the following general form:

$$f = \sum_i a_i w_i \tag{2}$$

where a_i are *coefficients* and w_i are *weights*.

The weights w_i are used to express the objective part of the evaluation while the coefficients are used to express a subjective component. The coefficients a_i are normally obtained with a statistical analysis of Bach's chorales.

6.1.1. The harmonic function

The harmonic function $f_h(C)$, is defined as follows:

$$f_h(C) = \sum_{i=1}^{n-1} a_i w_i, \tag{3}$$

evaluates the harmonic quality of a chorale $C = (c_1, \ldots, c_n)$ by considering all pairs of consecutive chords c_i, c_{i+1}. The objective is to maximize $f_h(C)$. In our approach the coefficients represent the style of Bach while the weights represents well known rules from the theory of harmony. More specifically we consider the following three possible cases:

1. c_i and c_{i+1} are chords in the same *major* tonality.
2. c_i and c_{i+1} are chords in the same *minor* tonality.
3. c_{i+1} is identified as a modulation change, that is c_i belongs to the previous tonality while c_{i+1} belongs to a new tonality (regardless of the mode, major or minor).

For each of these cases, we have defined a set of coefficients and weights.

- *Weights.* We relied upon well-known rules from the theory of harmony. We used as reference the description of the major harmonic progressions given by [41, p. 17]. The following table summarizes the "rules" for chord passages in a major tonality.
  To compute the weights w_i we need to define a distribution for the classes "often", "sometimes" and "seldom" defined in Table 2. So, given a specific chord c_i, the weight for the next chord will be a function of the preceding one according to such a distribution. More precisely, we set a probability distribution $(X_{often}, X_{sometimes}, X_{seldom})$ and we assign the weight as follows: c_{i+1} will be one of the chords in the "often" class the X_{often} percentage of the times, one of the chords in the "sometimes" class the $X_{sometimes}$ percentage of the times and one of the chord in the "seldom" class the X_{seldom} percentage of the times.
  To choose the best distribution we have used the following approach.
  We denote by d the distribution used, k the number of iterations and by p_{max} the maximum cardinality of a generated set of solutions. We fix the distribution $d \in Dist$, where $Dist$ is a set of distributions $(X_{often}, X_{sometimes}, X_{seldom})$, built according

Table 3
Weights for chord passages in a major tonality.

Major	Degree						
Degree	*I*	*ii*	*iii*	*IV*	*V*	*vi*	*vii°*
I →	0.266	0.025	0.025	0.266	0.266	0.150	0.002
ii →	0.025	0.400	0.025	0.075	0.400	0.075	0
iii →	0.016	0.016	0.400	0.150	0.016	0.400	0.002
IV →	0.075	0.075	0.025	0.400	0.400	0.025	0
V →	0.400	0.025	0.025	0.075	0.400	0.075	0
vi →	0.050	0.266	0.075	0.075	0.266	0.266	0.002
vii° →	0.400	0.016	0.016	0.400	0.016	0.150	0.002

Table 4
Typical harmonic chord passages in a minor tonality (see [41]). The symbol - means that for a given degree there is no chord in the corresponding class.

Minor	Cadence			
Degree	*Often*	*Sometimes*	*Seldom*	*Never*
i →	*i, iv, V*	*VI*	*ii°, III, vii°*	*VII*
ii° →	*ii°, V*	*IV, VI*	*i, III*	*vii°*
III →	*III, VI*	*iv*	*i, ii°, V*	*vii°*
iv →	*iv, V*	*i, ii°*	*III, VI*	*vii°, VII*
V →	*i, V*	*IV, VI*	*ii°, III*	*vii°, VII*
VI →	*ii°, V, VI*	*III, iv*	*i*	*vii°, VII*
vii° →	*i*	–	*vii°*	–
VII →	*III, VII*	*VI*	*iv*	*i, ii°, V, vii°*

Table 5
Weights for chord passages in a minor tonality.

Minor	Degree							
Degree	*i*	*ii°*	*III*	*iv*	*V*	*VI*	*vii°*	*VII*
I →	0.266	0.016	0.016	0.266	0.266	0.150	0.016	0
ii° →	0.025	0.400	0.025	0.075	0.4	0.075	0	0
III →	0.012	0.012	0.400	0.150	0.120	0.400	0	0.120
iv →	0.075	0.075	0.025	0.400	0.400	0.025	0	0
V →	0.400	0.025	0.025	0.075	0.400	0.075	0	0
vi →	0.050	0.266	0.075	0.075	0.266	0.266	0	0
vii° →	0.950	0	0	0	0	0	0.050	0
VII →	0	0	0.400	0.050	0	0.150	0	0.400

to personal considerations about the meaning of often, sometimes and seldom. We remark that in our experiments, we have $|Dist| = 50$.

Furthermore, we fix the number of iterations $k \in K = \{10, 50, 100, 500, 750, 1000, 2500, 5000, 7500, 10,000\}$, and the max size $p_{max} \in P = \{10, 50, 100, 500, 750, 1000\}$.

For each triple (d, k, p_{max}) we run 5 executions of the music splicing system based on the Tonality-degree representation and we choose the best solution C_{best}, according to the evaluation function described in this section. In order to choose the best distribution we observed the structure of the generated solutions. In our approach we say that a chorale is *well-formed* whether it starts with a *I* degree and ends with the cadence $V - I$. This is a typical condition of well composed chorales. Thus, we compute the average number of music compositions well-formed. As better results, we have obtained that with $d = (80, 15, 5)$, 87% of the solutions are well-formed, with $d = (85, 10, 5)$, 81% are well-formed, with $d = (70, 20, 10)$, 76% are well-formed, with $d = (60, 25, 15)$, 73% are well-formed, and with $d = (50, 40, 10)$, 69% are well-formed.

So, we set as $(X_{often}, X_{sometimes}, X_{seldom}) = (80, 15, 5)$ the distribution used for the experiments which will be described in Section 7.

For example, if c_i is chord *I*, then the coefficient for c_{i+1} will be $0.8/3 \simeq 0.26$ for each one of *I, IV* and *V*, it will be about 0.26 for *vi* and $0.05/2 = 0.025$ for *ii* and *iii*. Table 3 shows all the weights for chord passages within a major tonality.

Similarly, we obtain the weights for chord passages within a minor tonality. Table 4 summarizes the typical chord passages suggested by Piston and DeVoto [41]. The passages are very similar to the ones in a major tonality with some difference due to the possibility of using the *VII* chord in a minor tonality. Table 5 shows the corresponding weights. Notice that for the row the chord *vii°*, there are no chords in the "sometimes" class; hence for this row we use a split of 95–5% among the "often" and the "seldom" class.

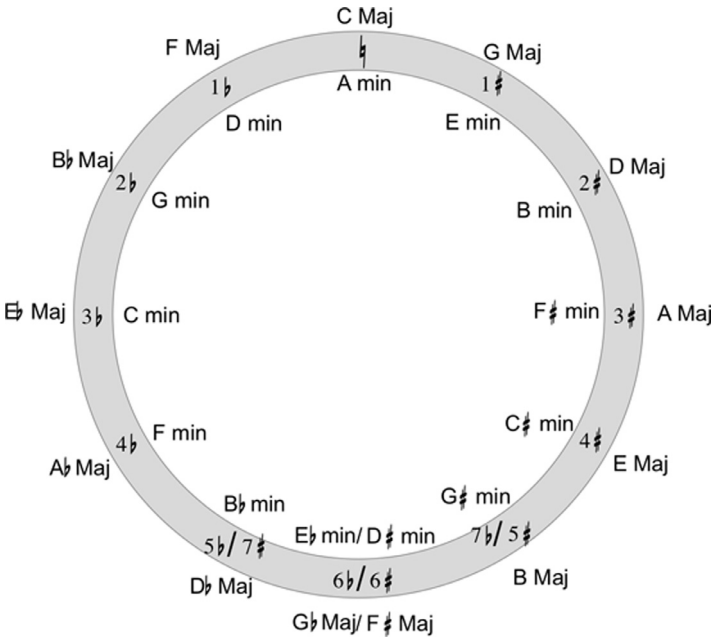


Fig. 2. Circle of fifths.

Table 6
Coefficients for consecutive chords in the same major tonality.

Major	Degree						
Degree	*I*	*ii*	*iii*	*IV*	*V*	*vi*	*vii°*
I	8.1	2.5	0.5	0.7	18.5	1.7	0.3
ii	0.9	0.6	0.4	0.2	10.4	0.2	0.2
iii	0.9	0.3	0.6	0.5	0.8	0.6	0.1
IV	0.6	0.5	0.1	0.3	5.3	0.1	0.2
V	22.5	0.6	1.5	0.8	10.8	2.0	0.1
vi	0.4	0.6	0.6	0.4	2.2	0.7	0.1
vii°	0.3	0.0	0.0	0.1	0.1	0.1	0.1

To assign the weights for consecutive chords when a change of tonality occurs, we used the distance in the circle of fifths (see Fig. 2), between the starting and the ending tonality. More specifically the weight $w_{Modulation}(S, E)$ is the length of the shortest path from the starting tonality S to the ending tonality E. For example, the distance between D major and C minor is 5 because we can go from D major to C minor either counterclockwise using 5 steps or clockwise using 7 steps. As before, the weights for the chord passages are computed by defining a distribution for the classes often, sometimes and seldom, and so the values are always between 0 and 1. The weights for the modulations, instead, are computed as the distance on the circle of fifths of the tonalities. Thus, in order to maintain such weights in a comparable range with the weights for chord passages (between 0 and 1), we normalize the above distance over the maximum possible value, that is 6. For example, the (normalized) harmonic distances from C to G, D and A are, respectively 1/6, 2/6 and 3/6; the (normalized) harmonic distances from C to B♭, E♭ and A♭ are, 2/6, 3/6 and 4/6, respectively, the (normalized) harmonic distances from A♭ to B♭ minor, D♯ minor and G♯ minor are, 1/6, 2/6 and 3/6 respectively.

- *Coefficients.* The coefficients have been obtained by performing a statistical analysis over a large corpus of Bach's chorales. In details, we have written a program that analyses chorales and that extracts information from the used harmonization. In particular, we looked for adjacent chords and we counted the percentage of passages from one chord to the subsequent one. We analyzed a corpus of Bach's chorales, ranging from chorale BWV 253 through chorale BWV 306 and from chorale BWV 314 to chorale BWV 438.

Tables 6–8 summarize the result of the analysis. Notice that the sum of all the percentages in Tables 6 and 7 is smaller than 100 because there have been cases where our program was not able to identify the chords; those cases have not been classified, but simply ignored. In Table 8 many entries are not specified because those specific changes of tonality were not encountered.

As an example, let us consider the music fragment shown in Fig. 1. Thus, the harmonic value: $f_h(C) = f_h(I, V) + f_h(V, I) + f_h(I, V) = 18.5 * 0.266 + 22.5 * 0.4 + 18.5 * 0.266 = 22.7706 = 18.842$.

Table 7
Coefficients for consecutive chords in the same minor tonality.

Minor	Degree							
Degree	*i*	*ii°*	*III*	*iv*	*V*	*VI*	*vii°*	*VII*
I	17.1	0.7	1.4	6.4	15.6	1.4	2.1	1.4
ii°	0.0	0.0	0.0	0.0	3.3	0.0	0.0	0.0
III	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
iv	2.3	0.2	0.2	1.4	9.5	0.4	0.3	0.3
V	22.6	0.0	0.0	0.8	8.0	0.0	0.0	0.1
VI	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
vii°	0.5	0.0	0.1	0.2	0.4	0.0	0.0	0.0
VII	1.1	0.0	0.0	0.0	0.2	0.0	0.0	0.0

Table 8
Coefficients $a_{Modulation}$ for change of tonalities. Major tonalities are shown in bold.

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	*Cm*	*C#m*	*Dm*	*D#m*	*Em*	*Fm*	*F#m*	*Gm*	*G#m*	*Am*	*A#m*	*Bm*
C	–		0.1			2.95		4.6		0.1												1.48		
C#		–																						
D			–					3.2		2.4														1.15
D#				–							1.85													
E					–					1.9														
F						–					2.7													
F#	2.4						–								1.1									
G	4.5		2.9					–									1.65							
G#									–													1.2		
A			2.87							–														
A#						2.5					–									1.6				
B				1.9								–												
Cm													–											
C#m														–										
Dm															–									
D#m																–								
Em								1.7									–							
Fm																		–						
F#m																			–					
Gm											1.4									–				
G#m																					–			
Am	1.5																					–		
A#m																							–	
Bm			1																					–

6.1.2. The melodic function

The melodic function $f_m(C)$ evaluates the melodic quality of a chorale C by performing an "exception analysis" to identify stylistic anomalies and errors. Each exception has an associated severity level that indicates its relative importance: "warning" and "error". A warning exception is intended to highlight a feature that might be stylistically unusual; an error exception indicates a problem that should be corrected. We assign the weights to an exception on the basis of his severity level: 2 for errors and 1 for warnings.

We consider two exception classes: motion exceptions and voicing exceptions. For each class we define a certain number of subtypes (see Table 9). Many other categories and subtypes are possible and the rule system adopted by our algorithm can be easily expanded.

The coefficients have been obtained with a statistical analysis of a large corpus of Bach's chorales, as done for the harmonic function.

Given a chorale C, its melodic value is given by:

$$f_m(C) = \sum_i a_i w_i,$$

where the index i runs over all errors and the values of a_i and w_i are reported in Table 9.

The aim is to minimize $f_m(C)$ (notice that since $f_m(C) \geq 0$ the minimization problem can be easily turned into a maximization problem by considering the value of $-f_m(C)$; in the implementation we exploit this observation).

6.2. The rhythmic transformation of the voices

We remark that when a chorale is generated, it only represents a sequence of chords organized in tonalities areas. In order to make it a musical composition by inserting rhythmic variations, we need to set some rhythmic parameters. Specif-

Table 9
Single voice and two voices errors: weights and coefficients.

Exception Class	w_{mel}	a_{mel}	Meaning
Motion exceptions			
Parallel octaves	2	0.05	consecutive octaves, same two voices
Direct octaves	1	0.01	octaves by similar leap, any two voices
Parallel fifths	2	0.14	consecutive fifths, any two voices
Direct fifths	1	0.02	fifths by similar leap, any two voices
Parallel unisons	2	0.01	consecutive unisons, any two voices
Direct unisons	1	0.01	unisons by similar leap, any two voices
Voice exceptions			
Voice jump	1	6.90	average motion, any voice
Voice crossing	1	0.19	voice above/below adjacent voice, any pair
Voice overlap	1	0.04	voice above/below previous adjacent note
Voice range	1	0.08	voice out of normal vocal range
Voice spacing	1	0.14	wider that an octave, upper voices

ically, we need to set the *meter* (see Section 3) of the composition and to assign to each note a duration. Thus, we apply an operator which:

1. Set the meter value, by choosing randomly one of the following values: $\frac{2}{4}$, $\frac{3}{4}$, $\frac{4}{4}$, $\frac{6}{8}$, $\frac{9}{8}$ and $\frac{12}{8}$.
2. Set the duration of the notes. Initially, the duration of each note is assumed to be equal a beat duration. Then the operator inserts, in each chord c_i, non-harmonic tones (see Section 3.1). Let B_i, T_i, A_i, S_i be the lists of notes for c_i, for each note n_j with a uniform distribution of probabilities, the operator (if possible) adds a non-harmonic tone n'_j in the list, after n_j. Notice that, the insertion of n'_j after n_j involves a redistribution of the original duration of n_j, which we split evenly between the two notes n_j and n'_j (half to each).

7. Experimental analysis

In this section we report the results of a set of tests that we carried out to assess the validity of the proposed Tonality-degree representation. Specifically, we analyzed both the *music quality* of the produced compositions, as well as the *performance* of the system in terms of execution times and memory consumption. Moreover, with regard to the music quality assessment, we analyzed the compositions from a *subjective* point of view by interviewing music experts, and by an *objective* point of view by examining the harmonic and melodic values of the generated music compositions.

In our experiments we compared the composer that leverages the Tonality-degree representation, with our earlier system that exploits the Note representation, and with four other composers that implement meta-heuristics: (1) Multi-Objective Genetic Algorithm, (2) Tabu Search, (3) Simulated Annealing, and (4) Particle Swarm Optimization.

Notice that we have used the chromosome representation to implement the meta-heuristic composers, the organization of the initial population, and the operators as defined in [12]. We remark that each strategy takes as input a figured bass line and produces a complete 4-voice harmonization of the bass line.

7.1. Configuration of the experiments

As defined in Sections 4 and 5, we denote by $\mathcal{S}_{\text{NMSS}}$ and $\mathcal{S}_{\text{TDMSS}}$ the music splicing systems based on the Note representation and the Tonality-degree representation, respectively. We denote by k the number of iterations and by p_{max} the maximum size of a generated language. We fix the number k of iterations in the set $K = \{10,\ 50,\ 100,\ 500,\ 750,\ 1000,\ 2500,\ 5000,\ 7500,\ 10,000\}$, and the max size p_{max} in the set $P = \{10,\ 50,\ 100,\ 500,\ 750,\ 1000\}$.

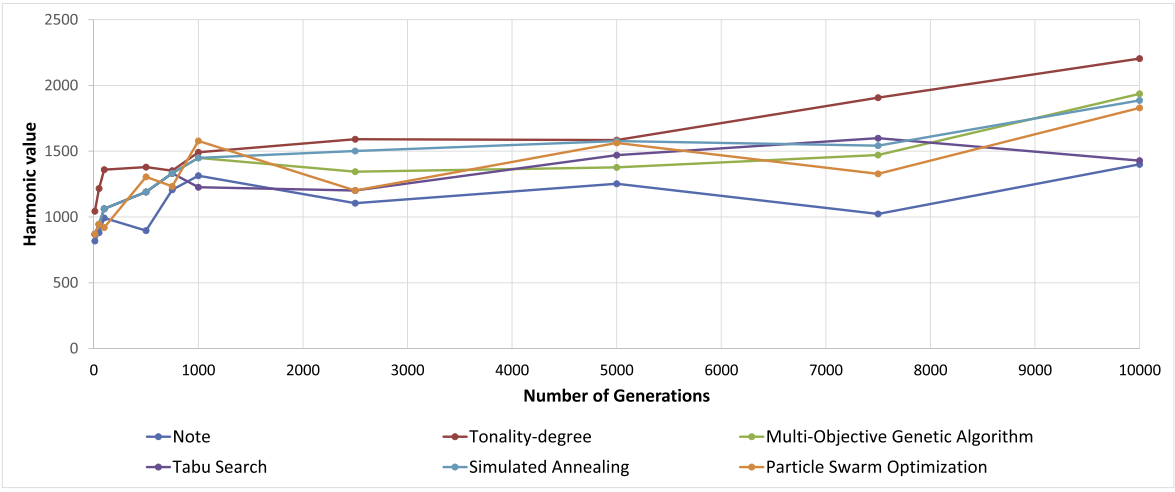
For each pair $(k,\ p_{max})$ we performed the following steps:

1. We run 10 executions of both $\mathcal{S}_{\text{NMSS}}$ and $\mathcal{S}_{\text{TDMSS}}$. For each of these two systems, among the obtained solutions, we choose the best solution C_{best}, according to the evaluation function described in Section 6.1. The bass line of C_{best} will be used in the next step.
2. We run 10 executions of each composer implementing one of the meta-heuristics. Each composer takes as input: (1) the bass line of C_{best}, (2) k the number of generations, (3) p_{max} the size of the initial population, (4) the chromosome definition given in [12], and (5) the evaluation function described in Section 6.1 as the fitness function.
3. For each experiment and for each composer we computed the average harmonic and melodic values, the average execution time and the average memory consumption.

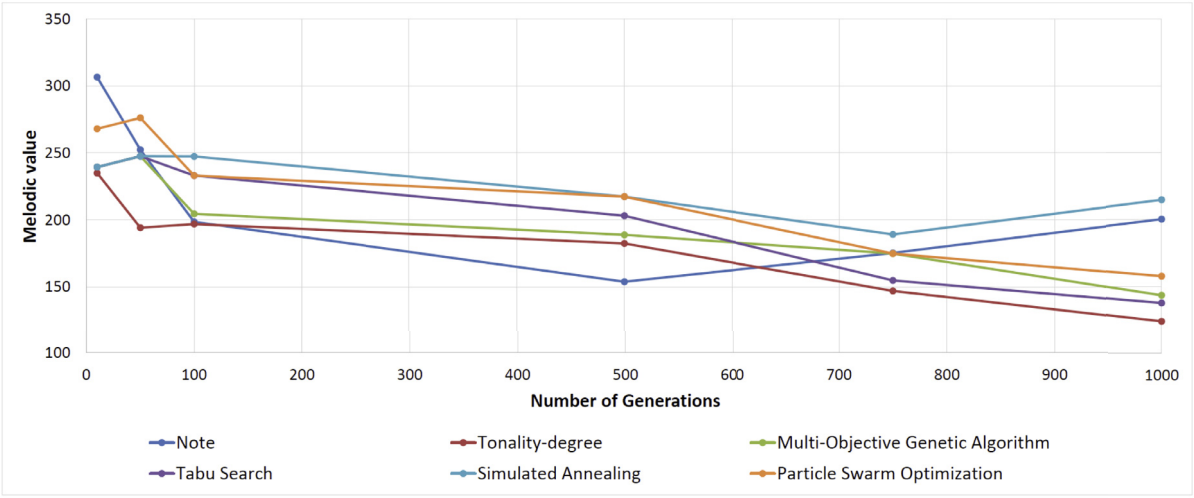
In total, we have $|K| \times |P|$ experiments and in the following section we report the obtained results.

7.2. Results

Results of this study are summarized in Figs. 3 and 5. As aforementioned, we report results about both music quality and performance.



(a) Harmonic value



(b) Melodic value

Fig. 3. Performance results in terms of harmonic and melodic metrics.

7.2.1. Music quality

The quality of music produced by our system has been evaluated quantitatively, by using the corresponding harmonic and melodic analysis, and qualitatively, by interviewing music experts.

Harmonic and melodic analysis. As shown in Fig. 3(a) and (b), the Tonality-degree music splicing system generates better solutions in terms of the average harmonic and melodic values. We have observed that for the harmonic function the systems reached a stable behavior after approximately 10,000 generations. In fact after 10,000 generations the Tonality-degree music splicing system always generates better solutions, and the average value of the better solutions produced by every approach remains substantially the same. Thus, we decided to show the behavior of the composers up to 10,000 generations. About the melodic function: while we have to maximize the harmonic function, we have to minimize the melodic function. Hence, the better solutions are those that have a lower melodic value (in terms of melodic quality). For each approach the corresponding function is decreasing. Furthermore, we have observed that for the melodic metric, the systems reached a stable behavior after approximately 1000 generations. Indeed, after 1000 generations the Tonality-degree music splicing system always generates better solutions, and the average melodic value of the better solutions produced by every approach remains substantially the same. Thus, we decided to show the behavior on the corresponding composers up to 1000 generations.

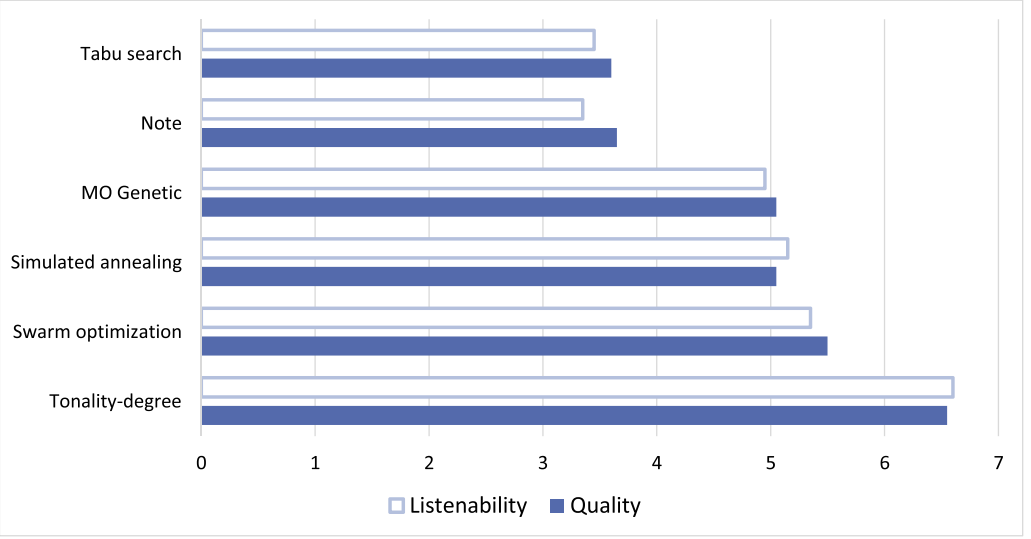


Fig. 4. Results about the average quality and listenability metrics of the best solutions.

Table 10
Correlation coefficients between Harmonic and Melodic values against Quality and Listenability metrics. Correlations significant at 0.01 level (**), and at 0.05 level (*).

	Quality	Listenability
Harmonic values	0.932**	0.930**
Melodic values	−.467*	−.488*

Music experts interview. It is well-known that the quality and the listenability of a musical composition is a subjective judgment. This means that any evaluation may deeply depend on several factors, such as music expertise, personal preferences, and so on. Moreover, the individual's judgments may vary according to several metrics, such as, lyrical content, arrangement, instrumentation, engagement, innovation, emotion, and so on.

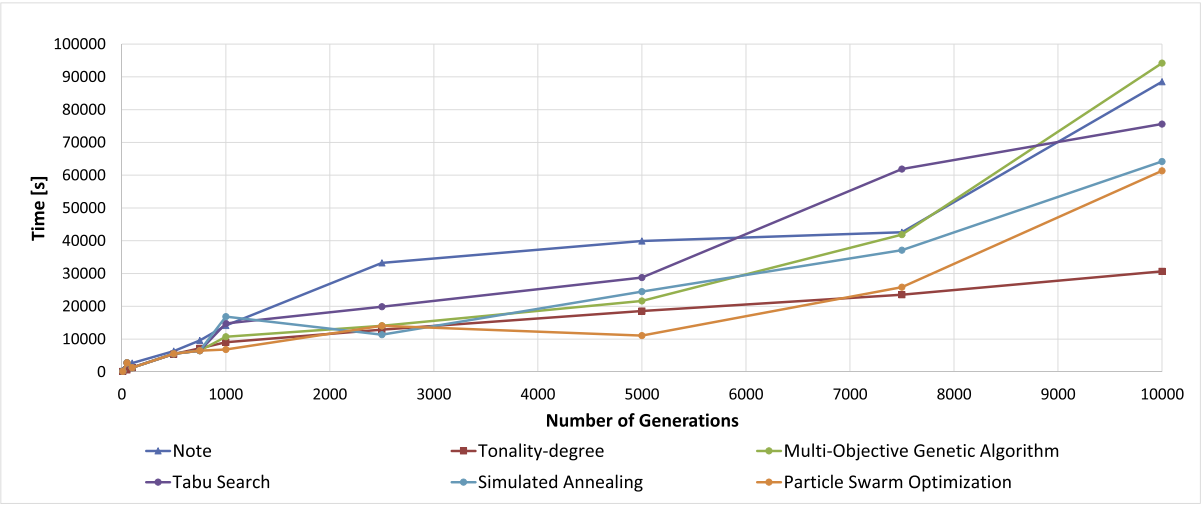
However, in order to derive some insights about the quality and the listenability of the compositions produced by our composer against the quality and the listenability of the compositions produced by composers based on the other metaheuristics, we interviewed a small sample of people with music backgrounds. Specifically, we recruited five domain experts among conservatory's teachers and professional musicians. All participants had more than 20 years of experience in the music field.

We defined a list of MIDI files by selecting, for each system, the best 4 solutions produced during the experiments, for a total of 24 music compositions. Each composition was long about 64 measures. The list was submitted to the experts in a random order. In our music subjective evaluation test, we asked participants to listen to such music pieces and respond to the following questions: (1) *"How do you rate the quality of the composition?"*, and (2) *"How do you rate the listenability of the composition?"* (where rating is on a 7-point Likert scale). The quality metric is solely based on some melodic and harmonic rules. The listenability metric is based solely on the musical perception of the interviewee (not on the resemblance of the music to the original by Bach). We also asked participants to provide a motivation about their judgments (open-ended optional questions).

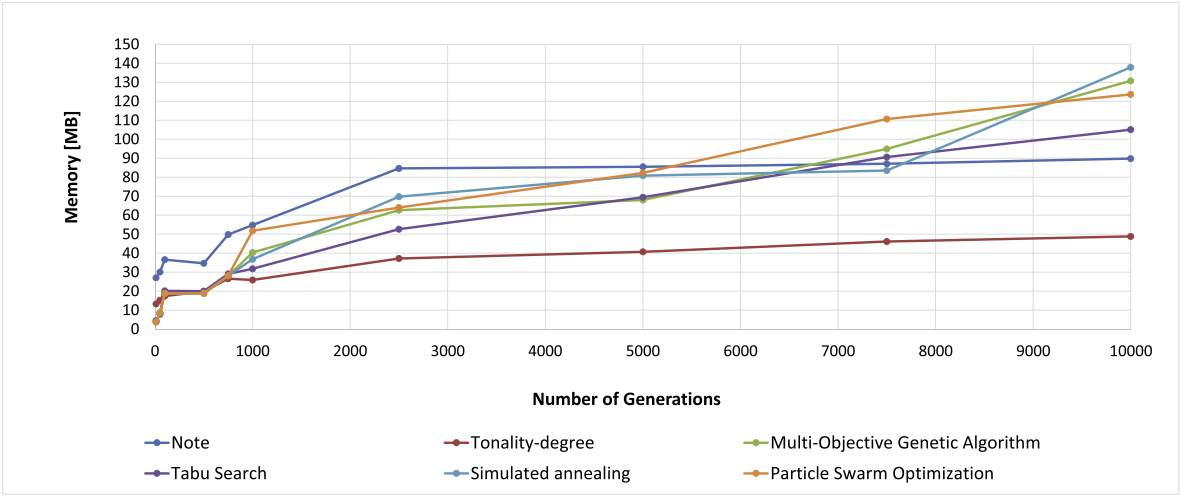
We remark that during the test, participants were not aware of the harmonic and the melodic values of the music compositions.

As shown in Fig. 4, results of test revealed that compositions produced by the Tonality-degree music splicing system have the best rating in terms of both "quality" and "listenability". Participants were highly impressed about the soundness of the compositions produced by the Tonality-degree music splicing system, especially by considering that music was automatically produced by a computer. Moreover, all participants agreed about the fact that such compositions seemed to adhere to the harmonic and melodic rules, and that the generated music seemed never dissonant, and in particular 2 compositions seemed to fully reflect Bach's style. These positive results are due to the fact that our composer is able to start with fragments extracted by Bach's chorales, and through the application of splicing rules, is able to generate music whose structure appear coherent with the Bach's style.

Furthermore, we measured whether a correlation existed between the harmonic and melodic values and quality and listenability subjective metrics, respectively.



(a) Execution time



(b) Memory consumption

Fig. 5. Performance results: execution time and memory consumption.

As we can see in Table 10, a positive correlation exists between harmonic values and quality and listenability parameters, while we found out a negative correlation between melodic values and the subjective parameters. During the splicing process, the quality of the produced solutions increases due to the application of the splicing rules. In our case, the splicing rules are designed according to the harmonic rules. Therefore, during the splicing process, the harmonic improvement is more evident than the melodic one. The perception of the quality and of the listenability by music experts may be influenced by this behavior, and so the relation between quality and harmonic values may be stronger than the relation between listenability and melodic values.

Both the best produced compositions and the questionnaire submitted to music experts are available online.[3]

7.2.2. Performance

In terms of average execution time, when the number of generations is less than 5000, all systems show comparable behaviors (with except for the Note representation). For a number of generations greater than 7000, the Tonality-representation produces results in a shorter time.

[3] http://www.di.unisa.it/~delmal/research/usability/Splicing/.

In terms of average memory consumption, as reported in Fig. 5(b), within 750–1000 generations the results of all strategies are very similar. For a number of generations greater than 750–1000 generations, the Tonality-degree representation produces results by consuming less memory.

Similar to the harmonic function analysis, about the execution time and the memory consumption, after approximately 10,000 generations, the behavior of the systems and the average values for each approach, remain substantially the same, with the Tonality-degree music splicing system that produces always better solutions in a shorter time and by consuming less memory.

In summary, the Tonality-degree music splicing system outperforms the other analyzed approaches by allowing to obtain high quality results. In particular, we observed that the production of high music quality solutions is carried out in a shorter time and using less memory, with respect to the other approaches. In a further analysis,[4] out of the scope of this paper, we performed a user evaluation with a sample of students having skill in music and evolutionary methods. Results showed that the Tonality-degree representation allowed participants to produce code with better quality and quicker. Moreover, participants were satisfied about the easiness of the Tonality-degree representation.

8. Conclusion

Various bio-inspired processes have been used to define algorithms for automatic music composition: evolutionary algorithms, bio-inspired algorithms, formal grammars, cellular automata, machine learning. In this work we provided a new system for automatic music composition based on the splicing model, a mechanism for generating words inspired by the biological recombination of the DNA.

Starting with the approach proposed in [10], we have defined an augmented representation approach in which additional musical information is incorporated into the word representation of the chords. We have showed that the use of this representation improves musical splicing system in terms of time and memory performance, and in terms of harmonic and melodic quality. Also, in this paper we compared the music produced by the system based on the Tonality-degree enhanced representation with the earlier Note representation and several meta-heuristics. The results show that the music splicing approach is a new valid bio-inspired approach for automatic music composition, in alternative to the widely studied automatic composers based on meta-heuristics strategies.

Some future work on classical music include: (1) an investigation of the behavior and the efficacy of a music splicing system by using the corpus of chorales of a different composer than Bach, (2) a statistical analysis on a very general corpus of chorales composed by the main classical composers, to compute an objective distribution for the weights used to evaluate the compositions. We remark that to build such a corpus is a very arduous task, due to the lack of the needed quantity of significant music materials in electronic format. Additionally, a further investigation could analyze the efficacy of a splicing system on different types of music genres such as contemporary music.

In this paper, in order to assess the quality of the produced music we exploited only considerations about the melodic and harmonic aspects. There are several other aspects that could be considered, such as, the structure of the composition, the overall logical coherence, the rhythmic aspect, and so on. Considering all these other facets will probably lead to composition that are much closer to a Bach's original. The current output is certainly not close to being similar to a Bach's composition but represents a first step.

It would be also interesting to investigate other applications of splicing systems based on the approach proposed in this work. By leveraging the application of the combinatorics on words and formal languages in different fields, including bioinformatics, data compression and algorithms, one can investigate whether splicing system strategies are able to produce better results with respect to meta-heuristics already used in those fields.

References

[1] G. Acampora, J.M. Cadenas, R. De Prisco, V. Loia, E.M. Ballester, R. Zaccagnino, A hybrid computational intelligence approach for automatic music composition, in: Proceedings of IEEE International Conference on Fuzzy Systems, Taipei, Taiwan, 27–30 June, 2011, pp. 202–209.
[2] G. Assayag, C. Rueda, M. Laurson, C. Agon, O. Delerue, Computer-assisted composition at IRCAM: from patchwork to openmusic, Comput. Music J. 23 (3) (1999) 59–72.
[3] J.A. Biles, GenJam: a genetic algorithm for generating jazz solos, in: Proceedings of International Computer Music Conference, 1994, pp. 131–137.
[4] J.A. Biles, Genjam in perspective: a tentative taxonomy for GA music and art systems, Leonardo 36 (1) (2003) 43–45.
[5] P. Bonizzoni, C. De Felice, R. Zizza, The structure of reflexive regular splicing languages via schützenberger constants, Theor. Comput. Sci. 334 (1–3) (2005) 71–98.
[6] P. Bonizzoni, N. Jonoska, Regular splicing languages must have a constant, in: Developments in Language Theory, in: Lecture Notes in Computer Science, 6795, Springer, 2011, pp. 82–92.
[7] D. Cope, Experiments in Musical Intelligence, Computer Music and Digital Audio Series, A-R Editions, 1996.
[8] D. Cope, The Algorithmic Composer, Computer Music and Digital Audio Series, A-R Editions, 2000.
[9] D. Cope, Virtual Music, The MIT Press, 2004.
[10] C. De Felice, R. De Prisco, D. Malandrino, G. Zaccagnino, R. Zaccagnino, R. Zizza, Chorale music splicing system: an algorithmic music composer inspired by molecular splicing, in: Proceedings of the 4th International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design, EvoMUSART 2015, Copenhagen, Denmark, April 8–10, 2015, 2015, pp. 50–61.
[11] R. De Prisco, A. Eletto, A. Torre, R. Zaccagnino, A neural network for bass functional harmonization, in: Proceedings of International Conference on Applications of Evolutionary Computation – volume Part II, EvoCOMNET'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 351–360.

[4] http://www.di.unisa.it/~delmal/research/usability/Splicing/UserEvaluation.pdf.

[12] R. De Prisco, G. Zaccagnino, R. Zaccagnino, EvoBassComposer: a multi-objective genetic algorithm for 4-voice compositions, in: Proceedings of Genetic and Evolutionary Computation Conference, GECCO, ACM, 2010, pp. 817–818.
[13] R. De Prisco, G. Zaccagnino, R. Zaccagnino, A genetic algorithm for dodecaphonic compositions, in: Proceedings of International Conference on Applications of Evolutionary Computation - Volume Part II, EvoApplications'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 244–253.
[14] R. De Prisco, R. Zaccagnino, An evolutionary music composer algorithm for bass harmonization, in: Proceedings of EvoWorkshops, in: Lecture Notes in Computer Science, 5484, Springer, 2009, pp. 567–572.
[15] K. Ebcioglu, An expert system for harmonizing four-part chorales, in: S.M. Schwanauer, D.A. Levitt (Eds.), Mach. Models Music, MIT Press, Cambridge, MA, USA, 1992, pp. 385–401.
[16] M. Geis, M. Middendorf, An ant colony optimizer for melody creation with baroque harmony, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25–28 September 2007, Singapore, 2007, pp. 461–468.
[17] C. Gimenes, E. Miranda, C. Johnson, On the learning stages of an intelligent rhythmic generator, in: Sound and Music Computing, Salerno, Italy, 2005, pp. 244–253.
[18] C. Gimenes, E. Miranda, C. Johnson, Towards an intelligent rhythmic generator based on given examples: a memetic approach, Sound and Music Computing, Glasgow, UK, 2005.
[19] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours, Bull. Math. Biol. 49 (1987) 737–759.
[20] T. Head, D. Pixton, Splicing and regularity, in: Recent Advances in Formal Languages and Applications, Springer, 2006, pp. 119–147. Esik et al. editors
[21] T. Head, G. Păun, D. Pixton, Language theory and molecular genetics: generative mechanisms suggested by DNA recombination, in: Handbook of Formal Languages, 2, Springer-Verlag, 1996, pp. 295–360.
[22] D. Herremans, K. Sörensen, Fux, an android app that generates counterpoint, in: Proceedings of 2013 IEEE Symposium on Computational Intelligence for Creativity and Affective Computing (CICAC), 2013, pp. 48–55.
[23] D. Herremans, S. Weisser, K. Sörensen, D. Conklin, Generating structured music for bagana using quality metrics based on markov models, Expert Syst. Appl. 42 (21) (2015) 7424–7435.
[24] L. Hiller, Computer music, Sci. Am. 201 (6) (1959) 737–759.
[25] L. Hiller, L. Isaacson, Experimental Music, McGraw-Hill, 1959.
[26] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
[27] A. Horner, L. Ayers, Harmonization of musical progression with genetic algorithms, in: Proceedings of International Computer Music Conference, 1995, pp. 483–484.
[28] A. Horner, D. Goldberg, Genetic Algorithms and Computer Assisted Music Composition, Technical report, University of Illinois, 1991.
[29] Jacob, Composing with Genetic Algorithms, Technical report, University of Michigan, 1995.
[30] L. Kari, S. Kopecki, Deciding whether a regular language is generated by a splicing system, in: DNA Computing and Molecular Programming, in: Lecture Notes in Computer Science, 7433, Springer, 2012, pp. 98–109.
[31] D. Lehmann, Harmonizing melodies in real-time: the connectionist approach, in: Proceedings of the International Computer Music Association, 1997, pp. 27–31.
[32] R. McIntyre, Bach in a box: the evolution of four part baroque harmony using the genetic algorithm., in: Proceedings of International Conference on Evolutionary Computation, 1994, pp. 852–857.
[33] B. Melián, J. Moreno, J. Moreno, Metaheurísticas: una visión global, Intel. Artif. Rev. Iberoam. Intel. Artif. 7 (19) (2003) 7–28.
[34] E. Miranda, Composing Music with Computers, Focal Press (2001).
[35] E. Miranda, On the evolution of music in a society of self-taught digital creatures, Digit. Creativity 1 (1) (2003) 29–42.
[36] E. Miranda, On the music of emergent behaviour: what can evolutionary computation bring to the musician? Leonardo 36 (1) (2003) 55–58.
[37] E. Miranda, S. Kirby, P. Todd, On computational models of the evolution of music: from the origins of musical taste to the emergence of grammars., Contemp. Music Rev. 22 (3) (2003) 91–111.
[38] A.F. Moore, Categorical conventions in music discourse: style and genre, in: Music & Letters, 82, 2001, pp. 432–442.
[39] A. Moray, K.I.W. Christopher, Harmonising chorales by probabilistic inference, in: Proceedings of Advances in Neural Information Processing Systems, 2004, p. 2004.
[40] S. Phon-Amnuaisuk, Composing using heterogeneous cellular automata, in: Proceedings of EvoWorkshops, in: Lecture Notes in Computer Science, 5484, Springer, 2009, pp. 547–556.
[41] W. Piston, M. DeVoto, Harmony, Norton, 1987.
[42] D. Pixton, Regularity of splicing languages, Discret. Appl. Math. 69 (1–2) (1996) 101–124.
[43] R.D. Prisco, G. Zaccagnino, R. Zaccagnino, A multi-objective differential evolution algorithm for 4-voice compositions, in: Proceedings of the 2011 IEEE Symposium on Differential Evolution (SDE), 2011, pp. 1–8.
[44] G. Păun, On the splicing operation., Discret. Appl. Math. 70 (1996) 57–79.
[45] B. Schottstaedt, Automatic Species Counterpoint, Technical Report, Stanford, 1984. STAN-M-19
[46] R. Whorley, D. Conklin, Proceedings of 5th international conference on Mathematics and computation in music, MCM 2015, London, UK, june 22–25, 2015, proceedings, Springer International Publishing, Cham, pp. 64–70.
[47] G. Wiggins, G. Papadopoulos, S. Amnuaisuk, A. Tuson, Evolutionary methods for musical composition, in: Proceedings of CASYS98 Workshop on Anticipation, Music and Cognition, 1998.
[48] R. Zizza, Splicing systems, Scholarpedia 5 (7) (2010) 9397.