Creative DNA Computing: Splicing Systems for Music Composition

Rocco Zaccagnino (✉ rzaccagnino@unisa.it)

University of Salerno: Università degli Studi di Salerno <https://orcid.org/0000-0002-9089-5957>

Roberto De Prisco

University of Salerno: Università degli Studi di Salerno

Research Article

Keywords: DNA computing, splicing systems, automatic music composition, artificial creativity

Posted Date: November 1st, 2021

DOI: <https://doi.org/10.21203/rs.3.rs-485629/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Creative DNA Computing: Splicing systems for music composition

Rocco Zaccagnino · Roberto De Prisco

Received: date / Accepted: date

Abstract Splicing systems are a form of DNA computing as they mimic the recombination process among DNA molecules. This work discusses the use of splicing systems to build automatic tools for reproducing human beings' *creativity*, in the context of *automatic music composition*. More specifically, this work describes three general splicing system approaches for automatic music composition, and their application to two specific cases, namely composing 4-voice music and composing Jazz solos in a given style. Examples of music composed by the systems are presented.

Keywords DNA computing · splicing systems · automatic music composition · artificial creativity

1 Introduction

Research on *unconventional models of computation* aims at defining paradigms and algorithms inspired by, or physically implemented in, chemical, biological and physical systems [1]. As explained in [2], an unconventional model is a “*computing scheme that today is viewed as unconventional because its time hasn't come yet-or is already gone*”. Within the scientific community, there is the growing consensus that the increasingly frequent studies on unconventional models of computation are to be attributed to the fact that one day the limit of today's conventional computing paradigms will be reached. The consequence is an ever-growing need for new models that can tackle complex problems using the ever-accelerating advances in technological devices.

Rocco Zaccagnino
Computer Science Department, University of Salerno, Salerno, Italy
E-mail: rzaccagnino@unisa.it

Roberto De Prisco
Computer Science Department, University of Salerno, Salerno, Italy
E-mail: robdep@unisa.it

An additional motivation for the interest about unconventional computational methods, comes from Artificial Intelligence [3]. The search for *mechanical intelligence*, i.e., the attempt to equip machines with intelligence, at some point had to consider a fundamental fact: artificial intelligence is different from biological intelligence. This distinction essentially depends on the deep difference between the learning process of machines and that of humans. Humans have the ability to learn much more quickly from small sets of data and possess an innate ability to construct abstractions. Turing himself has felt the need to imitate the evolutionary self-learning and organizational modeling capabilities of living beings [4]. Therefore, it has become essential for AI researchers to focus on phenomena such as emotional and social intelligence (instinct, creativity, emotions, etc).

The problem of reproducing human beings' *creativity* has always been a challenging task for the AI community and specifically for the research on bio-inspired systems. There are several examples of human real-life situations in which creativity plays a fundamental role, such as strategic ability in games, intuition in mathematical calculations and proofs, improvisational ability in unexpected situations, and inspiration in the creation of artistic works. Among these, the *musical* ability, or *musical estrus*, is a challenging task.

"But, what musical ability means exactly?" In the context of this work, musical ability is the ability of "composing" new music. The focus is on systems capable of automatically producing music by means of a computer program, without *any* human intervention.

Bio-inspired computational methods have been largely used to tackle the challenge of reproducing the musical abilities of humans. Among such methods there are Cellular Automata, e.g., [5, 6, 7, 8]), Evolutionary Algorithms, e.g., [9, 10, 11, 12, 13], and various DNA computing approaches, e.g., [14, 15, 16, 17].

Splicing systems [18] are an interesting DNA computing model of a two-steps recombination process between DNA molecules: (i) *restriction enzymes* (proteins) recognize a specific pattern in a DNA sequence and then cut the molecule in a specific point inside the recognized pattern; (ii) *ligase enzymes* paste together properly matched fragments, under chemical conditions. Although not new, splicing systems theory is still a challenging field of theoretical research, as shown by recent literature [22, 23, 24, 25]. The deep theoretical investigation on splicing systems is not well balanced by practical applications of the systems.

Main contribution. The aim of this work is that of delving into the use of splicing systems as a practical tool for unconventional music composition and it is based on work presented in [19, 20, 21, 22]. Except for the ones presented in this work, there are no previous models for music composition based on splicing systems. The main contributions of this work can be summarized as follows: (i) introduction of a novel technique, based on splicing system, for unconventional music composition; (ii) classification of the methods proposed in literature into three general approaches; (iii) in-depth description of the approaches; (iv) use of the approaches to produce example of music output. It

should be remarked that the use of splicing system as an evolutionary method is also novel and can be considered in its own as an additional contribution.

Organization. In Section 2, needed basic notions are reviewed and, in particular, Subsection 2.2 provides details about the use of splicing systems as music composers. Then, subsequent sections, dig into details about the construction of such systems: Sections 3, 4 and 5 describe three different approaches for the constructions of automatic music composers based on splicing systems. At the end of each section there is a sample music output. Finally, Section 6 contains concluding remark.

2 Background

The following discussion is not intended to cover all the needed background knowledge; it is assumed that the reader is familiar with both subjects, that is, splicing systems and music theory, and thus that he/she is able to follow what is written in this section. If not, the reader is encouraged to acquire a minimal background about the basic notions, mentioned in Subsections 2.1 and 2.3. Subsection 2.2 talks about splicing systems used as automatic composers discussing the key points, formulated as *composition questions*, that the system needs to take into account.

2.1 Splicing systems

Following Head's work [23], in which the biochemical operation of splicing has been formalized as an operation on strings, there have been further development, in particular Păun [24] and Pixton [25] have introduced alternative operations. Each of these operations take as input two words and can generate either one new word, 1-splicing, or 2 new words, 2-splicing. The splicing systems presented in this work use the 2-splicing operation.

A Păun's 2-splicing rule r is $r = u_1\#u_2\$u_3\#u_4$, where u_1, u_2, u_3, u_4 are strings over a given alphabet $\mathcal{A}$ such that $\#, \$ \notin \mathcal{A}$. The words u_1u_2 (concatenation of u_1 and u_2), and u_3u_4 (concatenation of u_3 and u_4) are called *sites* of r . Roughly speaking, a site is a substring of the input words where it is possible to "cut" the input words. Hence a rule identifies two points where it is possible cut the input strings. Given such a rule r , and two words x and y belonging to the current language, if x contains the first site u_1u_2 , that is $x = x_1u_1u_2x_2$, and y contains the second site u_3u_4 , that is $y = y_1u_3u_4y_2$, rule r produces the strings $z = x_1u_1u_4y_2$, $w = y_1u_3u_2x_2$. This operation is denoted by $(x, y) \vdash_r (z, w)$.

Splicing systems generate languages based on the splicing operation. From an initial set of words (often called the initial language) the system applies the rules to produce new words which are added to the set of words. This process can generate an infinite language.

Formally, a *splicing system* is a triple $\mathcal{S} = (\mathcal{A}, \mathcal{I}, \mathcal{R})$, where $\mathcal{A}$ is a finite alphabet such that $\#, \$ \notin \mathcal{A}$, $\mathcal{I} \subseteq \mathcal{A}^*$ is the initial language and $\mathcal{R} \subseteq$

$\mathcal{A}^* \# \mathcal{A}^* \$ \mathcal{A}^* \# \mathcal{A}^*$ is the set of rules. A splicing system $\mathcal{S}$ is finite if $\mathcal{I}$ and $\mathcal{R}$ are both finite sets. Let $L \subseteq \mathcal{A}^*$ and $\sigma'(L) = \{w', w'' \in \mathcal{A}^* \mid (x, y) \vdash_r (w', w''), x, y \in L, r \in \mathcal{R}\}$. The splicing operation on languages is defined as follows:

$$\begin{aligned}\sigma^0(L) &= L, \\ \sigma^{i+1}(L) &= \sigma^i(L) \cup \sigma'(\sigma^i(L)), \quad i \geq 0, \\ \sigma^*(L) &= \bigcup_{i \geq 0} \sigma^i(L).\end{aligned}$$

Definition 1 (Păun splicing language) Given a splicing system $\mathcal{S} = (\mathcal{A}, \mathcal{I}, \mathcal{R})$, the language generated by $\mathcal{S}$ is $L(\mathcal{S}) = \sigma^*(\mathcal{I})$. A language L is $\mathcal{S}$ -generated if there exists a splicing system $\mathcal{S}$ such that $L = L(\mathcal{S})$.

2.2 Splicing systems and music composition

The basic idea in using a splicing system for music composition is that of treating music compositions as words and to view the music compositional process as the results of operations on words. In such a perspective, a splicing system becomes a tool for generating languages of *musical words*.

The definition of a music splicing system is closely related to the specific musical compositional process taken into consideration. Since these are systems that manipulate words to create new words, the first question concerns the choice of a *suitable representation* of the music in terms of words. As shown in [21], this choice affects the effectiveness of the systems in generating good quality musical compositions in an acceptable time.

Once the representation to be used has been defined, it is necessary to define *suitable splicing rules* in such a way that they produce new words suited for the musical context being considered. The definition of the rules is very important because the rules determine the language being generated. Furthermore, from a theoretical point of view, the splicing system is an infinite word generative mechanism. In practical real-word applications, one needs to transform splicing into a finite process, i.e., a process that after a finite number of steps produces one or more qualitatively acceptable solutions. The idea is to transform the splicing process into an *evolutionary process* based on the use of an *evaluation function* (fitness) of the words, and a stop criterion of the generative process. Usually this criterion is chosen as a maximum number of iterations or a qualitative reference value that represents the desired quality of the music solutions produced.

To summarize, in the automatic process used by splicing systems to generate music, it is necessary to tackle the following three key points, that can be formulated as *composition questions*:

- cq1: “What is a suitable word-based representation in order to generate good quality musical compositions in acceptable time?”
- cq2: “How should splicing rules be defined in order to generate new words appropriate for the musical context considered?”

cq3: “How the splicing process can be transformed into an evolutionary process using appropriate evaluation functions and stop criterion?”

The splicing system composers presented in this work can be classified in three categories: (i) static evaluation-based, (ii) statistical evaluation-based, and (iii) machine learning-based. Sections 3, 4 and 5 will discuss the 3 different approaches analyzing how they address the above key points. For each approach a concrete specific musical problem will be considered (actually for the first two approaches the music problem considered is the same).

2.3 Tempered music system and classical rules

The tempered music system is the 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 it is useful to 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 the human 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 Bb, B, C, C# or Db, D, D# or Eb, E, F, F# or Gb, G, G# or Ab.

This system is based on *tonalities*. Western music considers two kinds of tonalities (or modes), *major and minor*, although other tonalities are sometimes used, especially in Jazz music, like for example the dorian and mixolydian modes. For each note, thus, there is a major tonality and a minor tonality, for a total of 12 major tonalities and 12 minor tonalities. A tonality defines a specific sequence of notes, called *scale* (of the tonality). 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. The position of a note within the scale is the *degree* of the note. Thus in C major, C is the first degree, D is the second degree and so on. Degrees are denoted with the roman numerals I, II, III, ..., VII. Given a tonality and the corresponding scale, the notes that do not belong to the scale are called *non-harmonic tones*. Non-harmonic tones can be classified in: *passing tones*, *auxiliary tones*, *appoggiatura tones* and *suspension tones* (see [26] for more details). Each music composition has its own *main tonality*.

Music improvisation refers to the ability of playing music extemporaneously, without planning or preparation, by inventing variations on a melody or creating new melodies on the spot. In the context of music improvisation, the term *solo* indicates an improvised melody, which can be transcribed for later study or performance. There are many musical genres in which the improvisation assumes a fundamental role during a performance. Usually the

improviser's choices are conditioned by his/her own personal experience or preferences, and also by the specific period to which music belongs. It is interesting to notice that music performers of the same period used similar music choices during their improvisations. Furthermore, during the improvisation, musicians often borrow and customize musical excerpts from previous performances (both own performances and those of other musicians). For every music performer it is possible to found music features that characterize his specific style, and an expert ear is able to perceive such aspects. Thus, given a corpus of solos, it is possible to extract such significant features and use them to recognize the style used.

One of the key points for extracting the significant features is the study of the role that each music note assumes in the specific context (chord) in which it is played. Formally, let Ch be a music chord, $S(Ch)$ be a scale chosen by the musician and n be a note played on Ch . The degree of a note n with respect to a scale $S(Ch)$ is denoted by $Degree(n, S(Ch))$. Since there are 7 notes in the scale and 12 notes in an octave, 7 degrees belong to the scale $S(Ch)$ and 5 notes are outside the scale $S(Ch)$. For example let $Ch = Dm7$ and $S(Dm7) = D, E, F, G, A, B, C$ (dorian mode). Let $n = G$, then $Degree(G, S(Dm7)) = IV$. Let $n = Eb$ then $Degree(Eb, S(Dm7)) = bII$.

Notice that the musician's choice of $S(Ch)$ for a chord Ch is a crucial choice for his/her style. Typically, it is possible to associate several scales to the same chord. For example, in Jazz music, a dominant chord is often substituted by the tritone chord. For example, let $Ch = G7$. Very traditional musicians could choose $S(G7) = G, A, B, C, D, E, F$ (mixolydian mode built on the V grade of the Cmaj scale). An alternative choice is $S(Ch) = Db, Eb, F, Gb, Ab, Bb, Cb$ (mixolydian mode built on the V grade of the Gbmaj scale). Obviously, the role of a note can be different depending on the chosen scale. In the previous example, let $n = F$. Then, if $S(G7) = G, A, B, C, D, E, F$ then $Degree(F, S(G7)) = VII$. Otherwise, if $S(Db7) = Db, Eb, F, Gb, Ab, Bb, Cb$ then $Degree(F, S(G7)) = III$. One can say that chord $G7$ has been substituted by the chord $Db7$. This process is known in Jazz as *substitution*. Further details can be found in [27].

3 Approach 1: Splicing systems based on *static weights*

In this section the first approach is presented: music splicing systems that use an evaluation function, for measuring the quality of the composition, whose definition is based on "weights" chosen by empirical observations, so that good patterns have heavier weights. The specific music problem considered is that of polyphonic k -voice compositions. The specific case of $k = 4$ (chorales) has been considered in [19,21,22]. Here the description is generalized to any k . First the music problem is formally described. Then the music splicing composer for polyphonic compositions is presented. Details regarding how the splicing composer addresses each one of the composition questions cq1, cq2, and cq3 (see Section 2.2) are provided. At the end of the section an example of music output is given.

3.1 The music composition problem: k -voices music

A k -voices music piece is composed of k voices (instruments or singers). Each voice can play notes in an admissible range. Music is organized in a sequence of *measures*, each one divided in *beats*. In each beat k notes can be simultaneously played (one voice plays/sings one note). The k notes played in a specific beat are called *chord*. Chords are built on the degrees of the scale of the tonality used. Chords are built on the degrees of the scale of the tonality used. The degrees of a scale are indicated with: $I, II, III, IV, V, VI, VII$. Usually, capital letters indicate major chords while small letters indicate minor chords (see [26] for details).

A k -voices music piece can be analyzed from a *vertical* point of view, i.e., the *harmonic* dimension of the piece, and from an *horizontal* point of view, i.e., the *melodic* dimension of the piece. The harmonic dimension regards the sequence of chords of the piece, while the melodic dimension regards the sequence of notes (melodies) played by each voice. As a consequence, rules can be established both for the harmonic and melodic dimension. As regarding the harmonic dimension, the rules are defined by considering specific chord sequences, called *cadences*, that have specific musical functions. The most used cadences in classical music are: $V \rightarrow I, II \rightarrow V, VI \rightarrow II, V \rightarrow VI, IV \rightarrow IV, IV \rightarrow I$ and $III \rightarrow VI$. Each cadence can be “encoded” through a set of splicing rules. As regarding the melodic dimension, there are also 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. According to music theory rules, for any given pair of lines, some specific patterns should be avoided (see [26] for details): (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. Splicing rules can model such musical rules.

3.2 Approach details

As explained in Section 2, a music splicing composition system is made up of 3 components: an alphabet, an initial set of words and a set of rules. The system $\mathcal{S}_1$ considered in this section will be specified by describing these three components, denoted with $\mathcal{S}_1 = (\mathcal{A}_1, \mathcal{I}_1, \mathcal{R}_1)$, with respect to the three composition questions discussed earlier.

cq1: word-based representation. The first step towards the definition of a word-based representation is the choice of “how to represent a composition using a word”. Let us consider a k -voice composition $C = (c_1, \dots, c_n)$ where each c_i is a chord, with $i = 1, \dots, n$. Obviously there are several ways to represent C by using a word. One possible solution consists in representing

each chord c_i with a specific word w_i and then representing C with the word $\mathcal{W}(C) = w_1 \cdots w_n$.

At this point, the problem comes down to deciding “how to define each word w_i ”. Such a decision depends on how much information regarding c_i one wants to put into w_i . As explained in Section 3.1, for the k -voice music problem, a chord is the set of the k notes played in a specific beat. Thus, in w_i one has to insert information regarding the k notes in c_i . The complexity of w_i depends on how much information regarding each note needs to be considered. One very simple solution proposed in this approach consists in representing each note using information regarding 3 basic parameters: (i) the *voice* that plays the note, (ii) the *name* of the note, and (iii) the *octave* in which it is located. More precisely, a chord c_i will be represented as $w_i = v_1 x_1 o_1 \cdots v_k x_k o_k$, where, for each $j = 1, \dots, k$:

- v_j is the voice which plays the j -th note in c_i ,
- x_j is the name of the j -th note in c_i ,
- o_j is the octave in which the j -th note in c_i is placed.

Clearly, it is necessary to define the *voice alphabet* used to indicate the voices, the *note alphabet* used to indicate the name of the notes, and the *octave alphabet* used to indicate the octaves. These alphabets are as follows: (i) the voice alphabet $\mathcal{A}_V = \{v_1, \dots, v_k\}$, where v_i indicates the i -th voice, (ii) the note alphabet $\mathcal{A}_N = \{C, C\#, Db, D, D\#, Eb, E, F, F\#, Gb, G, G\#, Ab, A, A\#, Bb, B\}$, and (iii) the octave alphabet $\mathcal{A}_O = \{1, \dots, m\}$.

Finally, let $\mathcal{A}_1 = \mathcal{A}_V \cup \mathcal{A}_N \cup \mathcal{A}_O$. Using $\mathcal{A}_1$ it is possible to represent k -voice music compositions as words.

Example 1 Consider the 4-voice music fragment C in Figure 1 (a fragment of Chorale BWV 6.6), $C = (c_1, c_2, c_3, c_4)$. Each chord c_i is represented by a word, w_i , with $w_1 = “v_1 A4 v_2 C5 v_3 F5 v_4 C6”$, $w_2 = “v_1 F4 v_2 F5 v_3 A5 v_4 C6”$, $w_3 = “v_1 Bb4 v_2 F5 v_3 Bb5 v_4 D6”$, $w_4 = “v_1 G4 v_2 G5 v_3 D5 v_4 Bb5”$. The entire music fragment is represented by the word $w = w_1 w_2 w_3 w_4 = “v_1 A4 v_2 C5 v_3 F5 v_4 C6, v_1 F4 v_2 F5 v_3 A5 v_4 C6, v_1 Bb4 v_2 F5 v_3 Bb5 v_4 D6, v_1 G4 v_2 G5 v_3 D5 v_4 Bb5”$.

cq2: rules definition. Rules use the word-based definition above described to generate words representing k -voices music composition. Notice that, in this approach, only harmonic theory rules are considered, and thus the splicing rules are defined according to such music rules.

Consider first the initial set of words $\mathcal{I}_1$ on which the rules will be initially applied. This presupposes the choice of a corpus of pre-composed k -voices music pieces, which is representative of the specific musical style (and genre) on which one wants to set the problem. As an example, in [19, 21] the initial corpus of music is a set of J. S. Bach’s chorales.

Once the corpus of pre-composed k -voices music pieces has been chosen, each one of the pieces needs to be transposed in every tonality, and each of such transposed piece is inserted into the *ground data set*, named $\mathcal{G}$. Obviously,

Fig. 1: A fragment of BWV 6.6

if t is the cardinality of the chosen corpus, then $|\mathcal{G}| = 12 * t$ pieces. Let $\mathcal{I}_1$ be the set containing the $12 * t$ words (obtained by applying the word-based representation above defined) that represents the pieces in $\mathcal{G}$. Notice that these are not the only words in $\mathcal{I}_1$: there are other words that will be explained shortly, when the rules will be presented.

Since the harmonic rules are defined on the sequences of chords, also single chords, from the pieces in $\mathcal{G}$, are added to $\mathcal{I}_1$, so that the produced music will be similar to that in the ground data set. During the chord extraction, information about its original function, that is, the degree of the scale on which the chord is built, is attached to the chord. This information is crucial in rearranging the chords, by means of splicing rules, so that specific sequences of chords will be produced. As a side note, notice that Approach 2 (described in Section 4) will use an enhanced representation to embed such information directly in the words.

As done for the initial set, each extracted chord is transposed in all 12 tonalities. Let $Chords(\mathcal{G})$ the set of these chords. For each extracted chord $c \in Chords(\mathcal{G})$, information provided by the harmonic analysis, about the degree on which the chord is built, is stored. The set of words associated to $Chords(\mathcal{G})$ is $\mathcal{W}(Chords(\mathcal{G}))$.

Consider now the definition of the splicing rules $\mathcal{R}_1$. As explained before, the set of splicing rules are modeled on classical harmonic rules. In particular, this approach considers a set of *cadences*, which are specific sequence of chords. In particular the cadences considered are $V \rightarrow I$, $I \rightarrow IV$, $II \rightarrow V \rightarrow I$, $IV \rightarrow V \rightarrow I$, $I \rightarrow II \rightarrow III$, $III \rightarrow II \rightarrow I$, and $V \rightarrow VI \rightarrow VII \rightarrow I$.

Additionally, it is customary to impose that a composition starts with a chord built on a specific degree of the scale d_s and ends with a chord built on a specific degree of the scale d_e , since this is what happens normally (usually

$d_s = d_e = I$). Notice that for each of these situations (each cadence, and the starting and ending of the composition) the splicing system has splicing rules. As a result, the splicing rules can be organized in three sets:

1. **Starting with d_s :** For each $c_i, c_j \in \text{Chords}(\mathcal{G})$, such that $\text{Degree}(c_i) = d_s$, $\mathcal{R}_1$ contains the rule $r = w_i \# \epsilon \# w_j$ where ϵ is the empty word, w_i is the word associated to c_i and w_j is the word associated to c_j . Moreover also w_i is added to $\mathcal{I}_1$.
2. **Cadences:** For each quadruple of chords $c_i, c_j, c_s, c_t \in \text{Chords}(\mathcal{G})$, such that $\text{Degree}(c_1) \rightarrow \text{Degree}(c_4) \in \text{Cadences}$ and $\text{Degree}(c_3) \rightarrow \text{Degree}(c_2) \in \text{Cadences}$, $\mathcal{R}_1$ contains the rule $r = w_1 \# w_2 \# w_3 \# w_4$ where w_i is the word associated to c_i , w_j is the word associated to c_j , w_s is the word associated to c_s , and w_t is the word associated to c_t . Words w_i, w_j, w_s , and w_t are also inserted into $\mathcal{I}_1$.
3. **Ending with d_e :** For each $c_i, c_j \in \text{Chords}(\mathcal{G})$, such that $\text{Degree}(c_j) = d_e$, $\mathcal{R}_1$ contains the rule $r = w_i \# \epsilon \# w_j$ where w_i is the word associated to c_i and w_j is the word associated to c_j . Word w_j is also inserted into $\mathcal{I}_1$.

cq3: evolutionary process. Approach 1 has been described by providing the 3 components of the system $\mathcal{S}_1 = (\mathcal{A}_1, \mathcal{I}_1, \mathcal{R}_1)$. What remains to describe is how the system produces the output composition. To this end, the splicing process is transformed into an evolutionary process, and using the system $\mathcal{S}_1$, through such a process, the language $L = L(\mathcal{S}_1)$ is generated.

- **Evolution:** Each iteration of the evolution corresponds to one application of all the rules in $\mathcal{I}_1$ to all possible pairs of words in the current language. So the language $L(\mathcal{S}_1)$ evolves by acquiring new words at each iteration.
 - **Stop criterion:** Several possibilities can be considered as stop criterion, such as, fixing a maximum number of iterations, or fixing a *quality threshold* that the solutions should reach in order to be able to say that a good quality has been achieved. In [19, 21], a fixed maximum number of iterations has been considered, because the choice of a threshold that expresses the concept of good quality in music is often too tied to subjective evaluations. So, let max be a fixed maximum number of iterations. Let $L_{max}(\mathcal{S}_1)$ be the language obtained after max iterations. Once the language $L_{max}(\mathcal{S}_1)$ has been generated, choose one single word $w \in L_{max}(\mathcal{S}_1)$ as the output of the algorithmic composer. The output of the composer is the composition represented by such a word; the choice of the word is made exploiting an evaluation function.
 - **Evaluation function:** a function f_h measures the harmonic quality of the compositions. The function is defined by considering well-known and widely accepted rules in music theory. Specifically, f_h assigns “weights” to pairs of consecutive chords. As specified at the beginning of the section, this approach uses static weights following common conventions in music, as proposed in [19].
- Thus, given a composition (word) $C = (c_1, \dots, c_n)$, the harmonic value $h(C)$ of C is obtained by giving a weight for $f_h(c_i, c_{i+1})$, with $i = 1, 2, \dots, n-1$.

1. The value of $f_h(C)$ is given by the sum of the weights for each pair (c_i, c_{i+1}) for $i = 1, 2, \dots, n - 1$. A composition with maximum f_h is given in output.

3.3 Implementation

This section describes the result of an execution of the music splicing system built with Approach 1. The systems has been implemented in Python by using the library `music21`¹. All experiments have been conducted with a 2,8 GHz Intel Core i7 quad-core machine equipped with 16 GB 2133 MHz LPDDR3 RAM. The time required to complete all the experiments was approximately 3h 20m.

Evolution experiments. Since 4-voice chorales are widely available from J.S. Bach's production, it is convenient to focus on the case of $k = 4$. The set $\mathcal{I}_1$, is defined considering the *ground set* $\mathcal{G}$ containing 10 J. S. Bach's chorales: BWV 2.6, BWV 10.7, BWV 11.6, BWV 14.5, BWV 16.6, BWV 20.7, BWV 28.6, BWV 32.6, BWV 40.8 and BWV 44.7.

Since the evolutionary process grows the language without bounds, any implementation risks of running out of memory quickly. For this reason it is good to keep bounded the size of the language by setting a maximum threshold p_{max} for the words in the language and exploiting the evaluation function to keep the best words in the language. For this reason, at the end of an iteration, if the cardinality of the generated language is greater than p_{max} , only the p_{max} solutions that have the highest harmonic value are kept in the set. Setting a limit to the size of the language is necessary because otherwise the language grows without bounds and this clearly will lead, soon or later, to an out-of-memory error.

Several experiments have been executed, using a different number t of iterations and several values for the maximum size p_{max} of the generated language. More precisely, the values for t are the ones in the set $T = \{50, 100, 250, 500, 750, 1000, 5000, 7500, 10000\}$, and values for p_{max} are the ones in the set $P = \{50, 100, 250, 500, 750, 1000\}$.

For each pair (t, p_{max}) , 5 executions of $\mathcal{S}_1$ have been run. For each experiment the *average* harmonic value of the 5 executions has been computed. The best result has been obtained with $t = 5000$ and $p_{max} = 1000$. Observe that the process is deterministic. Obviously, by changing the t and p_{max} parameters it is possible to obtain different solutions. A degree of non-determinism is present if, at the end of an iteration, there are many words with the same harmonic value and, in order to limit the size of the language to p_{max} , some of them have to be kept and other discarded. Thus, also the choice of which ones are kept in case of ties can also influence the results obtained. The implementation described here makes random choices in such cases.

¹ <http://web.mit.edu/music21/>.

Figure 2 shows the music score of the best solution obtained. Notice that when a k -voice-like composition 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, it is needed to set some rhythmic parameters. Specifically, the *meter* of the composition and the duration of each note. As done in [19,21] the implementation used here (i) *sets the meter value* by choosing randomly one value in a set of typical meter values, *sets the duration of the notes*, by adding, with a uniform distribution of probabilities, non-harmonic tones after the notes, ensuring no alteration in the total original duration. Figure 2 shows the music obtained.

4 Approach 2: Splicing systems based on *statistical weights*

The approach described in this section is an enhancement of the one presented in the previous section. With respect to Approach 1, Approach 2 uses an *hybrid* evaluation function that, on one hand, adheres to specific music rules, and on the other hand, extracts statistical information from a corpus of existing music compositions, with the goal of assimilating a specific composer’s style or genre’s style. The approach is applied to the same music composition problem that has been considered in Section 3, that is the k -voice music composition problem.

4.1 The music composition problem: k -voices music

The problem is the same as the one considered in Section 3, so the reader is referred to that section for details about the problem.

4.2 Approach details

As for the previous approach the music splicing system $\mathcal{S}_2$ is described by instantiating the three components $\mathcal{S}_2 = (\mathcal{A}_2, \mathcal{I}_2, \mathcal{R}_2)$, that is the alphabet, the initial set of words and the rules, and explaining how this specific approach tackles the 3 composition questions.

cq1: *word-based representation*. The word-based representation used in this approach is an extension of the word-based representation described in Section 3. The main difference lies in the additional information considered to represent each note. Specifically, for each note the information considered is the one used in Section 3 (i.e., the *voice* that plays such a note, the *name* of such a note, and the *octave* in which it is located) and, on top of that, also: (i) the *tonality* of the chord in which the note is played, (ii) the *quality* of the chord in which the note is played, and (iii) the *degree* of the chord in which the note is played. More precisely, a chord c_i is represented as $w_i = t_1 q_1 d_1 v_1 x_1 o_1 t_1 q_1 d_1 \cdots v_k x_k o_k t_k q_k d_k v_k x_k o_k$, where, for each $j = 1, \dots, k$:

APPROACH 1

$\text{♩} = 100$

VOICE 4

VOICE 3

VOICE 2

VOICE 1

VOICE 4

VOICE 3

VOICE 2

VOICE 1

VOICE 4

VOICE 3

VOICE 2

VOICE 1

18

VOICE 4

VOICE 3

VOICE 2

VOICE 1

Fig. 2: Approach 1: The composition given in output by S_1 .

- v_j is the voice which plays the j -th note in c_i ,
- x_j is the name of the j -th note in c_i ,
- o_j is the octave in which the j -th note in c_i is placed.
- t_j is the tonality of c_i ,
- q_j is the quality of c_i ,
- d_j is the degree of c_i .

Observe that, given the j -th note in the chord c_i , the information regarding tonality (t_j), quality (q_j) and degree (d_j), is concatenated twice, before and after the information regarding voice (v_j), name (x_j) and octave (o_j), resulting in the word $t_j q_j d_j v_j x_j o_j t_j q_j d_j$. As shown in [21], this choice is crucial for the definition of the rules.

For the alphabet, let $\mathcal{A}_V$ be the voice alphabet, $\mathcal{A}_N$ be the note alphabet, and $\mathcal{A}_O$ be the octave alphabet, defined in Section 3. Then, let the *tonality alphabet* $\mathcal{A}_T$ be $\{C, C\#, D\flat, D, D\#, E\flat, E, F, F\#, G\flat, G, G\#, A\flat, A, A\#, B\flat, B\}$, the *quality alphabet* $\mathcal{A}_Q$ be $\{M, m\}$, where M stands for major tonality and m for minor tonality, and the degree alphabet $\mathcal{A}_D$ be $\{1, 2, 3, 4, 5, 6, 7\}$.

Finally, let the alphabet $\mathcal{A}_2$ be $\mathcal{A}_2 = \mathcal{A}_V \cup \mathcal{A}_N \cup \mathcal{A}_O \cup \mathcal{A}_T \cup \mathcal{A}_D \cup \mathcal{A}_Q$.

Example 2 Consider again the 4-voice music fragment C in Figure 1, $C = (c_1, c_2, c_3, c_4)$. Each chord c_i is represented by a word w_i . Specifically: $w_1 = \text{“}\mathbf{BbM5} \ v_1 \ A4 \ v_2 \ C5 \ v_3 \ F5 \ v_4 \ C6 \ \mathbf{BbM5}\text{”}$, $w_2 = \text{“}\mathbf{BbM5} \ v_1 \ F4 \ v_2 \ F5 \ v_3 \ A5 \ v_4 \ C6 \ \mathbf{BbM5}\text{”}$, $w_3 = \text{“}\mathbf{BbM1} \ v_1 \ Bb4 \ v_2 \ F5 \ v_3 \ Bb5 \ v_4 \ D6 \ \mathbf{BbM1}\text{”}$, $w_4 = \text{“}\mathbf{BbM6} \ v_1 \ G4 \ v_2 \ G5 \ v_3 \ D5 \ v_4 \ Bb5 \ \mathbf{BbM6}\text{”}$, so $w = \mathbf{BbM5} \ v_1 \ A4 \ v_2 \ C5 \ v_3 \ F5 \ v_4 \ C6 \ \mathbf{BbM5} \ \mathbf{BbM5} \ v_1 \ F4 \ v_2 \ F5 \ v_3 \ A5 \ v_4 \ C6 \ \mathbf{BbM5} \ \mathbf{BbM1} \ v_1 \ Bb4 \ v_2 \ F5 \ v_3 \ Bb5 \ v_4 \ D6 \ \mathbf{BbM1} \ \mathbf{BbM6} \ v_1 \ G4 \ v_2 \ G5 \ v_3 \ D5 \ v_4 \ Bb5 \ \mathbf{BbM6}\text{”}$.

Notice that boldface text is used only to emphasize the novel music information embedded in the words (tonality, degree and quality), with respect to that defined in Section 3.

cq2: rules definition. Now the rules. The initial set of words $\mathcal{I}_2$ is defined as in the previous approach; the *ground data set* $\mathcal{G}$, is obtained by considering all the chosen pieces and their transpositions in all the tonalities, which gives $|\mathcal{G}| = 12 * t$ pieces, where t is the number of chosen pieces. The set $\mathcal{I}_2$ consists of the $12 * t$ words that represent the pieces in $\mathcal{G}$, by means of the word-based representation defined above and of some other words that will be explained shortly. Differently from the approach proposed in Section 3, in which the rules are built by considering all the combinations of the extracted chords, in this approach the degree and the tonality of the chord are integrated directly in the word representation of each chord. As a consequence, it is not necessary to transpose each chord in all 12 tonalities since to define a rule it suffices to use extracted chords having specific tonalities and degrees. This is an advantage because it guarantees a considerably smaller number of rules. In addition, with such a representation it is possible to directly extract the rules for the modulations. Thus, given an initial set of chorales $\mathcal{G}$, and the set $Chords(\mathcal{G})$ of chords extracted from $\mathcal{G}$, the time and space complexities of the music splicing system proposed in this approach, are linear in $|Chords(\mathcal{G})|$, i.e., $\mathcal{O}(|Chords(\mathcal{G})|)$.

The rules in $\mathcal{R}_2$ are partitioned in four sets:

1. **Group 1 (forcing d_s to start).** For $c_i, c_j \in \text{Chords}(\mathcal{G})$, satisfying $\text{Degree}(c_i) = d_s$ and $\text{Tonality}(c_i) = \text{Tonality}(c_j)$, $\mathcal{R}_2$ contains the rule $r = w_i | \epsilon \$ \epsilon | w_j$ where, w_i and w_j are the word associated to c_i and c_j , respectively. The word w_i is also added to $\mathcal{I}_2$.
2. **Group 2 (forcing cadences).** For each quadruple of chords $c_i, c_j, c_s, c_t \in \text{Chords}(\mathcal{G})$, if $\text{Degree}(c_i) \rightarrow \text{Degree}(c_t)$, with $\text{Tonality}(c_i) = \text{Tonality}(c_t)$ is in *Cadences*, and $\text{Degree}(c_s) \rightarrow \text{Degree}(c_j)$, $\text{Tonality}(c_s) = \text{Tonality}(c_j)$, is also in *Cadences*, $\mathcal{R}_2$ contains the rule $r = w_i | w_j \$ w_s | w_t$, where w_i is the word associated to c_i , w_j is the word associated to c_j , w_s is the word associated to c_s , and w_t is the word associated to c_t . The words w_i, w_j, w_s , and w_t are also added to $\mathcal{I}_2$.
3. **Group 3 (forcing d_e as ending).** For $c_i, c_j \in \text{Chords}(\mathcal{G})$, satisfying $\text{Degree}(c_j) = d_e$ and $\text{Tonality}(c_i) = \text{Tonality}(c_j)$, $\mathcal{R}_2$ contains the rule $r = w_i | \epsilon \$ \epsilon | w_j$ where w_i and w_j are the words associated to c_i and c_j , respectively. The word w_j is also inserted into $\mathcal{I}_2$.
4. **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, $\mathcal{R}_2$ contains the rule $r = w_i | w_{i+1} \$ w_j | w_{j+1}$ where w_k is the word associated with c_k , for $k = i, i+1, j, j+1$. The words w_i, w_{i+1}, w_j , and w_{j+1} are also added to $\mathcal{I}_2$.

cq3: evolutionary process. Having defined the system $\mathcal{S}_2 = (\mathcal{A}_2, \mathcal{I}_2, \mathcal{R}_2)$ what remains to do is to describe the evolutionary process that generates the language $L = L(\mathcal{S}_2)$. As for the system described in the previous section, there is an evolution process, a stop criterion and an evaluation function. Actually the first two are the same as the ones already seen in Section 3. The evaluation function is different.

- **Evolution process:** As before, in each iteration, the rules $\mathcal{R}_2$ are applied to the current language.
- **Stop criterion:** As before, there is a fixed maximum number of iterations max and the evolution process is repeated for such a number of iterations to obtain $L_{max}(\mathcal{S}_2)$. From this language, one single word is chosen as the output composition.
- **Evaluation function:** The evaluation function is used to select the single word from $L_{max}(\mathcal{S}_2)$. This approach is different because it defines an evaluation function that, (i) on one hand adheres to rules from classical music, and (ii) on the other hand exploits *statistical information* from a corpus of existing music, which is, somehow, representative of the specific musical style (and genre) that is being considering. For example, in [21], statistical information is extracted from a set of Bach's chorales. The result is a *multi-objective evaluation function* composed of two sub-functions: an *harmonic function* f_h and a *melodic function* f_m . Both functions have the

following form:

$$f = \sum_i a_i w_i$$

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

Intuitively, the weights w_i are used to express the objective part of the evaluation and represent well known rules from the theory of harmony. The coefficients, instead, are used to express a subjective component, and are obtained with a statistical analysis of the chosen corpus of existing music (capture the style of the music pieces in the corpus chosen). The interested reader can find further details about the weights and the coefficients in [21].

Harmonic function. Similarly to what was done in in Section 3, for the unique evaluation function f , given a k -voice composition $C = (c_1, \dots, c_n)$, the *harmonic value* $f_h(C)$ is calculated by considering all pairs of consecutive chords c_i, c_{i+1} . The objective is to maximize $f_h(C)$. However, in this approach the f_h definition is more complex with respect to that proposed in Section 3: the weights are not statically fixed, but are the result of transforming musical conventions into a probability distribution. Furthermore, they are multiplied by coefficients obtained with a statistical analysis of the chosen corpus.

Regarding the *coefficients*, they have been obtained by performing a statistical analysis over the chosen corpus. Specifically, by looking for adjacent chords it is possible to figure out the percentage of passages from one chord to the subsequent one. As an example, see Table 6 and Table 7 in [21], which show the result obtained by performing a statistical analysis on a subset of the J.S.Bach's chorales corpus.

For what concerns the *weights*, instead, usually in classical harmony the *frequency distribution* of usage for the cadences, is organized in classes "often", "sometimes", and "seldom" (see and [21] - Tables 2 and 3, for further details). As an example, in classical music, the degree I (first major degree):

- "often" goes to I, IV and V ,
- "sometimes" goes to vi ,
- "seldom" goes to ii and iii .

Suppose to have a corresponding *probability distribution*, i.e., $(X_{often}, X_{sometimes}, X_{seldom})$. Then, 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, it is possible to use the probability distribution $(X_{often}, X_{sometimes}, X_{seldom})$ to assign the weight as follows: c_{i+1} will be one of the chords in the "often" class, X_{often} percent of the times, one of the chords in the "sometimes" class, $X_{sometimes}$ percent of the times, and one of the chord in the "seldom" class, X_{seldom} percent of the times. Obviously, such a probability distribution can be defined in several ways. As an example, in [21], the distribution chosen after a statistical evaluation) of Bach's chorales was $(X_{often}, X_{sometimes}, X_{seldom}) = (80, 15, 5)$. Finally, to compute the weights, given the degrees of two chords c_i and c_{i+1} (for example: I for c_i and V for c_{i+1}):

1. Select the value X in the probability distribution (X_{often} , $X_{sometimes}$, X_{seldom}) by searching the passage $c_i \rightarrow c_{i+1}$ in the frequency distribution (for example: I “often” goes to V , and so $X = X_{often}$).
2. Count the number N of degrees which are the X frequency (for example: there are $N = 3$ degrees in the “often” frequency of I , i.e., I, IV , and V).
3. Calculate the weight as $\frac{X/100}{N}$ to obtain a value in percentage (for example: if $X_{often} = 80$ and $N = 3$, then the weight is $\frac{0.8}{3} = 0.26$).

Notice that the argument explained above regards passages of chords in the same tonality (major or minor). However, the same argument can be applied to passages of chords in different tonalities, named *modulations* in music. As an example, see Table 8 in [21].

Melodic function. The melodic function f_m evaluates the melodic quality of a chorale C and is defined as $f_m(C) = \sum_i a_i w_i$, where the index i runs over all “errors”. The objective is to minimize $f_m(C)$. Such errors can be found by performing an “exception analysis” to identify stylistic anomalies and formal 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 formal problem that should be corrected. There exist two exception classes: *motion exceptions* and *voicing exceptions*. Examples of motion exceptions are parallel octaves and parallel fifths. Examples of voice exceptions are voice jumps, voice crossings (see Table 9 in [21]).

Once defined the exceptions, the idea is to assign to each of them a coefficient and a weight. The coefficients can be obtained with a statistical analysis of a reference corpus of music, as done for the harmonic function. The weights can be assigned according to the severity level of the exception. As an example, in classical music the parallel fifths exception has a high value of severity, while the voice jump has a lower value of severity. As an example, in Table 9 of [21], the weight 2 is assigned to parallel fifths exceptions, and the weight 1 is assigned to voice jump exceptions. Furthermore, again in such a table, the statistical coefficient calculated for the parallel fifths exception is 0.05, while that for voice jump exception is 6.90.

4.3 Implementation

This section reports the result of an experiment with an implementation of the described approach. The details regarding the language used to implement the system, and the hardware platform used to conduct the experiments are the same as in Section 3.3. The time required to complete all the experiments was slightly longer, i.e., approximately 4h 15m.

Evolution experiments. As for the Approach 1, 4-voices chorales were considered and the same set of Bach’s chorales has been used as the ground set $\mathcal{G}$:

BWV 2.6, BWV 10.7, BWV 11.6, BWV 14.5, BWV 16.6, BWV 20.7, BWV 28.6, BWV 32.6, BWV 40.8 and BWV 44.7. And, as before, each one has been transposed in every tonality. The set $\mathcal{I}_2$ consists of the words corresponding to each chorale and to each transposition.

As for the experiment in the previous section, a set of values for the maximum number t of iterations and for the maximum size p_{max} of the language has been considered, namely $t \in T = \{50, 100, 250, 500, 750, 1000, 5000, 7500, 10000\}$, and $p_{max} \in P = \{50, 100, 250, 500, 750, 1000\}$.

Also in this case for each pair (t, p_{max}) 5 executions of $\mathcal{S}_2$ were run. Notice that, differently from the approach described in Section 3.3, in this case there is a multi-objective problem, i.e., it is necessary to look for solutions that simultaneously maximize the harmonic value and minimize the melodic value. To this, at the end of each iteration only solutions in the Pareto front are considered for the next iteration. After the last iteration, the output solution is chosen from the Pareto front selecting the one with the best harmonic value. For each experiment both the average harmonic value the average melodic value of the 5 runs have been computed.

The best output was obtained for $t = 7500$ and $p_{max} = 1000$. Figure 3 shows the music obtained. Notice that, as for Approach 2, the final score has been obtained by applying the same rhythmic operator used in Section 3.3 and defined in [19, 21].

5 Approach 3: evaluation based on the *prediction of stylistic patterns*

This section presents a music splicing system based on an evaluation function with a completely different approach: it measures the quality of the composition by using a method to predict patterns coherent with a specific style; the patterns are used to guide the splicing system during the composition. For this approach, the specific music problem that is considered is that of recognizing a music style and/or to compose music in a specific style. For this approach, beside the music splicing system, two additional components, called *recognizer* and *predictor*, are needed.

5.1 The music composition problem: music style recognition and composition

The problem considered is the problem of both recognize and compose music for a specific performer's style. To address such a problem, the approach exploits: (i) a *recognizer*, i.e., a machine learning-based classifier to learn a specific music performer's style, (ii) a *composer*, i.e., a music splicing system to compose melodic lines in the style learned by the recognizer, and (iii) a *predictor*, i.e., a machine learning method to predict patterns coherent with the style learned by the recognizer, and used by the composer to evaluate the "stylistic" goodness of the composed music pieces.

APPROACH 2

$\text{♩} = 50$

The musical score consists of four staves, each labeled on the left as VOICE 4, VOICE 3, VOICE 2, and VOICE 1 from top to bottom. The music is written in 4/4 time with a key signature of two flats. A tempo marking $\text{♩} = 50$ is placed above the first staff. The score is divided into three systems of four measures each. The first system (measures 1-4) shows the initial entry of the voices. The second system (measures 5-8) continues the development. The third system (measures 9-12) concludes the piece with double bar lines.

Fig. 3: Approach 2: The composition given in output by S_2 .

The composer is defined by an initial set of melodies coherent with a specific style, and a set of rules built by using the predictor. The goal is to generate a language containing words that represent pieces of “new” melodies coherent with the chosen style. The chosen music style is denoted, generically, with “STYLE”. When the composer generates a composition, it uses the predictor to verify that the musical patterns within such a composition are actually similar to the expected patterns, i.e., the patterns predicted by the predictor. The best solution is the one that contains the largest number of relevant patterns expected by the predictor.

5.2 The Recognizer

The *recognizer* $\mathcal{R}ec$ is a machine learning-based method able to recognize a specific style. As will be discussed in Section 5.3, $\mathcal{R}ec$ is necessary to build the predictor used by the composer to evaluate the compositions generated.

Suppose that $\mathcal{R}ec$ needs to be trained to recognizing the style `STYLE`. Formally, given a melody m , the objective of $\mathcal{R}ec$ is that of classifying m in two possible classes: *coherent* with `STYLE` or *not coherent* with it. To achieve this $\mathcal{R}ec$ is trained on a corpus of music pieces $\mathcal{M}$, containing solos performed in the style `STYLE`. In order to train $\mathcal{R}ec$ in an effective way, it is necessary to decide what information, or *features*, of the melodies in the training set will be useful to $\mathcal{R}ec$ in understanding the style. Let v_j be the feature vector containing such significant features of some melody m_j , obtained through the feature extraction model to be discussed shortly. The recognizer classifies m_j by using v_j as input.

Feature Model. To capture the most significant features of a melody the n -gram model [28] is used. Such a model identifies *tokens* in melodies whose importance can be determined through some *statistical measure*. In this approach a token is the set of relevant information about a music note. Thus an n -gram represents a sequence of n notes. Once n -grams have been constructed by the training set of melodies, a statistical measure is used to calculate their relative importance. The result is a list of n -grams, ordered according to their importance.

- **The token.** A music note is described by a token with three parameters (k_1 , k_2 , and k_3). Such parameters are referred to as the *token’s parameters*:
 - k_1 indicates the *name chord*,
 - k_2 refers to the *type chord*,
 - k_3 indicates the *role* of the note respect to the chord.

The chords considered are those derived from the modes of the major, melodic minor and harmonic minor scales (see [27] for further details). Table 1 in [20] provides a description of such type of chords. To summarize, the β -tuple $K^i = [k_1^i, k_2^i, k_3^i]$ is the token that describes the music note played at i^{th} time interval. For example $K^5 = [2, 4, 4]$ says that at the 5^{th}

time interval the music note played has degree *III* ($k_3 = 4$ means degree *III*) in the scale corresponding to the chord *D7* ($k_1 = 9$ means name chord *D* and $k_2 = 4$ means type chord 7). Thus the note played is *F#*.

- **The statistical measure.** The statistical measure proposed to evaluate the relative importance of n -grams is the *term frequency with inverse document frequency* (*tfidf*). The idea is to use the *tfidf* measure: (i) for giving more weight to terms that are less common in $\mathcal{M}$, since such terms are more likely to make the corresponding melody stand out, and (ii) for transforming the corpus $\mathcal{M}$ to a feature vector space.

The following method comes from [20]. First, each n -gram, i.e., a sequence of n tokens which corresponds to a sequence of n music notes, is a term t . Then, the *boolean term frequency* (*tf*) and the *inverse document frequency* (*idf*) can be defined as:

- $tf(t, m_j) = 1$ if $t \in m_j$ (the sequence of notes corresponding to t occurs in the melody m_j), 0 otherwise,
- $idf(t, m_j, \mathcal{M}) = \log(|\mathcal{M}| / |\{ m_j \in \mathcal{M} : t \in m_j \}|)$.

Finally, the feature vector v_j corresponding to a melody m_j is built as follows: the i -th component of v_j , denoted with $v_j[i]$ is equal to $tfidf(t_i, m_j, \mathcal{M})$, where $tfidf(t, m_j, \mathcal{M}) = tf(t, m_j) \cdot idf(t, m_j, \mathcal{M})$.

The classifier. Let m_j be a melody and v_j be its corresponding feature vector. Such a vector contains an element for each significant feature of the melody, and the value of this element is the *tfidf*. Several machine learning models can be trained to perform this task. In [20] a One-Class Support Vector Machine has been used.

5.3 The Predictor

The *predictor* $\mathcal{Pre}$ is a machine learning method to predict patterns coherent with the style learned by $\mathcal{Rec}$. The predictor $\mathcal{Pre}$ is used by the music splicing composer defined in the following, to evaluate the “stylistic” goodness of the composed music pieces.

Let $\mathcal{M}$ be the chosen set of solos, and n be the value used for the construction of the n -grams as described in Section 5.2. Then, the idea is to define a machine learning predictor $\mathcal{Pre}$ that given an n -gram at time i has to predict the n -gram at time $t + 1$. This is equivalent to saying that given the sequence of n music notes at time i , $\mathcal{Pre}$ has to predict the sequence of n music notes at time $i + 1$.

The training set. The data set of n -grams is defined as follows. Let $\mathcal{T} \subseteq \mathcal{M}$ be the training set used for the training of $\mathcal{Rec}$. For each $m_j \in \mathcal{T}$ such that $\mathcal{Rec}$ says that m_j is coherent with the style style ($f(m_j) < 0$), consider the sequence of n -grams extract by m_j . Let $Ngrams(m_j) = (ng_1, \dots, ng_{k_j})$ be the sequence of n -grams extract from m_j , and insert the pair (n_i, n_{i+1}) in the training set for $\mathcal{Pre}$, for each $1 \leq i \leq k_j - 1$.

The architecture. Several machine learning models can be trained on the set $\mathcal{T}$ above described. In [20] an Long short-term memory (LSTM) network has been used.

5.4 The Splicing System: approach details

This section describes the splicing system $\mathcal{S}_3 = (\mathcal{A}_3, \mathcal{I}_3, \mathcal{R}_3)$, which is the core of Approach 3. In the following, details about $\mathcal{A}_3, \mathcal{I}_3, \mathcal{R}_3$ are provided together with a discussion about the composition questions.

cq1: word-based representation. A suitable alphabet $\mathcal{A}_3$ is needed. Let $\mathcal{A}_3$ be $\mathcal{A}_3 = \mathcal{A}_N \cup \mathcal{A}_T \cup \mathcal{A}_D \cup \mathcal{A}_S$ where $\mathcal{A}_N$ is the *chord name alphabet*, $\mathcal{A}_T$ is the *chord type alphabet*, $\mathcal{A}_D$ is *note degree alphabet* and $\mathcal{A}_S$ is the *separator alphabet*. Specifically, $\mathcal{A}_N = \mathcal{A}_D = \{0, \dots, 11\}$, $\mathcal{A}_T = \{0, \dots, 20\}$ and $\mathcal{A}_S = \{\tau, \mu_N, \mu_T, \mu_D\}$. Alphabet $\mathcal{A}_3$ is used to represent solos (melodies) as words.

As explained in Section 5.2 each token represents a music note. Thus, given a melody $m = (n_1, \dots, n_l)$, for each note n_i , the token K^i is represented as a word w_i over $\mathcal{A}_3$. Specifically $w_i = \mu_N x_i \mu_T y_i \mu_D v_i$, where $x_i \in \mathcal{A}_N$, $y_i \in \mathcal{A}_T$ and $v_i \in \mathcal{A}_D$, for each $1 \leq i \leq l$. Thus, m is represented by the word $\mathcal{W}(m) = \tau w_1 \tau w_2 \tau \dots \tau w_n \tau$.

Example 3 Consider the melody m , shown in Fig. 4 (first 2 measures of the standard Jazz “Now’s the time” of Charlie Parker), and consider the chords organization proposed in [20] (see Table 1). For this melody $m = (n_1, \dots, n_{12})$. Each n_i is played in F7 chord. The chord name F has value 5. Chord F7 is the associated scale is the mixolydian mode of the major scale, thus the chord type has value 4. Now, let us analyze the role of notes respect the F7: $n_1 = n_5 = n_7 = n_{11} = C$ which has degree v (position value 7), $n_2 = n_3 = n_6 = n_8 = n_9 = n_{12} = F$ which has degree i (position value 0), and $n_4 = n_{10} = G$ which has degree ii (position value 2). Thus, there are 10 tokens: $K^1 = K^5 = K^7 = K^{11} = (4, 5, 0)$, $K^2 = K^3 = K^6 = K^8 = K^9 = K^{12} = (4, 5, 7)$, and $K^4 = K^{10} = (4, 5, 2)$. So, $\mathcal{W}(m) = \tau w_1 \tau \dots \tau w_{12} \tau$ and $w_1 = \mu_N 4 \mu_T 5 \mu_D 0$, $w_2 = \mu_N 4 \mu_T 5 \mu_D 7$, $w_3 = \mu_N 4 \mu_T 5 \mu_D 7$, $w_4 = \mu_N 4 \mu_T 5 \mu_D 2$, $w_5 = \mu_N 4 \mu_T 5 \mu_D 0$, $w_6 = \mu_N 4 \mu_T 5 \mu_D 7$, $w_7 = \mu_N 4 \mu_T 5 \mu_D 0$, $w_8 = \mu_N 4 \mu_T 5 \mu_D 7$, $w_9 = \mu_N 4 \mu_T 5 \mu_D 7$, $w_{10} = \mu_N 4 \mu_T 5 \mu_D 2$, $w_{11} = \mu_N 4 \mu_T 5 \mu_D 0$, and $w_{12} = \mu_N 4 \mu_T 5 \mu_D 7$.

Fig. 4: First 2 measures of Charlie Parker’s Now’s the time.

cq2: rules definition. The idea is to start from an initial set of melodies known to be coherent with the style `STYLE`. Consider the following set of n -grams (the n used by `Rec`): let $\mathcal{T}$ be the training set used for the training of `Rec`. For each $m_j \in \mathcal{T}$ such that `Rec` says that m_j is coherent with the style `STYLE` ($f(m_j) < 0$), consider the list $Ngrams(m_j)$ of n -grams extracted by m_j . Let $Ngrams(m_j) = (ng_1, \dots, ng_{k_j})$, and insert $\mathcal{W}(ng_i)$ in the set $\mathcal{I}_3$, for each $1 \leq i \leq k_j - 1$.

Unlike the problem considered in Sections 3 and 4, in which it was possible to model rules by using music theory, in this case there are no such rules (composing melodies coherent with a style is not coded through rules). Here the predictor `Pre` as needed and is used as follows.

1. *Group 1 (forcing sequence n-grams).* Let $w_i \in \mathcal{I}_3$ and n_i such that $w_i = \mathcal{W}(n_i)$; the rule is defined as $r = w_i | \epsilon \$ \epsilon | \mathcal{W}(\text{Pre}(n_i))$ where, $\text{Pre}(n_i)$ is the n -gram predicted by `Pre` with n_i as input. These rules force the composer to paste words corresponding to n -grams that really follows n_i in the training set $\mathcal{T}$.
2. *Group 2 (forcing sequence tokens).* Given $w_i, w_j \in \mathcal{I}_3$, n_i such that $w_i = \mathcal{W}(n_i)$, n_j such that $w_j = \mathcal{W}(n_j)$, where $n_i = (k_{i,1}, \dots, k_{i,n})$, $n_j = (k_{j,1}, \dots, k_{j,n})$ and there exist l_1, l_2 such that $k_{i,l_1} = k_{j,l_2}$. Then, the rule is defined as $r = \mathcal{W}(k_{i,1}, \dots, k_{i,l_1-1}) | \epsilon \$ \epsilon | \mathcal{W}(k_{j,l_2}, \dots, k_{j,n_j})$. If n_i and n_j share a token, then these rules force the composer to cut w_i and w_j in correspondence of such a token, and paste the remaining words.

cq3: evolutionary process. Also in this approach a maximum number of iterations is used as stop criterion. As for the definition of the above rules, also for the definition of the function f_e to evaluate the melodies produced, there are no music rules that can be used. So the idea is to explout the predictor `Pre` as follows. Let w be a word generated by the composer $\mathcal{S}_3$. Let $m = (n_1, \dots, n_l)$ the melody such that $w = \mathcal{W}(m)$. Now, let $Ngrams(m) = (ng_1, \dots, ng_{l-1})$ be the sequence of n -grams extracted from m . The function f_e is defined as:

$$f_e(m) = \sum_{1 \leq i \leq l-1} (tfidf(ng_i, m, \mathcal{M})) + Diff(ng_{i+1}, \text{Pre}(ng_i))$$

where $Diff(ng_i, \text{Pre}(ng_i))$ is the *difference* between ng_{i+1} and $\text{Pre}(ng_i)$ that is the n -gram predicted by `Pre` with ng_i as input. Such a difference is defined as follows: let $K^{i+1} = [k_1^{i+1}, k_2^{i+1}, k_3^{i+1}]$ be the token for ng_{i+1} and $K'^{i+1} = [k_1'^{i+1}, k_2'^{i+1}, k_3'^{i+1}]$ be the token for $\text{Pre}(ng_i)$. Then $Diff(ng_{i+1}, \text{Pre}(ng_i)) = |k_1^{i+1} - k_1'^{i+1}| + |k_2^{i+1} - k_2'^{i+1}| + |k_3^{i+1} - k_3'^{i+1}|$.

5.5 Implementation

This section describes an implementation of a music splicing system based on Approach 3. As explained in Section 5.2 and Section 5.3, in this case it is necessary to implement a recognizer `Rec` to learn a specific music performer's

style, and a predictor $\mathcal{Pre}$ to predict patterns coherent with the learned style and used to guide the splicing composition. Following [20], the implementation described here uses a One-Class Support Vector Machine to implement $\mathcal{Rec}$, and a Long short-term memory network to implement $\mathcal{Pre}$.

The details regarding the language used to implement the music splicing system, and the hardware platform used to conduct the experiments are the same as in Section 3.3. However, in this case, the recognizer $\mathcal{Rec}$ and the predictor $\mathcal{Pre}$ have been implemented using the Python library scikit-learn². The time required to complete all the experiments was approximately 4h 5m.

Evolution experiments. The focus is on Jazz music, that is `STYLE=Jazz`. The data set $\mathcal{M}$ consisted of Jazz solos, transcribed in MusicXML format, from one of the most popular Jazz musicians: Louis Armstrong. Specifically, $|\mathcal{M}| = 50$.

As for the previous cases, the experiments have been run using a number t of iterations in the set $T = \{50, 100, 250, 500, 750, 1000, 5000, 7500, 10000\}$, and the a max size p_{max} in the set $P = \{50, 100, 250, 500, 750, 1000\}$.

For each pair (t, p_{max}) 5 executions have been run. For each experiment the average f_e function value described in Section 5.4 has been computed. Notice that the function f_e evaluates a composition in terms of “stylistic” goodness. The best solution has been obtained by setting $t = 500$ and $p_{max} = 1000$.

An example of music composition generated. Observe that, by definition, when a word w is generated it only represents a sequence of music notes $m = (n_1, \dots, n_k)$. A rhythmic structure can be added to m by defining an operator that: (i) applies *rhythmic transformations* to m , (ii) *modify the duration* of the notes, (iii) *create rest notes*, (iv) *tie notes* together, and (v) *create triplet notes*. Each of these operation is performed with a uniform distribution of probabilities. In Figure 5 shows the music score of the best solution after applying such operator, as defined in [20].

6 Conclusions

Starting from the famous Adleman experiment, DNA Computing has aroused more and more interest both in the *application field*, coming to be used in various contexts, and in the *theoretical field*, in which researchers have borrowed tools from various areas of theoretical computer science, such as formal language theory, coding theory, and combinatorics on words. Recently, several works, appeared in the literature, have used DNA Computing for reproducing human beings’ creativity and in particular music creativity. In this work, the use of splicing systems, as a tool for automatic music composition has been investigated, for the first time, identifying the crucial points behind the problem and providing in-depth details about how the proposed approaches tackle the problem.

² <https://scikit-learn.org/stable/>

APPROACH 3

Fig. 5: Approach 3: The composition given in output.

Declarations

Funding Not applicable.

Conflicts of interest The authors declare that there is no conflict of interest regarding the publication of this paper.

Availability of data and material Examples of music compositions produced during the experiments can be found online.³

Code availability Not applicable.

References

1. E. Braund, E.R. Miranda, in *Proceedings of the 9th Conference on Interdisciplinary Musicology—CIM14. Berlin, Germany* (Citeseer, 2014)
2. T. Toffoli, N. Margolus, Programmable matter: Concepts and realization, *Physica D: Nonlinear Phenomena* **47**(1), 263 (1991). DOI [https://doi.org/10.1016/0167-2789\(91\)90296-L](https://doi.org/10.1016/0167-2789(91)90296-L)
3. Artificial intelligence: an empirical science, *Artificial Intelligence* **77**(1), 95 (1995)
4. A.M. Turing, *Mechanical intelligence* (North-Holland, 1992)
5. A. Adamatzky, *Physarum machines: computers from slime mould*, vol. 74 (World Scientific, 2010)
6. I. Xenakis, *Formalized music: thought and mathematics in composition*. 6 (Pendragon Press, 1992)

³ <https://cutt.ly/Tbke2EP>

7. E.R. Miranda, Granular synthesis of sounds by means of a cellular automaton, *Leonardo* **28**(4), 297 (1995)
8. E.R. Miranda, Cellular automata music: An interdisciplinary project, *Journal of New Music Research* **22**(1), 3 (1993)
9. E.R. Miranda, J.A. Biles, *Evolutionary Computer Music* (Springer-Verlag, Berlin, Heidelberg, 2007)
10. J.A. Biles, *Creative Evolutionary Systems* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002), chap. 8, pp. 165–187. (Title of chapter: GenJam: Evolution of a Jazz Improviser)
11. J. Jeong, Y. Kim, C.W. Ahn, A multi-objective evolutionary approach to automatic melody generation, *Expert Syst. Appl.* **90**(C), 50 (2017)
12. C.H. Liu, C.K. Ting, in *2015 IEEE Congress on Evolutionary Computation (CEC)* (2015), pp. 2145–2152
13. R.D. Prisco, G. Zaccagnino, R. Zaccagnino, Evocomposer: An evolutionary algorithm for 4-voice music compositions, *Evol. Comput.* **28**(3), 489 (2020)
14. L.M. Adleman, Molecular computation of solutions to combinatorial problems, *Science* **266**(5187), 1021 (1994)
15. E.R. Miranda, L. Bull, F. Gueguen, I.S. Uroukov, Computer music meets unconventional computing: towards sound synthesis with in vitro neuronal networks, *Computer Music Journal* **33**(1), 9 (2009)
16. E.R. Miranda, Harnessing the intelligence of physarum polycephalum for unconventional computing-aided musical composition., *Int. J. Unconv. Comput.* **10**(3), 251 (2014)
17. E. Braund, E.R. Miranda, in *Proceedings of 10th International Symposium on Computer Music Multidisciplinary Research (CMMR): Sound Music and Motion* (2013)
18. T. Head, Formal language theory and dna: an analysis of the generative capacity of specific recombinant behaviors, *Bulletin of mathematical biology* **49**(6), 737 (1987)
19. C. De Felice, R. De Prisco, D. Malandrino, G. Zaccagnino, R. Zaccagnino, R. Zizza, in *International Conference on Evolutionary and Biologically Inspired Music and Art* (Springer, 2015), pp. 50–61
20. R. De Prisco, D. Malandrino, G. Zaccagnino, R. Zaccagnino, R. Zizza, in *International Conference on Evolutionary and Biologically Inspired Music and Art* (Springer, 2017), pp. 97–113
21. C. De Felice, R. De Prisco, D. Malandrino, G. Zaccagnino, R. Zaccagnino, R. Zizza, Splicing music composition, *Information Sciences* **385**, 196 (2017)
22. R.D. Prisco, D. Malandrino, G. Zaccagnino, R. Zaccagnino, R. Zizza, in *Theory and Practice of Natural Computing - 6th International Conference, TPNC 2017, Prague, Czech Republic, December 18-20, 2017, Proceedings, Lecture Notes in Computer Science*, vol. 10687, ed. by C. Martín-Vide, R. Neruda, M.A. Vega-Rodríguez (Springer, 2017), *Lecture Notes in Computer Science*, vol. 10687, pp. 219–231
23. T. Head, Formal language theory and dna: an analysis of the generative capacity of specific recombinant behaviours, *Bulletin of Mathematical Biology* **49**, 737 (1987)
24. G. Păun, On the splicing operation., *Discrete Applied Mathematics* **70**, 57 (1996)
25. D. Pixton, Regularity of splicing languages., *Discrete Applied Mathematics* **69**(1-2), 101 (1996)
26. W. Piston, M. DeVoto, *harmony* (W. W. Norton, 1987)
27. M. Levine, *The jazz theory book* (Curci, 2009)
28. C.H. Hsiao, M. Cafarella, S. Narayanasamy, (ACM, 2014), OOPSLA, pp. 49–65