Integration of Non-Parametric Fuzzy Classification with an Evolutionary-Developmental Framework to perform Music Sentiment-based Analysis and Composition

Ralph Abboud[1] and Joe Tekli[2]

Lebanese American University (LAU), School of Engineering, E.C.E. Dept.

Byblos Campus, 36 Byblos, Lebanon

Abstract. Over the past years, several approaches have been developed to create algorithmic music composers. Most existing solutions focus on composing music that appears theoretically correct or interesting to the listener. However, few methods have targeted sentiment-based music composition: generating music that expresses human emotions. The few existing methods are restricted in the spectrum of emotions they can express (usually to two dimensions: valence and arousal) as well as the level of sophistication of the music they compose (usually monophonic, following translation-based, predefined templates or heuristic textures). In this paper, we introduce a new algorithmic framework for autonomous Music Sentiment-based Expression and Composition, titled MUSEC, that perceives an extensible set of six primary human emotions (e.g., anger, fear, joy, love, sadness, and surprise) expressed by a MIDI musical file, and then composes (creates) new polyphonic, (pseudo) thematic, and diversified musical pieces that express these emotions. Unlike existing solutions, MUSEC is: i) a hybrid crossover between supervised learning (*SL*, to learn sentiments from music) and evolutionary computation (for music composition, *MC*), where *SL* serves at the fitness function of *MC* to compose music that expresses target sentiments, ii) extensible in the panel of emotions it can convey, producing pieces that reflect a target crisp sentiment (e.g., love) or a collection of fuzzy sentiments (e.g., 65% happy, 20% sad, and 15% angry), compared with crisp-only or two-dimensional (*valence*/*arousal*) sentiment models used in existing solutions, iii) adopts the evolutionary-developmental model, using an extensive set of specially designed music-theoretic mutation operators (*trille*, *staccato*, *repeat*, *compress*, etc.), stochastically orchestrated to add atomic (individual chord-level) and thematic (chord pattern-level) variability to the composed polyphonic pieces, compared with traditional evolutionary solutions producing monophonic and non-thematic music. We conducted a large battery of tests to evaluate MUSEC's effectiveness and efficiency in both sentiment analysis and composition. It was trained on a specially constructed set of 120 MIDI pieces, including 70 sentiment-annotated pieces: the first significant dataset of sentiment-labeled MIDI music made available online as a benchmark for future research in this area. Results are encouraging and highlight the potential of our approach in different application domains, ranging over music information retrieval, music composition, assistive music therapy, and emotional intelligence.

Keywords: Music Sentiment Analysis, MIDI, Evolutionary Algorithms, Algorithmic Composition, Supervised Learning, Fuzzy Classification.

1. Introduction

Long before the existence of computers, music experts attempted to develop computational procedures to automatically compose music. Notably, the great composer Wolfgang Amadeus Mozart made a dice game to create a bespoke eight-bar minuet using random dice tosses. Yet, such efforts paled in comparison with the sophisticated and appealing musical pieces that human composers would produce. This aspect of composition faded into the background as time elapsed until computers became more accessible toward the end of the twentieth century, where interest in algorithmic music composition was rekindled. The Illiac Suite (Di Nunzio A. 2014; Sandred O. et al. 2009) was one of the first computer-assisted composition tools, written in 1957 by Hiller and Isaacson. Since then, several approaches and models have been adopted to automate the music composition process.

Early approaches have focused on "translating" phenomena and patterns (extracted from texts, images, or measurements) into musical pieces using so-called translational models (mapping variations from patterns into music), e.g., (Hiller L. 1970; Hiller L. et al. 1959; Wolfram Tones Inc. 2005). Other studies have leveraged well-known mathematical models (e.g., formal grammars or Markov chains), oftentimes in tandem with musical rules and stochastic processes, to create novel and well-structured music, e.g. (Husarik S. 1983; Serra M. H. 1993). A common criticism for these approaches is that they are quite rigid in representing music variability and expressiveness (Papadopoulos G. et al. 1999). More recent methods have exploited machine learning techniques to emulate a human composer's "inspiration" process by learning from existing compositions (getting "inspiration") to create new ones, e.g., (Adiloglu K. et al. 2007; Reimer M. A. et al. 2014). Other recent methods have utilized evolutionary approaches which strive to compose a large number of musical pieces and ultimately keep the "best" ones, simulating the biological process of natural selection. Yet, traditional evolutionary algorithms require full-fledged pieces of music as input (initial population), to produce (mutated and crossed) pieces as output. In a recent approach in (Molina A. et al. 2016), the authors adopted an evolutionary-developmental (Evo-Devo) model to construct more elaborate pieces from simpler ones. Initial results showed that the produced compositions were deemed appealing by human listeners.

---

[1] The author is currently completing his Master's and has been accepted in the Ph.D. program of the Computer Science dept., Univ. of Oxford, UK

[2] The author is currently conducting a Fulbright Visiting Scholar research mission in the Computer and Info. Science dept., Univ. of Michigan, USA

While most existing solutions have focused on composing music that appears theoretically correct or interesting to the listener, few methods have targeted sentiment-based music composition: generating music that expresses human emotions. Existing sentiment-based solutions reduce human emotions (e.g., anger, fear, surprise, etc.) to two main dimensions: *valence* (pleasure received from an emotion), and *arousal* (engagement with the emotion), e.g., (Hoeberechts M. et al. 2009; Huang C. et al. 2013), which tend to diverge indicating a potential bias or ambiguity in the model (Russell J. 1980). Another limitation of existing methods is the level of sophistication of the music being composed: producing monophonic pieces[1] (Hoeberechts M. et al. 2009; Kirke A. et al. 2017), or relying on predefined templates or heuristics to extend a base monophonic melody into polyphonic music (Kirke A. et al. 2011).

The main goal of our study is to develop a sentiment-based music composer that can produce musical pieces that are: i) expressive in the human emotions they can convey, ii) more advanced in the level of sophistication of the music textures that they produce, while being iii) appealing and enjoyable by listeners. To this end, we introduce MUSEC, a framework for autonomous Music Sentiment-based Expression and Composition, designed to perceive an extensible set of six human emotions (e.g., anger, fear, joy, love, sadness, and surprise) expressed by a symbolic MIDI[2] musical file, and then compose (create) new original musical pieces (in MIDI format) with sophisticated polyphonic textures that can express these sentiments. To achieve this, MUSEC first "learns" how to perceive emotions in music using a supervised machine learning process. We view this as a required step toward sentiment-based composition, similarly to human composers who first need to appreciate a human emotion to be able to truly reflect it in their compositions. Then, MUSEC starts composing using a dedicated evolutionary-developmental approach, by evolving simple (atomic) music patterns into more sophisticated and full-fledged pieces that express the user's target sentiments.

More specifically, MUSEC consists of four main modules: i) music (symbolic and frequency-domain) feature parser (*FP*), ii) music theory knowledge base (*KB*, including music-theoretic operations and rules to produce "correct" music), iii) music sentiment learner (*SL*, consisting of a non-parametric fuzzy classifier[3] coined with a dedicated music similarity evaluation engine) that learns to infer sentiments in music from exposure to previously analyzed musical pieces, and iv) music sentiment-based composer (*MC*: the core component of MUSEC, consisting of an evolutionary-developmental framework, with specially tailored evolution, mutation, and sentiment-based trimming operators), to generate new, original, and diversified music compositions that express target emotions.

Different from existing solutions (cf. Section 3.2.4), MUSEC is a hybrid crossover between supervised learning (*SL*, to learn sentiments from music) and evolutionary computation (for music composition, *MC*), where *SL* serves at the fitness function of *MC* to compose music that expresses target sentiments. In turn, the composer (*MC*) feeds the learner (*SL*) in order to better infer sentiments from exposure to newly composed pieces. Also, MUSEC is expressive and extensible in the panel of emotions it can convey, producing pieces that reflect a target crisp sentiment (e.g., love) or a collection of fuzzy sentiments (e.g., 65% happy, 20% sad, and 15% angry), where the number of sentiment categories can be extended following the user's preferences, compared with crisp-only or two-dimensional (*valence*/*arousal*) sentiment models used in existing solutions. In addition, it adopts the evolutionary-developmental model, using an extensive set of specially designed music-theoretic mutation operators (*trille*, *staccato*, *repeat*, *compress*, *single suspension*, etc.), which are stochastically applied within the evolutionary process, to add atomic (individual chord-level[4]) and thematic (chord pattern-level) variability to the composed polyphonic pieces, compared with traditional evolutionary solutions producing monophonic and non-thematic music.

We conducted a large battery of tests to evaluate MUSEC's effectiveness in sentiment learning accuracy, music composition quality, enjoyment (appeal), and accuracy in expressing target emotions, involving assessments by non-expert music listeners as well as expert music instructors and composers[5]. Inter-tester correlations were evaluated and matched with MUSEC's scores to account for human listener subjectivity. We also evaluated MUSEC's efficiency in feature parsing time, sentiment expression time, and music composition time. To our knowledge, this is most extensive experimental evaluation of a music sentiment analysis and composition approach to date. Our system was trained on a specially constructed set of 120 MIDI pieces, including 70 sentiment-annotated pieces: the first significant dataset of sentiment-labeled MIDI music made available online[6] as a benchmark for future research in this area. Results are encouraging and highlight the potential of our approach in different application domains, ranging over sentiment-based music retrieval, music composition, assistive music therapy, and emotional intelligence.

The remainder is organized as follows. Section 2 provides background in music theory and the MIDI standard. Section 3 reviews the literature in music sentiment analysis and algorithmic composition. Our MUSEC framework is

---

[1] Simplest form of musical textures where only one note is played at a time, in contrast with polyphonic music where more than one note is played simultaneously.

[2] Musical Instrument Digital Interface: a digital music format designed for symbolic music representation and processing by computers.

[3] A fuzzy classifier is a classifier which assigns membership scores to input data objects, producing fuzzy categories with fuzzy boundaries, such that an object, e.g., a musical piece, can be part of one category and the other at the same time (e.g., 80% excitement and 20% fear), in contract with traditional crisp classifiers which categorize data in crisp/distinct categories (Kotsiantis S.B., 2007). In our current system, we utilize fuzzy *k*-NN due to its flexibility and effectiveness, yet any other fuzzy classifier could be used, e.g., (Abu Arqub O., 2017; Abu Arqub O. et al. 2016; Amin F., et al. 2017; Fahmi A. et al. 2018; Fahmi A. et al. 2019; Fahmi A. et al. 2017)..

[4] A chord is a combination of 3 or more notes (cf. Section 2).

[5] http://www.lau.edu.lb/news-events/news/archive/music_composers_face_off_with_/. Details are provided in Section 8.

[6] Available online at: http://sigappfr.acm.org/Projects/MUSEC, including MUSEC synthetic compositions and all experimental results.

developed in Section 4. Section 5 evaluates the solution's theoretical complexity. Section 6 presents experimental results. Potential applications are discussed in Section 7, before concluding with future directions in Section 8.

2. Background

Our study builds upon concepts from music theory and uses the MIDI format to process musical files. We therefore provide a brief presentation of both concepts in the following subsections. Interested readers can refer to (Danhauser A. 1994) for a more detailed review of music theoretic concepts and paradigms.

2.1. Music Theory

Music is innate to human beings by nature, who get sentimentally attached to musical pieces or have their moods altered by them, by instinctively and effortlessly following a tune's beat or melody. However, when asked to properly describe a musical piece's features, non-expert listeners struggle to convey their own perceptions to others. This is where music theory comes into play. Music theory is a formalization of the relationships and interplay between the different frequencies that make up the music we listen to. In other words, it defines rules and recommendations to help describe, reproduce, and compose music (Danhauser A. 1994). In the following, we only provide a brief overview of the music theoretic constructs covered in our study.

Music comprises of ***notes***, which combine to create a piece's overall melody. Notes denote a certain frequency being played at a given point in time. This abstraction is further supported by the notions of ***chroma*** and ***pitch***. On one hand, ***chromas*** define a logarithmic classification of notes based on their fundamental frequencies[1]. In occidental music[2] theory, we identify 12 main chroma classes (i.e., C, C#/D♭, D, D#, E♭, E, F, F#/G♭, G, G#/A♭, A, A#/B♭, B), such that every note invariably belongs to one class. On the other hand, ***pitch*** helps distinguish between two notes sharing the same chroma class, based on their different fundamental frequencies. For instance, a note with a frequency of 440 Hz and another with a frequency of 880 Hz both belong to the *A* chroma class, but have different pitches.

The concept of ***intervals*** is introduced to distinguish between notes. Intervals describe the gap between two musical notes, as a logarithmic measure of the ratio of the two notes' fundamental frequencies. Intervals are an essential construct used for determining the harmony and appeal of music, and as such have been extensively studied and categorized. Intervals are measured in ***tones***: a unit which helps perform interval computations using frequency additions rather than frequency ratio multiplications. Popular intervals in music theory include the perfect fourth (2.5 tones), the perfect fifth (3.5 tones), the minor third (1.5 tones), and the octave (6 tones), where a tone denotes the smallest gap between two distinct notes having the same chroma.

The concepts we have discussed above are also used when many notes are sounded together, forming so-called ***chords***, consisting of an appealing combination of notes. A chord is a group of notes (normally three or more) that are played together following a certain interval structure. Intervals are used to assess chords' structure and musicality, while chromas and pitches help identify a particular realization of a certain chord structure. More formally, intervals define a chord's ***type***, which in occidental music theory could be *major*, *minor*, *augmented* or *diminished*, while pitches and chromas define a chord's ***root note*** (i.e., the note that best represents the chord). Manipulating a chord's structure whilst maintaining the same use of pitches creates chord ***inversions***, i.e., chords having a different relative ordering of notes, which are very popularly used in music composition. The sequence of chords played in a piece, also known as its ***chord progression***, can very accurately describe the said piece's musicality. Connecting consistent chord progressions in a musical piece is therefore an integral part of any music composition task.

A **musical key** is a set of interval-related pitches and chords, whose combinations produce coherent and enjoyable music. Analogously to chords, a key also has its own ***root note*** and ***type*** (which is usually: *major* or *minor*). The key type used in a composition is known to correlate with its overall "feel" or perception by human listeners, with *minor* keys usually producing sadder compositions and *major* keys usually producing happier and more upbeat musical pieces, e.g., (Hevner K. 1935; Livingstone S. R. et al. 2010). For greater beauty and unpredictability, composers can also change keys within the same musical piece: a process known as ***modulation***.

Music texture is how the melodic, rhythmic, and harmonic materials are combined in a composition, thus determining the overall quality (or so-called sophistication) of the piece. Here, we distinguish between two grand types of textures: **monophonic** where a single melodic line is played with no accompaniment (i.e., only one note is played at a time), and **polyphonic**, where multiple melodic lines (i.e., more than one note) are played simultaneously.

2.2. MIDI Standard

MIDI, short for Musical Instrument Digital Interface, is a standardized symbolic music format designed to record musical performances using so-called *high-level music features* (i.e., features based on musical note abstractions, such as musical key, chord progressions, etc.), rather than traditional low-level audio/sound features (i.e., features based on frequency data used to describe audio formats, such as spectral components of audio samples and frequency histograms, etc.).

---

[1] A *fundamental frequency* is the lowest frequency produced by the oscillation of an object. In music, it is perceived as the lowest *partial* (simple tone) present that is distinct from the *harmonics* of higher frequency. In the remainder of this paper, terms *frequency* and *fundamental frequency* will be used interchangeably, unless explicitly stated otherwise.

[2] Music produced following the traditions of *Western* (European) culture, compared with *Oriental* (Byzantine, Mizrahi, or Asian) music.

A MIDI file consists of several ***tracks***, each of which can play a different instrument independently of the other tracks. For any MIDI file, the basic time unit is the ***tick***, regulating all note onsets and time measurements within the MIDI standard. Within every track, a set of ***MIDI events*** can occur at a certain tick position to indicate a change within the melody or in the overall piece. These events usually carry ***MIDI messages***, e.g., *meta messages* which add further information to a MIDI file such as the piece's key and tempo (i.e., the speed at which a musical piece is played); as well as NOTE ON and NOTE OFF messages which respectively signal the start or end of a certain note. The latter messages, which help define the onset of a note in MIDI, provide two main parameters: i) ***velocity***: A 7-bit number between 0 and 127 indicating the intensity with which the note is played (the higher velocity, the more powerfully and intensely the corresponding note is played), and ii) ***MIDI pitch***: A 7-bit number between 0 and 127 specifying the musical pitch to be played, where each value maps to a specific note frequency. The tick position of the message's event specifies the time at which notes are turned on and off.

In the following, we rely on the MIDI format in devising our abstractions of musical note and musical piece in MUSEC (described in more detail in Section 4).

3. Literature Review

Developing an autonomous and self-learning sentiment-based music composition system requires combining knowledge from different domains in the literature, namely *music sentiment analysis* and *algorithmic music composition*, which we briefly review in the following subsections. We also describe the few existing works that specifically address the problem *of sentiment-based music composition*[1], to better situate and compare our approach w.r.t.[2] its predecessors.

3.1. Music Sentiment Analysis

Music Sentiment Analysis (or MSA) is one of many hot problems within the broader field of Music Information Retrieval (MIR), which deals with the representation, description, storage, and retrieval of affective information from music (Demopoulos R.J. et al. 2007; Kirke A. et al. 2009). Much like standard IR systems, MIR systems (and MSA systems in particular) convert music documents into feature representations, which are then utilized to retrieve relevant information. In the case of MSA, the retrieved information comes down to crisp sentiment categories or fuzzy sentiment scores.

3.1.1. Sentiment Representation Models

Performing MSA first requires a *sentiment model* through which human emotions can be represented, so that they can be relayed to music feature vectors. In this context, two main categories of models have been adopted in the literature: i) dimensional, and ii) categorical. On one hand, following the *dimensional* approach, every emotion is mapped to a certain dimensional space, such that all emotions and their combinations can be represented in the said space. Though various models have been proposed in the literature (Ravi K. et al. 2015; Zentner M. et al. 2010), the one most commonly adopted in MSA is Russell's *valence-arousal* model (Russell J. 1980), which maps human emotions to two essential dimensions: i) *valence*: the pleasure received from an emotion, and ii) *arousal*: the engagement with the said emotion. On the other hand, following the *categorical* approach, human emotion is represented by a set of pre-defined sentiment categories (e.g., anger, surprise, fear, love, etc.), the combination of which is believed to portray the complete human spectrum of emotions (Zentner M. et al. 2010). This model has been extended from crisp ($\in \{0, 1\}$) to scaled ($\in [0, 1]$) sentiment categories (Abbasi A. et al. 2008; Subasic P. et al. 2001), allowing a more expressive representation of the sentiment space. Methods in this category are commonly used in text-based sentiment analysis (Ravi K. et al. 2015) are have been recently adapted toward MSA (Hoeberechts M. et al. 2009).

Both models mentioned above present advantages and disadvantages. The dimensional model, on one hand, provides a reduced dimensional space (namely 2 dimensions) which is relatively easier (faster) to process (computationally), compared with a multi-category space of n (crisp, or scaled) emotion categories. Nonetheless, states where both valence and arousal dimensions converge (e.g., both valence and arousal are high, or both are low) occur more often than states were they diverge (Russell J. 1980), indicating a potential bias or ambiguity in the model (as stated by the model's creator in (Russell J. 1980)), which in turn highlights the model's limited expressiveness in distinguishing different emotions. On the other hand, the categorical model provides an intuitive and expressive approach to represent human emotions, where different emotion categories are used to represent the human emotion spectrum. More or less categories can be considered following the user's needs, which in turn decreases or increases computational efficiency. However, this also begs the question of how to choose the proper subset of emotion categories that best fit the application or scenario at hand, which remains an open debate among researchers in the field (Morreale F. et al. 2016).

In our current study, we adopt the extended (scaled) categorical model, considering six primary categories of emotions commonly adopted in the literature (Ekman P. 1993): anger, fear, joy, love, sadness, and surprise. More (or less) emotion categories can be later considered following the user's preferences.

---

[1] Also referred to in the literature as *affective music composition*

[2] with respect to

3.1.2. Music Analysis Features

Musical features range over *high-level symbolic* features (a.k.a.[1] *music-theoretic* features, based on musical note abstractions) and *low-level frequency-domain* features (a.k.a. *statistical* features, based on frequency data used to describe audio formats) (Demopoulos R.J. et al. 2007). Many approaches in the literature combine both feature ranges into so-called multimodal feature vectors (Schedl M. et al. 2014). With the introduction of the MIDI format in the 1980s, more sophisticated musical features became available, fueling further interest in this area. Some approaches in MIR have also built on breakthroughs in text-based sentiment analysis to improve music sentiment analysis, by incorporating music lyrics as an additional entry to be analyzed (Panda R. et al. 2013).

One of the earliest MSA solutions, developed in the late 1980s by Katayose *et al.* (Katayose H. et al. 1989) firmly placed its emphasis on purely music-theoretical features. In this approach, the authors develop an artificial music expert, a system that can detect and treat music like a human: through its emotions. To do this, they introduced "*quasi-sentiments*", a semantic/emotional meaning behind a given piece, so as to emulate how a human would react to a piece. Their extraction technique consists of mapping musical phenomena to these quasi-sentiments using a set of pre-defined rules. For example, a certain chord progression could correspond to a sad emotion, while a certain key or tempo could indicate a happy emotion. Through a simple rule-based approach, the authors were able to use musical features parsed from the input musical piece to infer its underlying sentiments.

More recent efforts attempt to use as many features as possible, be it content-based (symbolic and/or sampled audio) or textual (lyrics of a song) to extract the sentiments from a given musical piece (Fleischman M.B. et al. 2013; Panda R. et al. 2013; Wohlfahrt-Laymanna J. et al. 2017; Xiao H. et al. 2010). For example, Panda *et al.* (Panda R. et al. 2013) perform sentiment-based retrieval based on a set of 253 simple musical features, 98 frequency domain features, 278 symbolic features, and 19 lyrical features. From this very large feature set, the authors seek to select the best combination of features to perform the sentiment analysis task. Results, based on optimal feature selection and retrieval performance testing for multiple machine learning and classification algorithms (SVM[2], *k*-NN[3], etc.) clearly showed that using multiple feature types can improve retrieval performance. Indeed, the best feature configuration for frequency domain-only features yielded an optimal f-value of 44.3%, while a hybrid feature selection of 15 frequency domain and 4 symbolic features scored an f-value of 61.1% (Panda R. et al. 2013). On one hand, this improvement shows the potential of using multimodal features, but it also shows that lyrical features did not help to improve system performance in this particular study. On the other hand, other studies, e.g., (Panda R. et al. 2013; Xiao H. et al. 2010), have highlighted the positive impact that lyrical features can make in MIR/MSA. In (Xiao H. et al. 2010), Hu and Downie incorporate lyrical features into their testing and report a 9.6% accuracy improvement over the best frequency domain-only features they tested. Approaches in (Fleischman M.B. et al. 2013; Wohlfahrt-Laymanna J. et al. 2017) have suggested considering user profiles, moods, and context information, in addition to content-based and textual music features, to generate sentiment-aware and contextually meaningful music playlists.

Therefore, we can see that the latest trend: i.e., performing sentiment analysis using multiple feature values; is receiving more interest and tends to produce better results. Yet, one can also realize that this domain is still very much in flux as extracting the best features for music sentiment analysis and retrieval remains an open research topic.

Interested readers can refer to (Demopoulos R.J. et al. 2007; Orio N. 2006; Song Y. et al. 2012) for detailed reviews on MIR and MSA.

3.2. Algorithmic Music Composition

Algorithmic Music Composition (AMC) is a research field aiming to produce autonomous computer systems capable of producing/composing new music, e.g., (Dubois R.L. 2003; Fernandez J. et al. 2013; Wolfram Tones Inc. 2005). Researchers in Artificial Intelligence (AI) view AMC as a sub-problem of the bigger problem of *computer creativity* (Boden M. A. 1994). Even today, it remains extremely difficult for an AI agent to innovate and to create something it has not previously seen in a thoroughly convincing fashion. This probably stems from a lack of understanding of the creative process itself, due to which many creativity theories and models were developed (Schank R. C. et al. 1995). In this context, several approaches have been adopted to automate the music composition process and emulate human composers, which we group in four main categories: i) translation-based solutions, ii) mathematical model-based solutions, iii) machine learning-based solutions, and iv) evolutionary solutions.

3.2.1. Translation-based Solutions

Following the translation-based approach (also known as: *data sonification*), the computer accepts as input any piece of data, e.g., text, image, measurement, or random process, and then "translates" it into music using a pre-defined set of (data specific) rules (Fernandez J. et al. 2013; Freeman J. 2015). One example of this approach is WolframTones (Wolfram Tones Inc. 2005), developed by WolframAlpha, which uses cellular automata patterns selected at random from a set of possible patterns, fed into dedicated progression rules and functions to generate new music. To ensure that the produced music is also appealing, pre-defined filters are applied to eliminate any potential causes of musical dissonance. The promise of this approach lies in that new and unexpected music can be created without the need for

[1] also known as

[2] Support Vector Machine

[3] *k*-Nearest Neighbor

sophisticated algorithms, since the novelty lies in the input itself. However, this promise is counterweighed by the difficulty in selecting appropriate inputs and converting them reliably into music. To perform these tasks, special care needs to be taken in designing the appropriate filters.

3.2.2. Mathematical Model-based Solutions

A second category of methods is mathematical model-based solutions, where mathematical constructs like *formal grammars* (Freeman J. 2015) and *Markov Chains* (Verbeurgt K. et al. 2004) are mapped to musical events so as to produce music. Using a *formal grammar*, an alphabet of musical states is defined, as well as a set of starting states and production rules to extend the initial musical states (McCormack J. 1996). Lindenmayer Systems, a.k.a. *L-systems*, are a special kind of formal grammars which are adapted to music composition (Manousakis S. 2006; Worth P. et al. 2005). They are a variant of formal grammars previously successfully applied to biological modeling (Prusinkiewicz P. et al. 1990) and allow parallel rewriting of grammar strings. For instance, DuBois's approach in (Dubois R.L. 2003) relies on L-systems, where symbols are defined as notes (musical objects), and dedicated transformations are used to create music. To support polyphony, brackets are used to surround a multitude of notes (objects). This approach also uses another L-system to add synthetic accompaniment to the generated music.

Though the formal and expressive structures of grammars can constitute a solid model for music composition, a common criticism of grammar-based methods is that they seem quite rigid in representing music diversity and expressiveness (Papadopoulos G. et al. 1999). To remedy this, some approaches have incorporated stochastic techniques to learn grammar parameters using *Markov Chain models* (Demopoulos R.J. et al. 2007; Fernandez J. et al. 2013). Following this approach, experts define musical states and transition probabilities to allow the system to move between states and generate music. State transition probabilities are inferred from previous states (i.e., states at previous iterations of the system). The larger the number of previous iterations considered in probability inference, the farther back a system reaches in its "memory" of states and transitions to perform decision making in its present iteration. Here, the usage of memory presents a quality/complexity trade-off (Fernandez J. et al. 2013). On one hand, a memory-less system (where present state transitions are independent of the previous states) has a relatively simpler transition probability matrix, but will probably behave more randomly and might seem less fit to produce organized music structures. On the other hand, a memory-based system takes previous states into account, and thus tends to produce more structured and better organized musical pieces, but is much more complicated to implement and to develop, particularly given the size of the resulting transition matrix.

3.2.3. Machine Learning-based Solutions

Some approaches have investigated machine learning-based solutions, in an attempt to learn from existing compositions so as to create new ones. Machine learning techniques can be used either as a standalone component to compose music directly (Reimer M. A. et al. 2014), or as part of a larger approach to learn parameters, such as learning transition probabilities with more recent Markov Chain-based approaches (Verbeurgt K. et al. 2004).

A common technique used in this context is *Artificial Neural Networks* (ANNs) (Kotsiantis S. B. 2007). ANNs are a parametric computational model designed to mimic the human brain by learning data with a set of parameters of fixed size: defining the structure and functionality of the so-called artificial brain or ANN. They consist of artificial neurons, which receive one to several stimuli and produce a single output. They are generally organized into several layers[1] and their activation functions are usually non-linear. Most commonly, these networks are fed examples so as to adjust their stimuli weights in order to achieve the desired output. This type of training is referred to as *supervised learning*[2], where experts prepare a set of labeled musical pieces (referred to as the *training set*) through which they train their networks and "teach" them to compose new pieces. Training pieces are either fed into the network as a single example (i.e. the piece itself is one training example), or in chunks (such that a single piece is temporally divided into several training examples). For instance, the authors in (Reimer M. A. et al. 2014) utilize multiple neural networks, organized in two layers, a feature layer and a creative layer, to create music. The ANNs used are known as ART (Adaptive Resonance Theory) neural networks, which are designed to train and test in real-time, and to train one example at a time. The feature layer consists of three ARTs, where each ART assesses a candidate input musical note based on three separate criteria: i) pitch, ii) the piece's overall melodic continuity, and iii) the melodic interval between the pitch and its predecessor. Based on the given input, every ART suggests its own continuation pitch, based on its own criteria. These suggestions are then used as input for the creative layer. The creative layer is the ultimate decision-making component in this approach. It takes the three previously computed suggestions and selects the one which changes its network weights the most. The rationale behind this decision-making process is that musical novelty is related to weight change: the more change is produced by a candidate note, the more innovative and attractive it is (Reimer M. A. et al. 2014). Eventually, the creative layer produces an output note, which in turn is fed

---

[1] An ANN with several hidden layers between the input and the output layers is called a *deep neural network* or a *deep learner*.

[2] It is a machine learning approach which allows the learning of a function that maps an input (e.g., musical piece) to an output (e.g., sentiment category or sentiment score) based on sample input-output pairs, so-called *labeled training data,* where each sample pair consists of a given input object (e.g., a music feature vector) and a desired output value (e.g., a sentiment category or a sentiment score). The produced mapping function is an approximation of the true mapping function between the sample training pairs (Kotsiantis S.B., 2007).

back into the feature layer, at which point the process starts anew, until a long enough monophonic piece is composed. A similar approach using ANNs is utilized in (Burton A. R. 1998) to learn and identify drum percussions.

However, the parametric nature of ANNs has its strong and weak points. On one hand, a set of fixed size parameters allows to simplify the learning process, since parametric assumptions about the ANN architecture and functionality (number of layers, network connectivity, activation functions, and activation thresholds, among others) remain independent of the number and nature of the training musical pieces considered. On the other hand, a set of fixed size parameters also limits what can be learned by the ANN: no matter how many or how different input music one feeds the parametric ANN model, it will not change its mind about how many parameters it needs, which tends to limit its expressiveness. This is the reason we adopt a non-parametric learning method in our approach (Section 4.3).

3.2.4. Evolutionary Solutions

Finally, *evolutionary* algorithms[1] have been developed to attempt to create music in a way much like nature creates individuals, organisms, and species. Following this approach, experts define the structure of their so-called "individuals", their properties, and corresponding evolution (crossover and mutation) mechanisms to produce new and diverse generations of offspring individuals. A fitness function is carefully designed to select the best among existing individuals (as potential survivors of their generation, and candidates for mating to produce offspring), so as to emulate the process of natural selection (Abu Arqub O. et al. 2014; Goldberg D. 1989; Whitley D. et al. 2012).

Generally, evolution is emulated following two methods. The first method follows the so-called *traditional evolutionary model*, where an individual's structure remains intact, and only its genes' expressions change, e.g., (Marques M. et al. 2000; Matic D. 2010; Ozcan E. et al. 2008; Pavlov S. et al. 2014). For instance, the authors in (Pavlov S. et al. 2014) adopt the traditional evolutionary model where they define their individuals as being *n*-bar musical pieces. Mutations that an individual (piece) can undergo include note pitch changes, note duration changes, and note position swaps. To emulate crossover, offspring (new pieces) randomly choose their 4 bars from their parents (pieces from the previous generation), such that the offspring form mixes of their parents. The authors utilize a music-theoretic fitness function to assess the quality of the interval jump between a composition's pitches, in order to create musically-correct monophonic pieces. Similar approaches are developed in (Marques M. et al. 2000; Matic D. 2010; Ozcan E. et al. 2008), where the authors represent music based on different rhythmic and melodic features, such as breaks and relative pitches. Modified genetic operators allow to change the scheduling of pitches and breaks in order to produce new pieces, while preventing premature convergence using a music-theoretic fitness function.

A second evolutionary method is the so-called *Evolutionary-Developmental* (or *Evo-Devo*) model (Molina A. et al. 2016), a high-level abstraction of the evolution process where individuals are considered to be initially very rudimentary, only to grow in sophistication as generations pass. In other words, the offspring are not simply mixes of their parents' gene material, but are composite aggregates of the latter. Here, an offspring's chromosome consists in part of mutated and crossed genes, to which is then aggregated other genetic material from individuals of the previous generation, producing a more sophisticated and complex genetic organism in every subsequent generation. The approach ends when individuals reach the amount (length) of sophistication required. This approach was used by IAMUS (Diaz-Jerez G. 2011), an artificial composer which compositions were deemed appealing to human listeners and have been performed in theatres to the public.

Considering the above classification, the approach adopted in our study can be viewed as a hybrid crossover between: i) an *evolutionary composer* agent who integrates ii) a non-parametric *machine learning* agent serving as its fitness function evaluator. Also, different from most approaches above which were designed to create musical pieces that appear theoretically correct or interesting, our solution adds a central functionality: producing music that expresses (reflects) sentiments (in the form of crisp emotions, e.g., love, sadness, or fuzzy emotions, e.g., 65% happiness, 35% anger).

3.3. Sentiment-based Music Composition

In recent years, few research efforts have sought to supplement AMC models with the ability to create compositions that reflect human emotions, much like human composers who create their own compositions in order to express their feelings. Interest in this problem is relatively recent, with almost all related works and publications appearing over the last few years, e.g., (Hoeberechts M. et al. 2009; Huang C. et al. 2013; Kirke A. et al. 2011; Kirke A. et al. 2017; Livingstone S. R. et al. 2010; Morreale F. et al. 2016). Most methods in this area fall within the *translation-based* AMC category, with some methods partly including simple *mathematical-based* AMC constructs.

In an initial study in (Livingstone S. R. et al. 2010), Livingstone *et al.* provide a set of rules mapping musical features, such as tempo and key, to the music's expected valence/arousal response. This set of rules (cf. examples in Table 1) provides a simple framework for a translation-based composition process which can produce music by manipulating a set of musical features. Lin and Huang use a similar rule-based mapping in (Huang C. et al. 2013) to manipulate their compositions' features, including rhythm and the pitches used. From an input valence/arousal score

[1] An evolutionary algorithm can be defined as a population-based metaheuristic optimization algorithm, which uses mechanisms inspired by biological evolution, such as reproduction, mutation, crossover, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions. The evolution of the population then takes place after the repeated application of the above operators (Goldberg D. 1989; Whitley D. et al. 2012).

entered explicitly by the user, composition features and their corresponding parameters are manipulated according to the previously mentioned rules to produce music. In (Hoeberechts M. et al. 2009), Hoeberechts and Shantz introduce the AMEE (Algorithmic Music Evolution Engine) system, which adapts its compositions in real-time to users' appraisals of a set of 10 pre-defined emotion categories, including *happiness*, *sadness*, *triumph*, and *defeat*, among others. In this approach, the system works in a hierarchical manner. It starts by defining high-level structures to the music being composed, such as phrases and sections, and then produces low-level realizations of the high-level structures, using forced abortions if needed to ensure that the said structure is respected.

Table 1. Sample *Musical feature*-to-*Valence/Arousal* mapping rules following (Livingstone S. R. et al. 2010)

Musical Feature	Valence	Arousal
Major Key	High	Unrelated
Minor Key	Low	Unrelated
High Volume	Unrelated	High
Low Volume	Unrelated	Low

Morreale and De Angeli in (Morreale F. et al. 2016) also exploit Livingstone's valence/arousal musical rules to develop ROBIN, an autonomous composer and composition assistant that can collaborate with users to create expressive music. From a user's gestures in the music room, valence/arousal scores are inferred, from which ROBIN manipulates seven features of the composition (including, *pitch contour*, *tempo*, *mode*, *complexity*, etc.). The *complexity* feature is defined as being the rhythmic irregularity of the piece, and is introduced by the authors to better portray users' emotions. In selecting notes and chord progressions in the composition process, the authors extend the simple rule-based translation model toward a more sophisticated mathematical model based on a probabilistic Markov process. Another mathematical model-based approach is introduced in (Kirke A. et al. 2011), where the authors leverage Livingstone's rules to perform sentiment-based composition, by obtaining music from EEG (Electroencephalogram) measurements. The authors develop a tool that continuously monitors users' brain activity using EEGs for a certain time period, from which it obtains real-time measurements. Then, valence/arousal levels are inferred from the users' EEGs using another set of rules (e.g., high left versus right frontal alpha implies high arousal). Similarly to Morreale and De Angeli's approach in (Morreale F. et al. 2016), the valence/arousal scores are used to represent the composer's understanding of musical features. Yet, the solution in (Kirke A. et al. 2011) offers the user a greater control over the composition process by allowing them to specify the musical structure and a (simple grammar-like) template of the piece they are composing (such as ABA or ABCA, where A, B, C and D are generic pre-defined theme labels), thereby promising better organized musical pieces.

In (Kirke A. et al. 2017), Kirke and Miranda apply rule-based musical feature mapping to develop a system called TRAC (Textual Research for Affective Composition) for creating affective movie soundtracks given an input movie script (Kirke A. et al. 2017). In this approach, the authors modify the valence/arousal model and include a third dimension: *dominance*, to reflect the power imposed by an emotion. Valence, arousal, and dominance scores are extracted locally throughout the script to obtain a time series for the three dimensions, in order to be used for sentiment-based composition. Sentiment extraction is performed at the most basic word-level (using the ANEW database (Bradley M. et al. 1999)) and then aggregated at the syntactic structure level (using syntax/parse trees (Carnie A. 2013)). Once the three time series are generated, the developed system randomly generates a theme, asks the user to specify a piece's (simple grammar-like) structure (similar to (Kirke A. et al. 2011)), and then generates the overall composition based on the perceived sentiment dimensions.

3.4. Discussion

To wrap up, we highlight the main issues and limitations facing existing sentiment-based music composition methods. First, most existing solutions utilize similar *translation-based* AMC models (Hoeberechts M. et al. 2009; Huang C. et al. 2013; Livingstone S. R. et al. 2010), with some methods partly integrating *mathematical* (probabilistic or template) *model*-based AMC techniques (Kirke A. et al. 2011; Kirke A. et al. 2017; Morreale F. et al. 2016). As a result, most of these methods create music with relatively simple algorithmic processes, where the main challenge lies in selecting appropriate inputs and converting them reliably into music. Second, most methods utilize the *dimensional* approach to represent sentiments, where 2 (or 3) sentiment dimensions (namely *valence* and *arousal*) allow for easy and fast processing, despite the model's bias and limited expressiveness in distinguishing different emotions (Russell J. 1980); compared with the (scaled) *categorical* approach allowing a more expressive multi-dimensional space of *n* emotion categories (similarly to AMEE (Hoeberechts M. et al. 2009), the only sentiment-based composition method we know of to adopt the categorical model). Third, the produced compositions vary in terms of the type of music texture (monophonic versus polyphonic), and thus the quality/sophistication of the music produced. On one hand, AMEE (Hoeberechts M. et al. 2009) and TRAC (Kirke A. et al. 2017) only produce monophonic pieces, relying on random phrase generation and fixed rules for phrase concatenation and transformation. On the other hand, the EEG-based composer (Kirke A. et al. 2011) and ROBIN (Morreale F. et al. 2016) produce polyphonic music, albeit using author-developed heuristics to extend an initial monophonic melody into polyphonic music.

4. MUSEC Framework

To address most of the limitations above and provide an expressive sentiment-based composer, we introduce MUSEC (Music Sentiment-based Expression and Composition) a hybrid sentiment-based AMC framework, integrating non-parametric machine learning MSA within an evolutionary composition framework. It composes music that reflects any crisp or scaled combination of an extensible set of 6 primary human emotions (i.e., anger, fear, joy, love, sadness, and surprise) while producing sophisticated and varied polyphonic music textures. MUSEC's overall architecture is depicted in Fig. 1. It is made of four modules: i) music feature parser (*FP*), ii) music theory knowledge base (*KB*), iii) music sentiment learner (*SL*), and iv) music sentiment-based composer (*MC*).

We develop MUSEC's modules in the following subsections.

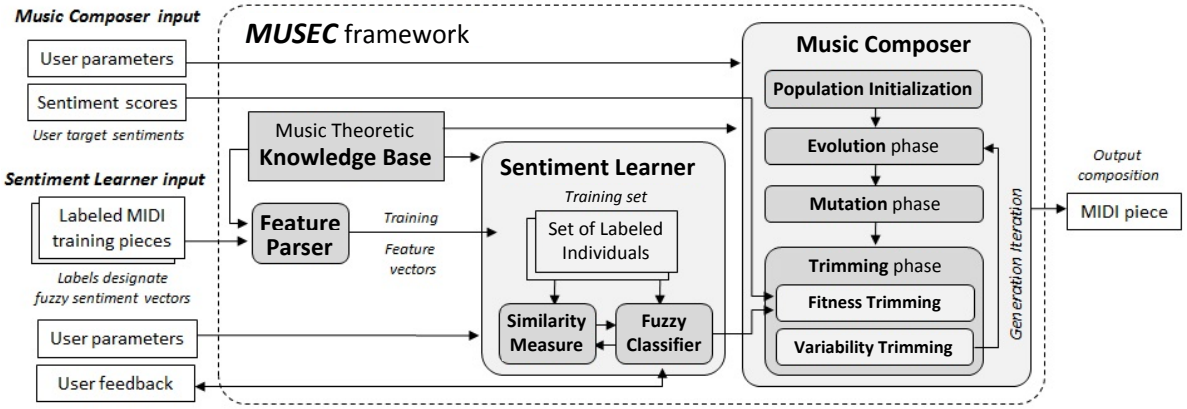


Fig. 1. Simplified activity diagram describing MUSEC's overall architecture

4.1. Feature Parsing (*FP*) module

4.1.1. Musical Features Parsing

MUSEC*'s FP* module allows converting an input MIDI file into a feature vector representation that is later used for sentiment analysis and inference. The feature vector consists of a combination of seven features (4 *symbolic* and 3 *frequency-domain* features that were empirically shown to be effective in describing music (Panda R. et al. 2013)[1]), namely:

- **High-level symbolic features:**
  1) *Note density (ND)*: The number of notes performed per musical time beat, computed by dividing the number of notes played in a piece by its total duration in beats.
  2) *Note onset density (NOD)*: The number of distinct note onsets per musical beat. This feature differs from the previous one in that two notes played simultaneously count as one onset in computations. Therefore, $NOD \leq ND$ for any given musical piece. A comparison between *ND* and *NOD* can yield useful insights on a piece's melodic structure: $NOD << ND$ indicates that a piece's notes are played together (in rigid chords) most often, while $NOD \approx ND$ suggests the notes tend to be played sequentially rather than together.
  3) *Dominant key*: The key that is most common and most prominent in the musical piece. Unlike the previous features, identifying the *Dominant Key* is not a deterministic process and requires dedicated heuristics and music-theoretic knowledge to be extracted and processed from the target piece. In this study, we extract this feature following an adaptation of the Bayesian Key-Finding Model[2] (Temperley D. 2002) which was shown to achieve 91.4 % accuracy[3].
  4) *Chord progression*: The set of chords that best describe the musical melody. This feature is the most computationally complex and difficult to parse, and requires the use of heuristics coupled with a maximum-likelihood inference method to achieve satisfactory performance, which we describe in Section 4.1.2.

---

[1] More features could be later added following the user's needs.

[2] To determine the dominant key, a chroma histogram for the input music file is first computed, denoting the percentage of total piece duration in which every chroma can be heard. The histogram is later used to compute likelihood scores using Temperley's key profiles (Temperley D., 2002). *A Bayesian Approach*. The key with the highest score is finally selected as the dominant key (Temperley D., 2002).

[3] Dominant key misidentification can occasionally occur, particularly for pieces where modulations occur very frequently and for atonal music (Temperley D., 2002; Kyogu L., 2008) (e.g., modern music which does not abide by a fixed key).

– **Low-level frequency-domain features:**

5) *Piece tempo*: The overall rhythm/speed of a musical piece (expressed in Beats per Minute (BPM)). This feature can be easily parsed from a MIDI file's metadata.
6) *Average pitch*: A weighted average of every MIDI note's pitch value, with the weight being the note's duration. This feature provides an indication of the overall pitch at which the musical piece's notes are being played in the frequency domain (designating high, medium, or low pitch).
7) *Average intensity*: A weighted average of every MIDI note's velocity value, with the weight being the note's duration. This features indicates the overall intensity of a piece (e.g., calm or loud).

Note that in order to effectively perform feature parsing from MIDI files, we first process the latter for musical *notes* identification, where a *note* consists of a MIDI ON followed by a MIDI OFF message. Once identified, these messages are processed so as to produce *note* abstractions consisting of: i) MIDI Pitch, ii) MIDI velocity, iii) starting time, and iv) duration value. These abstractions are then utilized as seeds for parsing the aforementioned features.

4.1.2. Chord Progression Parsing

Chord progression parsing remains an open problem in the field of MIR (Demopoulos R.J. et al. 2007), with several dedicated studies, e.g., (Kyogu L. 2008; Zenz V. 2007), utilizing sophisticated mathematical methods such as *Hidden Markov Models* or *machine learning* techniques, in order to infer chords from (audio or symbolic) musical pieces. In this context, the state-of-the-art accuracy for symbolic (MIDI) music chord identification tops around 75% (Demopoulos R.J. et al. 2007). Hence, in our current study, we put forth a simple heuristic solution to fit the needs of our MUSEC framework, reaching 90% accuracy levels[1] when applied on MIDI pieces with specific characteristics.

An activity diagram depicting our chord progression parsing heuristic is shown in Fig. 2. It performs beat-based segmentation of a MIDI piece, and then identifies the dominant key in every segment, in order to infer the chord progression in the sequence of segments forming the piece. To do so, it first utilizes the piece's tempo to infer the length of a beat (step 1), which is then used to perform segmentation (step 2). Every segment that is less than 4 beats long is augmented with a *context* of 4 beats by combining it with previous and/or subsequent beat segments (step 3). Then, the dominant key is computed for every augmented segment (step 4), using the same approach adopted for *dominant key* feature parsing (cf. Section 4.1.1). Given an augmented segment and its dominant key, all chords that do not belong to the said key are eliminated (step 5) so as to avoid computing improbable chords, thus eliminating potential false positives stemming from decorative notes. At this point, the likelihood scores for every possible chord are computed based on the frequency of the said chord's chromas in the segment's chroma histogram (step 6). Chord likelihood is computed as the product of the chroma frequencies for 3-note chords, and as the product of the highest 3 frequencies for chords made of more than 3 notes (Zenz V. 2007) (such that the lowest frequency is dropped to eliminate bias towards smaller chords). When one or many chords are possible and likely enough (following a predefined threshold), the likeliest amongst the possible chords is chosen to label the analyzed segment as being the manifestation of the said chord (step 7). In the event that no chords are possible, or that no chord achieves a sufficiently high likelihood, the segment being processed is further augmented by adding notes from the subsequent beat, before reiterating the same processing on the newly augmented segment in the hope of identifying a chord (step 8). The process is repeated iteratively until reaching the end of the MIDI file, identifying the piece's chord progression where identical consecutive chords (if any) are combined into one.

Preliminary experiments on a set of 100 MIDI pieces show that our approach is fairly immune to noise caused by decorative notes and ornamentation, and that it performs pretty well on simple and structured music where there is a clear separation among chords (reaching 90% accuracy, cf. Section 6). Yet, note that our heuristic chord progression extraction solution does err when analyzing more complicated pieces where chords are intertwined (similarly to existing solutions in (Kyogu L. 2008; Zenz V. 2007)).

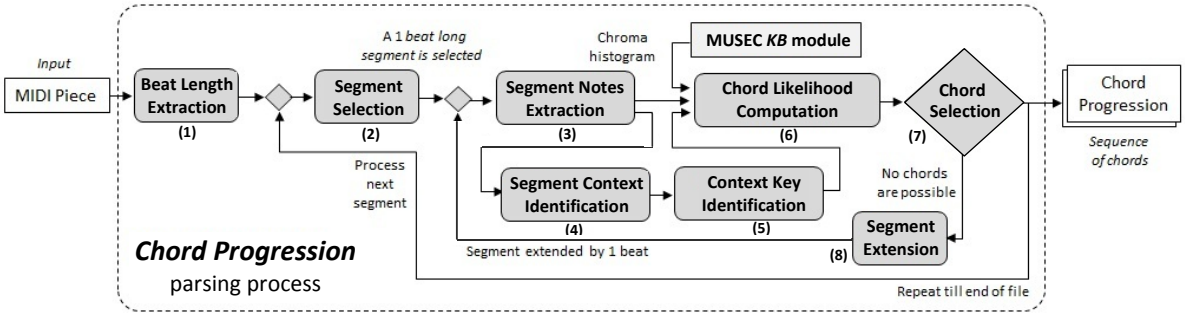


Fig. 2. Simplified activity diagram depicting our *chord progression* parsing heuristic

[1] Note that 100% accuracy in chord progression identification is difficult to obtain due to the very nature of chord progressions: where i) the same chord progression can be played in so many different ways while still portraying the same musical structure, and ii) it can be often difficult to separate between consecutive chords since notes are sometimes combined between them. Our heuristic performs accurately on relatively simple music where there is a clear chord structure, and a clear separation between chords with no rapid transitions between them.

Note that parsing symbolic features from MIDI files, such as *dominant key* and *chord progression*, require music-theoretic knowledge which we house in a dedicated knowledge base (*KB*) module.

4.2. Knowledge Base (KB) Module

MUSEC's *KB* module is a centralized repository of all necessary music-theoretical properties and methods that other MUSEC modules (namely *FP*, *SL*, and *MC*) require in order to perform their own operations. These are highlighted in Table 2 and Table 3 respectively.

For instance, the *FP* module retrieves *Temperley Profile* values from *KB* in order to compute the likeliest key of an input piece. The *SL* module's similarity evaluation component relies on *KB* to compute the *circle of fifths* distance between two keys. Also, evolution and mutation components from the *MC* module invoke support methods from *KB* to identify and build chords, retrieve the notes adjacent to the note being decorated, retrieve relative keys, and identify passing notes. The latter functionalities are further described in the following subsections.

Table 2. Main properties of MUSEC's *KB* module

Property	Description	Used by
Temperley Key Profiles	Used for likeliest key estimation during *dominant key* feature extraction (cf. Section 4.1)	*FP*
Circle of Fifths	Used for key similarity evaluation following distance computation within the *circle of fifths* (cf. Section 4.3.2).	*SL*
Chord Types per root for major and minor keys	List of chord types that can be built on every note of a given key, depending on the key's type. Used for computing the atomic toolbox during the *atomic evolution* phase (cf. Section 4.4.3).	*MC*
Alteration Lists	They contain the number of flats and sharps per key based on key type and root, and are utilized for *chord construction* (cf. Section 4.4.3).	
String Lists	Required to convert pitches and chromas from MUSEC's internal representation to human-legible values and labels (cf. Section 4.4.2)	

Table 3. Main methods of MUSEC's *KB* module

Method	Input	Output	Description	Used by
Chord / Key Compatibility checker	Chord and Key	Boolean (compatible/not compatible)	Checks whether a chord is part of a key's seven main chords. Used during *chord progression parsing* to eliminate unlikely chord possibilities (cf. Section 4.1.2)	*FE*
Chord Likelihood Estimation	Musical Segment	Chord Likelihood Scores (%)	Used during *chord progression parsing* to produce likelihood estimates for all possible chords within a given musical segment, so as to determine the likeliest chord being played in the said segment (cf. Section 4.1.2)	
Progression Validator	Chord progression	Boolean (valid/not valid)	Determines whether a progression between two chords verifies all music-theoretical rules (cf. Appendix I). Used during *atomic evolution* phase (cf. Section 4.4.3)	*MC*
Chord Identifier	Note (in context key)	Chord type and root	Returns the chord type and root for a given note of a musical key. Used during the *atomic evolution* phase (cf. Section 4.4.3)	
Chord Building	Note (serving as chord/key root)	Chord	Builds chords of all types, following a given root note. Used during the *thematic evolution* phase (cf. Section 4.4.3)	
Relative and Neighbor Key determination	Key	Relative major/minor key and neighbor keys	Used to perform modulation/demodulation mutation operation in the *mutation* phase (cf. Section 4.4.4)	
Passing note	Two consecutive chords	Two Notes (so-called *passing*)	Identifies the highest notes in both chords that are in the same key and less than an octave apart. Used to perform the *passing notes mutation* operation (cf. Section 4.4.4)	

4.3. Music Sentiment Learner (*SL*) module

The *SL* module houses the core functionality through which MUSEC infers (extracts) the sentiments portrayed in a given musical piece, and serves as the fitness function for the *MC* module (described in Section 4.4). It consists of three main sub-components: i) a non-parametric fuzzy classifier coupled with ii) a music similarity function evaluator, as well as iii) an extensible training set of MIDI pieces required to train the classifier.

4.3.1. Non-Parametric Fuzzy Classifier component

It consists of a supervised learning algorithm allowing to compute sentiment scores for new incoming pieces based on their similarities with pieces it already learned, without making preliminary assumptions or adding constraints about the form of the mapping function (in contrast with parametric learners which mapping function needs to comply with a fixed set of parameters, cf. Section 3.2.3). We adopt the fuzzy *k*-NN learner (Keller J.M. et al. 1985; Shang W. et al. 2005) in our current system due to its flexibility and effectiveness, yet any other fuzzy classifier could be used, e.g.,

(Abu Arqub O. 2017; Abu Arqub O. et al. 2016; F. Amin et al. 2017; Fahmi A. et al. 2018; Fahmi A. et al. 2019; Fahmi A. et al. 2017). Unlike the traditional crisp *k*-NN algorithm (which classifies data in crisp/distinct categories) (Kotsiantis S. B. 2007), fuzzy *k*-NN produces fuzzy sentiment membership scores (it generates so-called fuzzy categories with fuzzy boundaries, such that an object, i.e., a musical piece, can be part of one category and the other at the same time), which is more in keeping with the nature and subjectivity of sentiments (e.g., a piece of music can express 70% excitement, 20% anger, and 10% happiness simultaneously).

Fuzzy *k*-NN follows an *instance-based* learning paradigm in that it does not perform explicit generalization from training data, but rather refers to its training data at every testing and labeling task (Keller J.M. et al. 1985). This makes it simple and appropriate for our music sentiment inference task, where the bulk of the work consists in evaluating the similarity between music feature vectors w.r.t. a reliable and expressive training set, allowing it to continuously adapt its fuzzy membership scores w.r.t. the number and nature of different input music pieces. The pseudo-code for our *SL* module is shown in Fig. 3. It accepts as input: i) the musical feature vectors of an initial set of sentiment-labeled MIDI pieces, required for training (i.e., learning phase), ii) initial fuzzy *k*-NN configuration factors (including the number and weight factor of neighbors), as well as iii) a new incoming piece's music feature vector; and produces as output: a six-dimensional sentiment vector reflecting the sentiments expressed by the incoming piece. The sentiment vector consists of scaled and normalized scores ($\in [0, 1]$) associated with each of the six primary emotion categories considered in our study (i.e., anger, fear, joy, love, sadness, and surprise). The incoming piece's feature vector is compared with those in the training set (Fig. 3, lines 1-6). Then, sentiment vector scores for the incoming piece are computed based on those of the *k* most similar training pieces identified previously (lines 7-15). These are subsequently normalized in order to obtain scores $\in [0, 1]$ for every dimension (i.e., sentiment category) in the output sentiment vector (lines 16-21).

In our current study, we set the number of nearest neighbors *k* to 3 and the neighbor weight factor β parameter to 2, following general usage in *k*-NN classification (Keller J.M. et al. 1985; Shang W. et al. 2005). Yet, these parameters can be fine-tuned by the user following the nature and properties of the musical pieces utilized, in order to optimize sentiment scoring quality (cf. experimental results in Section 6).

Algorithm: Sentiment Learner (*SL*)

Input: Training Set of musical pieces' feature vectors: ST
Incoming musical piece's Feature Vector: FV_{in}
Configuration parameters: k and β
// k: number of nearest neighbors to consider for score computation
// β: weight factor of neighbors in fuzzy score computation

Output: A 6-valued sentiment vector: SV_{out}

Begin

Initialize a priority queue *pQueue* // used to sort training vectors by similarity
For every feature vector $FV_{Training}$ in ST 1
{ 2
Compute similarity score $Sim_F(FV_{in}, FV_{Training})$ // $\in [0, 1]$ 3
Push $FV_{Training}$ into *pQueue* // with highest priority 4
} 5
6
Initialize a 6-valued vector of sentiment scores to all zeros: $SV_{out}[] = (0\ 0\ 0\ 0\ 0\ 0)$
Initialize a 6-valued vector of normalization scores to all zeros: $SV_{Norm}[] = (0\ 0\ 0\ 0\ 0\ 0)$ 7
// SV_{Norm} is utilized to normalize SV_{out} 8
For i = 1 to k // considering the k most similar neighbors
{ 9
Poll *pQueue* to retrieve 6-valued expert sentiment vector $SV_{Training}$ 10
Retrieve pre-computed similarity score $Sim_F(FV_{in}, FV_{Training})$ for this *pQueue* entry 11
For j = 1 to 6 // Cover all six sentiments 12
{ 13
$SV_{out}[j] = SV_{out}[j] + \frac{SV_{Training}[j]}{1 - Sim_F(FV_{in}, FV_{Training})^{\beta - 1}}$ 14
15
// a higher β provides more weight to nearer (more similar) neighbors
$SV_{Norm}[j] = SV_{Norm}[j] + \frac{1}{1 - Sim_F(FV_{in}, FV_{Training})^{\beta - 1}}$ 16
} // summing weight scores separately for later normalization
} 17
18
For j = 1 to 6
$SV_{out}[j] = \frac{SV_{out}[j]}{SV_{Norm}[j]}$ // Normalizing output sentiment vector scores 19
20
Return SV_{out}
21
End

Fig. 3. Pseudo-code of *SL*'s main algorithm

4.3.2. Music Similarity Evaluation component

Music feature similarity evaluation allows the fuzzy *k*-NN component to perform its estimations. It accepts as input the music feature vectors of two MIDI files (parsed using the *FP* module) and returns a similarity score ∈ [0, 1] highlighting their similarity or divergence (0/1 designating minimum/maximum similarity respectively). Similarity computation is done through the aggregation of individual feature-level similarity scores into an overall similarity score. As for the aggregation function, various mathematical formulations can be utilized, among which the *maximum*, *minimum*, *average* and *weighted sum* functions. Here, we exploit the weighted sum function as it provides flexibility in performing similarity evaluation, adapting the process w.r.t. the user's perception of music feature similarity. Given two musical pieces p_1 and p_2 with parsed music feature vectors FV_1 and FV_2 respectively, and given F the set of 7 music features considered in our study where $f \in F$ stands for every individual feature:

$$Sim_F(p_1, p_2) = Sim_F(FV_1, FV_2) = \sum_{f \in F} w_f \times Sim_f(FV_1, FV_2) \quad \in [0,1]$$
$$\text{where} \quad \sum_{f \in F} w_f = 1, \quad \forall w_f \in [0,1], \text{ and } \forall\, Sim_f(FV_1, FV_2) \quad \in [0,1] \qquad \textbf{(1)}$$

Five of the seven features used in MUSEC are scalar (including: *note density*, *note onset density*, *piece tempo*, *average pitch*, *average intensity*) and can be compared using typical *Jaccard* (or any other scalar vector) similarity evaluation measure. Yet, the remaining two (symbolic) features: *dominant key* and *chord progression*, require dedicated and more sophisticated similarity measures.

Dominant keys can be compared using as reference the music-theoretical *circle of fifths* (Danhauser A. 1994) (shown in Fig. 4) highlighting the relationships between all musical keys, which we include as part of MUSEC's *KB* module. While key comparison is performed intuitively in music-theory as the separation between keys in the circle of fifths, we concretize it mathematically using an adaptation of the shortest path problem. Given the graph representation of the circle of fifths in Fig. 4, we evaluate the similarity between two keys *A* and *B* using typical graph navigation techniques, namely *Dijkstra*'s shortest path algorithm (Cormen T.H. et al. 2009), as the inverse of the minimum distance path between the nodes representing the two keys being compared. To emphasize the difference between major and minor keys (where minor keys are more similar to each other than to major keys, and vice versa), we assume that edges connecting nodes representing keys of the same type (both minor, or both major) have cost =1, whereas edges connecting a major (minor) key node with a minor (major) key node have a higher cost =2. As a result, the maximum possible distance between any pair of keys = 8, which is obtained when the two keys being compared are one major-minor edge and 6 same-type edges apart. Consequently, the similarity between two dominant keys *X* and *Y* is computed as the inverse of normalized distance:

$$Sim_{DomKey}(X,Y) = 1 - Dist_{DomKey\_Norm}(X,Y) \quad \in [0,1] \qquad \text{where } Dist_{DomKey\_Norm}(X,Y) = \frac{Dijkistra(X,Y)}{8} \qquad \textbf{(2)}$$

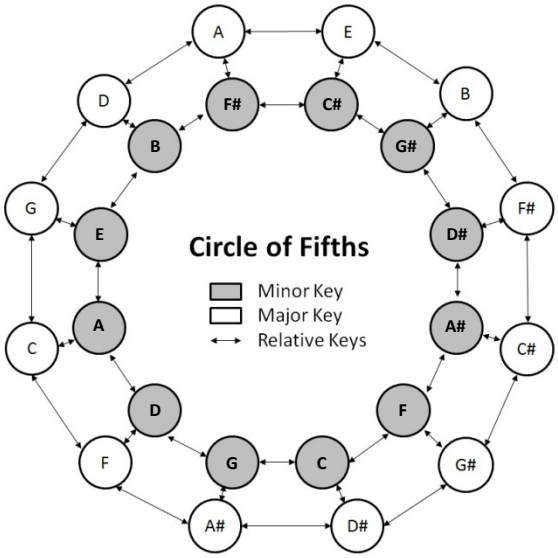


Fig. 4. *Circle of fifths* graph representation

As for *chord progressions*, we compare them using the *Tonal Pitch Step Distance* (TPSD) approach developed in (W. Bas de Haas 2008) and refined in (Bas De Haas W. et al. 2008; Bas De Haas W. et al. 2013). Following TPSD, the chord progressions to be compared are first converted into sequences of distance values (i.e., distance series), consisting of the distances between every chord in one piece and the other piece's root chord. The distance between

two chords is evaluated using a 5-layered approach, where the layers represent progressively more complete representations of the chords and their context key (e.g. Layer 1 only consists of the chord's root, Layer 3 consists of all the chord's notes, and Layer 5 includes all 12 chromas within a musical octave) producing integer distance values $\in [0, 13]$, where 0 indicates total similarity and 13 complete divergence. Consequently, overall chord progression distance is calculated by cycling the shorter progression (evaluated in terms of duration) over the longer one, and computing a *difference series* between the two series at every cycle. The cycle yielding the smallest total difference is selected. Though this cycling allows comparing chord progressions in a more complete manner, it imposes a severe computational overhead (Bas De Haas W. et al. 2008)[1]. To remedy this, TPSD can be applied on the beginning of both progressions being compared (Bas De Haas W. et al. 2013)[2], thereby performing the comparison in average linear time[3]. We adopt the latter (linear time) approach in our study, and evaluate the distance between two chord progressions as the average distance over the difference series, normalized by 13 (i.e., the maximum possible TPSD value). Consequently, similarity is evaluated as the inverse of the normalized distance. More formally, given two chord progressions cp_1 and cp_2:

$$Sim_{ChordProg}(cp_1, cp_2) = 1 - Dist_{ChordProg\_Norm}(cp_1, cp_2) \in [0,1] \quad \text{where } Dist_{ChordProg\_Norm}(cp_1, cp_2) = \frac{TPSD(cp_1, cp_2)}{13} \quad \textbf{(3)}$$

4.3.3. Training Set

The training set forms the basis through which MUSEC's *SL* module makes estimations, providing the "expertise" this module uses to infer the incoming pieces' sentiment scores. The quality of the training set is also an essential part of overall system development since noisy or inaccurate training examples would produce inaccurate learning. Given the lack of readily available sentiment-annotated MIDI files, we have developed our own training set, which consists of 40 real musical pieces annotated with the help of 30 human testers via dedicated online surveys[4], as well as 80 synthetic MUSEC compositions which were rated in-house by MUSEC's development team. The resulting 120 piece set[1] is diverse and almost evenly distributed among all six primary emotions considered in MUSEC, with almost 20 pieces targeting every emotion category. Details regarding the training set construction process are provided in Section 6.3.1.

Considering synthetic pieces as part of our training set highlights MUSEC's self-learning functionality, depicted in Fig. 5. It allows the user to easily expand the training set by feeding back to the MUSEC's *SL* module some of its own compositions (i.e., the best and most sentiment-expressive ones, following user judgments), where every composition – with its associated sentiment vector – provides an added training example to the expanding training set. Here, combining both real and synthetic pieces to expand the training set would allow the system to gain more insight into new ways it can meet the given sentiment scores. For instance, a piece provided by the user to train the *SL* module could have a peculiar feature vector, which the system would otherwise have not sought to emulate in producing sentiment scores. Through its self-learning functionality, the system could: i) adapt to its users' idiosyncrasies and inclinations (through their sentiment ratings of real/synthetic pieces) while ii) learning new ways to potentially produce a given sentiment target (through synthetic compositions produced by the system itself, which are evaluated by users, and then fed back to the system for training).

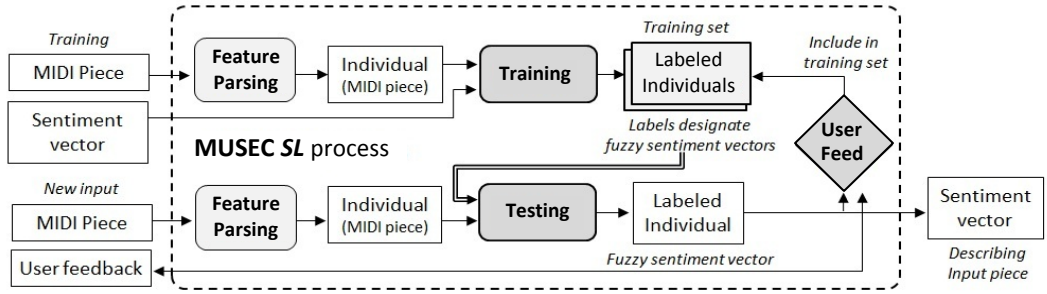


Fig. 5. Simplified activity diagram describing *SL*'s overall process and its *self-learning* functionality

---

[1] It requires $O(n \times m \log(n+m))$ where n and m designate the number of chords in the two pieces (chord progressions sequences) being processed.

[2] Consider two chord progression sequences A and B, consisting of chords $A_1, A_2, \ldots, A_m$ and $B_1, B_2, \ldots, B_n$ respectively. Without loss of generality, consider the case where $m < n$. Following the standard TPSD algorithm in [6], the shorter sequence is compared with the longer one at every position, e.g., $A_1, \ldots, A_m$ versus $B_1,\ldots,B_m$, then $A_1, \ldots, A_m$ versus $B_2,\ldots,B_{m+1}$, and so forth until $A_1, \ldots, A_m$ versus $B_{n-m},\ldots,B_n$. Then the comparison yielding the smallest difference is selected as the final similarity (or distance) value. With the more efficient version of the TPSD algorithm in (Bas De Haas W. et al. 2013), the chord progression sequences are only compared from their starting positions, e.g., $A_1, \ldots, A_m$ is only compared with $B_1,\ldots,B_m$, and that score is utilized as the chord progressions similarity (distance) score. Despite this linear relaxation of the original algorithm, TPSD computation remains the most expensive among all other feature similarity computations put together (cf. experiments in Section 6.2.2).

[3] To the expense of a potential loss of precision when processing long musical pieces (consisting of a large chord progression sequences).

[4] Available online at: http://sigappfr.acm.org/Projects/MUSEC, *SL* survey form #1 (first part, 24-pieces), #2 (second part, 8-pieces), and #3 (third part, 8-pieces), along with the resulting sentiment-labeled dataset.

4.4. Music Composer (*MC*) module

MUSEC's *MC* module is the "creative" mind through which our system produces novel music. Guided by the system's experience in music sentiment expression (provided by the *SL* module) and its understanding of music-theory (provided by the *KB* module), *MC* utilizes evolutionary computations to: i) produce several candidate pieces that can possibly reflect target sentiment scores, and then ii) select the most promising candidates to further develop in later iterations, toward producing more sophisticated pieces which meet the user's target sentiment scores.

4.4.1. Overview

MC's overall activity diagram is depicted in Fig. 6. It adopts the *Evolutionary-Developmental* (or *Evo-Devo*) model approach in performing music composition: starting with "simple" individuals (representing simple MIDI musical chunks, made of a couple of notes each, with the notes' properties), and evolving them into longer and more sophisticated ones, whilst keeping only those which seem "fittest" and most diverse, where fitness in our case promotes sentiment-expressive music. To evolve its population, our composer leverages music-theoretical knowledge from MUSEC's *KB* module to produce several musical continuations for every individual (piece), thereby producing offspring (pieces) that extend their parents. Offspring are then exposed to a battery of 18 dedicated mutation operators (e.g., *trille*, *staccato*, *repeat*, etc.) so as to maximize population variability, diversity, and sophistication. Consequently, a two-step *fitness-variability* selection (trimming) of offspring is conducted following two criteria: i) *fitness*: selecting pieces based on their relevance w.r.t. the user's target sentiment vector scores (evaluated through the *SL* module), and ii) *variability*: selecting the fittest (most sentiment-expressive) pieces that are most dissimilar from each other (to promote diversity in the final population). The most sentiment-expressive and varied pieces survive, while the others perish. Following the trimming phase, a final population is produced, and is then fed back to the dataset of individuals (as a new generation) after which the evolutionary process starts anew (evolution, mutation, and trimming). This iterative process repeats until the user is satisfied with the final compositions.

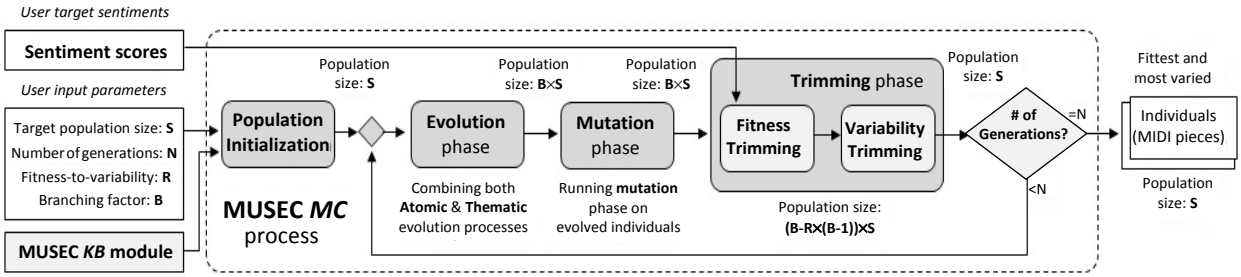


Fig. 6. Simplified activity diagram describing *MC*'s *evolutionary composition* process[1]

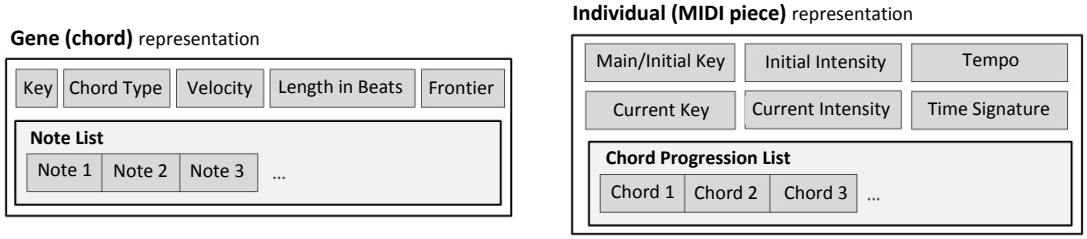
In addition to specifying the target sentiment scores to be portrayed in the compositions, the user can also control the composition process by altering several *MC* parameters, namely the *probability of different mutation operators* being applied, the *fitness-to-variability ratio* controlling the selectivity of both fitness and variability trimming, the *size of the candidate population*, the *number of offspring* produced per individual (referred to as the *branching factor*), and the *number of generations* (iterations) the evolutionary process goes through before halting.

The following sub-sections describe *MC*'s main components including: individual (MIDI piece) representation, evolution, mutation, and trimming mechanisms.

4.4.2. Individual MIDI Piece Representation

A MUSEC *individual* represents a candidate MIDI piece being developed to meet the user's specified target sentiment scores. Its core components are the chords that make up the corresponding MIDI piece. Here, we view chords as *genes* in evolutionary algorithms, i.e., the most essential constituents of an *individual*'s representation. The *genes* are affected by most of *MC*'s mutation operators (cf. Section 4.4.4), such that their expression defines the melodies and notes that make up the *individual*. Fig. 7.a and Table 4 respectively provide a simple graphical representation of a MUSEC *gene* structure and a description of its various properties.

[1] The population size transformation through the evolutionary process, taking into account user input parameters (namely the *branching* factor *B* and the *fitness-to-variability* ratio *R*), is described in detail in Section 4.4.5.



a. *Gene* (chord) representation **b.** *Individual* (MIDI piece) representation

Fig. 7. Simplified graphical presentations of a MUSEC *MC gene* (a) and *individual* (b)

Table 4. Properties of a MUSEC *gene* (chord) representation

Property	Meaning	Description
Length in beats	The number of beats the chord occupies	This value can change as a result of various mutations (e.g., *extend*, *steal*, *compress*, cf. Section 4.4.4).
Frontier Array	It consists of 3-to-4[1] MIDI pitches from the chord's notes	The frontier is used to compute valid musical chord continuations to the current chord during *MC*'s evolution phase (cf. Section 4.4.3), following the rules of music theory (coded in the *KB* module). Note that a chord's frontier is static and is not affected by *MC*'s mutation operations.
Note Array	A dynamic-size array containing all notes to be played as part of the chord	Unlike the *static* frontier, notes in this array are *dynamic*: affected by *MC*'s mutation operators which produce variations in performing the same chord, modifying the notes' ordering, timing, and length within the chord. Additional notes can also be added, and existing ones can be removed from the chord.
Chord Type	The chord type (*major*, *minor*, *dominant seventh*, etc.)	The root and chord type which identify the chord.
Velocity	The velocity (intensity) at which the chord's notes are played	Velocity is equal to the individual's intensity at the time of the chord's insertion.
Key	The chord's key	It also represents the individual's key at the time of the chord's insertion

Consequently, we define the MUSEC *individual* as a dynamic and extensible composition of *genes*. It is constructed based on several musical properties some of which change and grow as the individual evolves, allowing to produce an increasingly larger and more sophisticated musical piece while ensuring that musical conventions and rules are respected in the composition process. Fig. 7.b and Table 5 respectively provide a simple graphical representation of a MUSEC *individual* structure and its various properties.

Table 5. Properties of a MUSEC *individual* (MIDI piece) representation

Property	Meaning	Description
Main Key	The main key that the composition follows	The individual starts in this main key but can leave it due to a modulation (one of MUSEC's mutation operators), and can later return to it following another modulation.
Current Key	The key that the composition is currently using	When the individual modulates to another key, the main key will continue to indicate the original key, while the current key will reflect the new key. This property is mainly used to compute the continuation to a given piece, following the key it is currently using.
Starting Intensity	The starting MIDI velocity used in the individual	Through mutations, a piece's velocity can vary over time (i.e., getting louder or calmer).
Current Intensity	The current MIDI velocity used in the individual	It allows tracking the MIDI velocity applied at the current point in the individual.
Tempo	The overall speed and rhythm of the piece	Expressed in BPM, this value can change over time due to mutations that affect the individual.
Time Signature	The rhythmic structure of a piece expressed in terms of *time signature*	It commonly varies between binary (2/4), ternary (3/4), and quaternary (4/4), among other signatures[2].
Chord Progression List	The list of chord progressions in the musical piece	The complete sequence of chords (and their musical realizations) that make up a musical piece

[1] Following music theory (Danhauser A. 1994), a chord can be defined as having 3, 4 or more musical pitches in its fundamental definition, referred referred to as fifth, seventh, or ninth chords. Yet we only handle 3-to-4 pitch chords in our gene representation since larger chords can be represented using simpler fifth or seventh chords. This allows to avoid musical dissonance in composition, which could occur when larger chords are considered.

[2] We adopt 4/4 as the default time signature in our current system implementation.

As a result of this flexible structure at both the *gene* and *individual* levels, the pieces generated by MUSEC's *MC* module can take a wide range of varying and sophisticated forms, allowing for a varied expression of the user's input sentiments.

4.4.3. Population Initialization and Evolution

When the *MC* module is called upon to compose a musical piece (in order to express the user's target sentiment vector scores), it first starts by creating an initial (seed) population, consisting of a number of random initial individuals with random properties, so as to have as varied a population as possible (in terms of key, tempo, and starting intensities). During the ***population initialization*** phase, a single *gene* (chord) is introduced to every *individual*'s chord progression list, consisting of the root chord of the individual's key, added in its most basic form (i.e., in its root position, 1 beat long, with all its notes played simultaneously). As a result, the initialization phase produces a number of relatively short (1-beat), basic (root chords), but extremely heterogeneous musical individuals, ready to evolve into more sophisticated ones. Note that the size of the seed population can be chosen by the user (we initially set it to 50 individuals, which produced satisfying results in our experiments, cf. Section 6.4).

Once a population of individuals has been initialized, it goes through the ***evolution*** phase, in which every individual in the population grows more sophisticated by virtue of one or more added genes (chords). In this phase, we consider two evolution modes: *atomic evolution*, and *thematic evolution*.

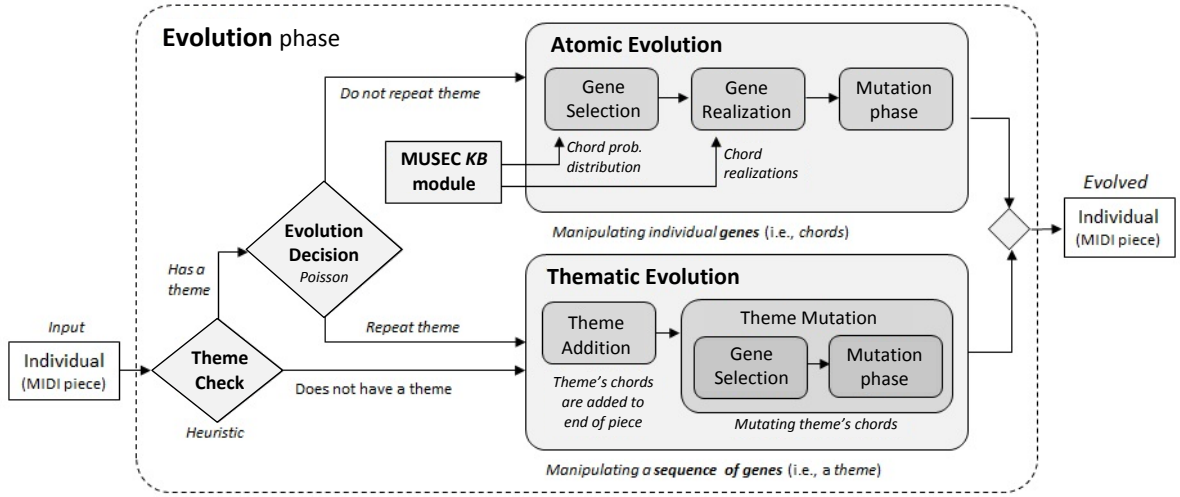


Fig. 8. Simplified diagram describing *MC*'s *evolution* phase

1. ***Atomic Evolution*** consists of adding a single gene, i.e., a single chord, to the individual. It selects the chord to be added based on music-theoretical grounds, following MUSEC's *KB* rules. More in particular, using the individual's current key, the *atomic evolution* process identifies the key's main seven chords (one per key note) and randomly selects one chord to realize based on a pre-defined chord probability distribution[1]. Then, a recursive function titled *Chord_Realizations* (cf. Appendix I) identifies all possible music-theoretic valid realizations of the selected chord, in terms of inversions and notes, using the individual's last chord's frontier from which a realization is randomly selected. The identified chord is packaged as a MUSEC gene structure to be inserted into the candidate individual. The newly inserted gene is then subjected to various mutations during the evolutionary composer's mutation phase (described in the following section), before obtaining the final version of the candidate individual.

While *atomic evolution* allows to produce new melodies, by creating new chord patterns and completing existing ones, nonetheless, it is an unstructured and pseudo-random[2] evolutionary approach, which, applied alone, produces unstructured and somewhat chaotic compositions. To avoid the latter and produce more structured music, inspired by how chord patterns occur and repeat in human compositions, we extend the atomic evolution process to produce chord patterns and pattern repetitions in the composition process. We refer to the latter as *thematic evolution*.

2. ***Thematic evolution*** repeats a given individual (MIDI piece)'s gene (chord) pattern so as to emulate human composers' concept of a *melodic theme*, which is extremely powerful and widely adopted in music composition (Danhauser A. 1994). A melodic theme is defined as the melodic subject of a musical composition, usually announced in the first measures of the piece. However, identifying the melodic theme in order to repeat it is not a simple task, given the diverse forms in which a theme can manifest itself in music. To remedy this, we adopt a

[1] In our current implementation of *MC*, we hard-coded the chord probability distribution (through which a chord is selected) based on empirical sampling from our training set. Yet, learning the chord probability distribution can be a research project in and of itself, and can entail different composition styles. For instance, the distribution could be learned from a composer's composition corpus, to produce pieces following the composer's own style (which we further discuss as an ongoing work in Section 8).

[2] Randomness is guided by MUSEC's *KB* music-theoretic rules.

simplifying heuristic, through which a ***melodic theme*** is defined as being the first chord progression starting and ending with an individual's root chord. Though simple, this assumption holds for a large number of compositions, and has produced satisfactory results in our experimental evaluation (cf. Section 6.4). The melodic theme's chords are added to the end of the piece, where mutations are applied to the repeated melodic theme's chords such that each repetition becomes a variation of its original version, much like how human composers vary their choruses slightly between different repetitions, allowing to avoid a redundant repetition of themes.

A functional diagram depicting *MC*'s evolution mechanism is shown in Fig. 8. The decision to perform *atomic evolution* or *thematic evolution* is made following a scan of the individual's chord progression. If no *melodic themes* are found (i.e., if there is no first chord progression starting and ending with an individual's root chord), then atomic evolution is performed (in hope of producing a new melodic theme). Otherwise, if a melodic theme exists already, then a random (Poisson) decision is made to run either thematic evolution or atomic evolution. The evolution process is repeated B times, where B denotes *MC*'s branching factor, such that every individual in the current population produces B offspring. The value of B can be chosen and fine-tuned by the user (we initially set B=5, which is a rather common choice in evolutionary computation literature, e.g., (Marques M. et al. 2000; Whitley D. et al. 2012)).

Given a branching factor B and an individual population size S, the evolution phase produces $S_{Evo}= B \times S$ individuals to form the new generation.

4.4.4. Mutation Phase

In order to compose diverse and sophisticated music, *MC* relies on several music-theoretical mutation operations. These mutations affect the realization of genes (chords), be it in terms of the order of note onsets, the decorations applied to these realizations, or the intensity and duration of the chord being played, among other features. Beyond gene realizations, we also introduce mutations that can affect the entire individual (MIDI piece), in particular its dynamic features such as: *tempo*, *current intensity*, and *key* (via modulations and demodulations).

A diagram depicting the mutation process, as part of *MC*'s overall evolutionary process, is shown in Fig. 9. A mutation operator op_i is performed by evaluating its frequency of occurrence $Freq(op_i)$ (i.e., the number of times it appears, over the total number of mutations) in the individual being composed. Occurrence frequencies are compared with a vector of *mutability thresholds* where every threshold $Thresh_{Op_i}$ value is associated with a given mutation operator, such that the mutation is randomly chosen and applied to the individual until reaching the designated threshold. Mutation occurrence frequencies are updated at the end of every mutation phase, considering the new mutation that took place on the new added gene (or on the entire individual). When the mutation operator's occurrence frequency in the individual being composed has reached its maximum threshold, it will not be considered among potential mutations in the subsequent mutation phase. In our current study, we allow users to manually set the mutability thresholds' vector, allowing them to control (or promote) variety in the produced offspring. In addition, mutability thresholds can be altered or fine-tuned to define or distinguish between different composition styles (which we further discuss among future research directions in Section 8).

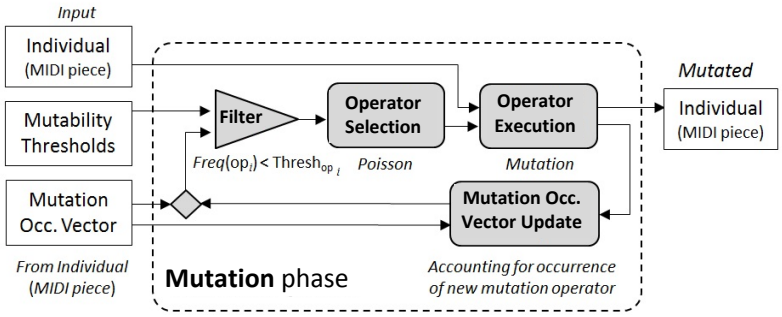


Fig. 9. Simplified diagram depicting *MC*'s *mutation* process

MC currently offers 18 different mutation operators, which we briefly describe below in Table 6 (a more detailed description of every operator is provided in Appendix II). Additional operators can be added later to allow even more variability and sophistication in the produced compositions.

4.4.5. Trimming Phase

Following *MC*'s *evolution* and *mutation* phases comes the *trimming* phase, which selects the offspring that it deems "fittest" and "most varied" to survive into the new generation (cf. evolutionary mechanism depicted in Fig. 6). Among the $S_{Evo}=B \times S$ individuals produced by the evolution phase (where B is the branching factor, and S is the population size), the trimming phase selects the S fittest individuals from S_{Evo} to survive into the next cycle, effectively "killing" $(B-1)\times S$ individuals. At the end of this phase, the evolutionary composer will have the exact same population size that it had at the beginning of its evolutionary cycle.

To perform the trimming of $(B-1)\times S$ individuals, we make use of two criteria evaluated consecutively in a two-step trimming process: i) *fitness*, which measures the individual (MIDI piece)'s relevance w.r.t. the user-specified

target sentiment vector, and ii) *variability*, which assesses an individual's variation (distinctiveness) from to its peers. *Fitness trimming* is performed as the first trimming step (since it highlights MUSEC's primary objective: expressing sentiments in music), followed by *variability trimming* as a subsequent phase (promoting diversity in the population).

Table 6. Brief description of MUSEC *MC*'s mutation operators

Target	#	Operator	Description
Single chord (gene)	**1**	***Trille***	It retrieves the next note above the highest note in the chord (*gene*), and then alternates rapidly between the two notes over the first half-beat of the chord being mutated.
	2	***Staccato***	It alters all the chord's notes, by reducing the duration of every note to an eighth beat, so as to have them played detached and separated from each other.
	3	***Repeat***	It divides the current chord duration into two (based on a random decision), and then repeats the notes being played as part of the original chord a second time: adding a copy of all notes in both divisions.
	4	***Compress***	It shrinks the chord's overall duration to a certain (randomly chosen) percentage of its original duration, thus raising overall piece note and note onset density.
	5	***Extend***	It extends the duration of a chord to a certain (randomly chosen) percentage of its original duration, thus lowering piece and note onset densities. It can be viewed as the symmetric counterpart of *compress*.
	6	***Silence***	It extends the duration of a chord to a certain (randomly chosen) percentage of its original duration. Yet, unlike from the *extend* operator, it does not extend the notes themselves, but rather preserves their original durations and instead creates a silence at the extended part of the chord.
	7	***Silence Suspension***	It identifies the note realizations of a chord's frontier notes (its root, third, and fifth), and then randomly chooses one of them and delays its entry by a quarter-beat, thus increasing note onset density.
	8	***Progressive Entrance***	It makes a chord's frontier notes enter (be played) progressively (in sequence), by randomly choosing a starting distribution and spreading over a half-beat duration, which indicates the beat timing at which every frontier note should be played.
	9	***Nota Cambiata***	It decorates the highest note of a chord by preceding it with three other notes in its key: i) a third above, ii) a second above, and iii) a second below it in its chord's key, assigning a random duration to each of these notes following the same logic adopted by the *progressive entrance* operator. The decorated note is delayed by half a beat to accommodate the new decoration.
	10	***Appoggiatura***	It decorates the highest note of a chord by preceding it with an adjacent note in its key: typically the note a second above or a second below it in the given key. Similarly to the *Nota Cambiata* operator, the decorated note is delayed by half a beat to accommodate the new decoration.
	11	***Double Appoggiatura***	It is a more sophisticated version of the *appoggiatura* mutation, where the decorated note is preceded with both its adjacent notes, in a randomly chosen order (i.e. which note is played first), and a duration distribution (using eighth beat time units) for the two added notes over the half-beat they are allocated to fit the decoration.
	12	***Octava***	It shifts the pitches of the chord's notes up or down by an octave (i.e. adds/subtracts 12 to the said notes' MIDI pitches). The choice of octave jump (up or down) is stochastically governed by the current average pitch of the chord, such that chords with a lower average pitch are likelier to be shifted up by an octave, and vice versa.
Two consecutive chords (adjacent genes)	**13**	***Tempo Steal***	Unlike the previous operators, *tempo steal* affects two chords, rather than just one. It selects two adjacent (consecutive) chords such that one "steals" a certain duration in beats from the other. The stolen duration value is a certain random percentage of the duration of the chord to be stolen from.
	14	***Passing notes***	Also applied on two consecutive chords, it checks the highest notes in both chords to verify whether they are in the same key and less than an octave apart. If so, the notes are added in sequence to the end of the first chord, considering a duration distribution allocating duration chunks following: total duration divided by the number of passing notes.
	15	***Anticipation***	Also applied on two consecutive chords, it identifies the highest note of the second chord and inserts it in the final half-beat of the first chord, thus emulating the music-theoretic concept of *anticipation*, and increasing both note density and note onset density.
Whole piece (individual)	**16**	**Tempo change**	It affects the whole individual (MIDI piece) by changing its overall tempo: in increments or decrements of 4 BPM (Beats Per Minute). The increase/decrease decision is made stochastically following the piece's current tempo, such that pieces that are slower are likelier to speed-up following this mutation and vice-versa.
	17	**Intensity change**	It changes the individual (MIDI piece)'s current intensity value (velocity) in steps of 20 (in the 0-to-127 MIDI intensity range), such that the increase/decrease decision is made stochastically following the piece's current intensity, where pieces that are of lower intensities are likelier to become louder following this mutation, and vice-versa.
	18	**Modulation**	It changes the individual (MIDI piece)'s current key to one of its neighbor keys (i.e. keys with which it shares connected edges) in the *circle of fifths* (cf. Section 4.3.2). It checks the last chord in the piece and identifies potential destination keys. If more than one alternative is possible (following the *circle of fifths*), then the operator randomly selects the destination key. The piece's current key is then changed to the new key.

To do so, we introduce a *fitness-to-variability ratio*, noted R, that specifies how much of the overall trimming of $(B\text{-}1)\times S$ individuals is performed in each step:

$$R = \frac{F}{F+V} \quad \text{where } F \text{ and } V \text{ designate the numbers of individuals trimmed based on } \textit{fitness} \text{ and } \textit{variability} \qquad \textbf{(4)}$$

respectively, such that $F+V$ represents the total number of trimmed individuals.

A ratio of R=1 indicates that trimming is completely based on *fitness*, while a ratio of R=0 indicates that it is solely based on *variability*. More specifically, for a given fitness-to-variability ratio R, the *fitness* criterion trims $F = R \times \big((B-1) \times S\big)$ individuals, hence shrinking the population size from $S_{Evo}=B \times S$ to $S_{Fit}= \big(B - \big(R \times (B-1)\big)\big) \times S$ individuals. Consequently, the variability *criterion* trims $V = (1-R) \times \big((B-1) \times S\big)$ to bring the population size down to S. The value of ratio R is chosen by the user (we adopt an initial value of 0.7, i.e., 70% of individuals are trimmed based on *fitness*, and 30% are trimmed based on *variability*[1]). The change of population size due to the trimming mechanism is represented in *MC*'s activity diagram in Fig. 6.

Algorithm: Fitness Trimming
Input: Set of individuals: I // a total of $S_{Evo}= B \times S$ individuals generated in the evolution phase
User target sentiment vector: SV_{user}
Output: List of survivors: $List_{Fit}$ // of cardinality $S_{Fit}=(B - (R \times (B - 1))) \times S$ ordered by *fitness*

```
Begin
  Initialize an ordered list List_Fit = ∅          // used to sort individuals by fitness      1
  Initialize a priority queue pQueue                                                          2
  For every individual i in I                                                                 3
    Run through MUSEC-ML to obtain sentiment vector SV_i                                      4
    Compute Sim_PCC(SV_i, SV_user)                                                            5
    Push i into pQueue following Sim_PCC as priority level                                    6
  Repeat                                                                                      7
    Pull individual i from pQueue                                                             8
    Add i to List_Fit                                                                         9
  Until reaching |List_Fit| = S_Fit                // number of required fitness trimming survivors  10
  Return List_Fit                                                                             11
End
```

a. Pseudo-code of *fitness trimming* process.

Algorithm: Variability Trimming
Input: List of individuals: $List_{Fit}$ // consisting of $S_{Fit}=(B - (R \times (B - 1))) \times S$ survivors of *fitness trimming*
// ordered following their *fitness* (from most to least fit)
Output: List of survivors: $List_{Out}$ // of cardinality $=S$, ordered following both *fitness* and *variability*

```
Begin
  Initialize an ordered list List_Out = ∅          // used to sort individuals by dissimilarity   1
  Select fittest individual i from List_Fit        // one that is ordered first in I              2
  Add i into List_Out as first survivor, and remove it from List_Fit                             3
  Repeat                                                                                         4
    For every individual j in List_Fit                                                           5
      Compute average feature similarity between j and all survivors in List_Out                 6
    Select individual k with lowest average feature similarity and highest fitness               7
    Add k into S as next survivor, and remove it from List_Fit                                   8
  Until reaching |List_Out| = S                    // required number of survivors               9
  Return List_Out                                                                               10
End
```

b. Pseudo-code of *variability trimming* process.

Fig. 10. Pseudo-codes of *MC's* two-step trimming process: *fitness* (a) and *variability* (b)

The pseudo algorithms of our *fitness trimming* and *variability trimming* processes are shown in Fig. 10. On one hand, the fitness trimming process (cf. Fig. 10.a) accepts as input the set of S_{Evo} individuals produced by the evolution phase, as well as the target user sentiment vector. It then evaluates the fitness of every individual using MUSEC's *ML* module (lines 1-4), which accepts as input the candidate individual (MIDI piece) and produces as output the sentiment vector expressed by the individual. The produced sentiment vector is then compared with the user's target sentiment vector (using *PCC*[2] vector similarity) in order to quantify their closeness (lines 5-6). Individuals which sentiment

[1] Emphasizing sentiment expression, while also promoting diversity.

[2] *Pearson Correlation Coefficient*. Note that any other vector similarity measure (such as *cosine* or *dice*) could have been used. We adopt *PCC* here since it is commonly utilized in the literature (Abbasi A. *et al.* 2008; O'Connor B. *et al.* 2010).

vectors are most similar to the user's target vector are chosen as candidates for survival to the next cycle. These amount to $S_{Fit} = \left(B - \left(R \times (B-1)\right)\right) \times S$ survivors, filtered from the $S_{Evo}=B \times S$ population through a sentiment similarity-based priority queue (lines 7-11), to be then fed as input to the subsequent *variability trimming* step.

On the other hand, the *variability trimming* process (cf. Fig. 10.b) accepts as input the list of S_{Fit} candidates identified in the fitness trimming phase, and filters out the least varied individuals among them. It allows identifying musically different (divergent) pieces, by comparing the candidate individuals' music feature vectors in order to select the most dissimilar ones, thus encouraging novelty and variation in the composition process. To do so, it first selects the fittest individual from the survivors of the fitness trimming phase (lines 1-2), and then compares it with the other survivors (line 3-6) in order to select the one with which it shares the minimum music feature vector similarity (line 7, cf. feature similarity evaluation in Section 6.2). The process is repeated iteratively, comparing each of the fittest remaining candidates with the already selected individuals, in order to select the (fittest) candidate that is most dissimilar (i.e., having the minimum average feature similarity) from the selected ones, until reaching the required number of candidates S (lines 8-10). Remaining survivors are discarded.

Finally, the user can choose to consider the whole population of S fittest and most varied individuals as the result of the composition process, in which case the system would be producing as output: S different pieces after every run. Nonetheless, in our case, we consider one single piece as the output of every run: corresponding to the single most fit (i.e., most sentiment expressive piece) among the S produced individuals[1].

5. Complexity Analysis

The overall time complexity of our approach simplifies to $O\left(N \times \left(\left(B \times S \times T \times N\right) + S^2\right)\right)$ where N represents the number of generations (number of iterations of the evolutionary process), B the branching factor (number of offspring per generation), S the population size (number of pieces maintained at the end of every evolutionary iteration), and T the size of the training set (used by the learner to infer sentiment scores). It is evaluated as the sum of the complexities of the main modules of the MUSEC framework, which mainly amounts to the complexity of the *MC* module (since it invokes the other modules within its execution process):

- *Feature Parsing* (*FP*): simplifies to $O(n + b)$ time where n represents the number of notes, and b the number of beats in a MIDI piece, and can be evaluated as the sum of the complexities of parsing all (symbolic and frequency domain) features from the input music (MIDI) piece. Initially, the *FP* module identifies all notes n from the piece by iterating through the piece's MIDI messages and identifying *Note On*/*Note Off* message pairs. This operation is linear w.r.t. the size of the input piece, i.e., $O(|p|)$ which comes down to $O(n)$ time. Tempo parsing is performed through reading the MIDI piece's tempo meta message (cf. Section 4.1), and is performed in parallel with note parsing while the MIDI piece's messages are being read. The same goes for parsing *note density, note onset density*, *average pitch*, *average intensity*, and *dominant key* features which are computed in parallel with note parsing, performing weighted average computations on the input piece's note properties (such as duration and pitch). However, the *chord progression* feature parsing does not depend on the number of notes, but rather on the number of beats, b, in the input piece, due to the beat-based nature of the chord parsing heuristic (cf. Section 4.1.2), and thus requires $O(b)$ time. Hence, *FP*'s complexity comes down to $O(n + b)$ time.
- *Sentiment Learning* (*SL*): the complexity of the fuzzy k-NN algorithm, which is the main building block of our *SL* module, simplifies to $O(|cp| \times T)$ time, where $|cp|$ designates the length of the smallest chord progression within two pieces being compared, and T the size (in number of pieces) of the training set. In fact, the algorithm is non-parametric and instance-based, and thus does not require any training time. Training the *SL* component consists in adding an element (a new piece) to its training set, which is done in constant $O(1)$ time. The algorithm's execution (fuzzy scoring) requires $O(|cp| \times T \times k)$ time where k is the number of k-NN neighbors considered when producing the fuzzy sentiment scores:
  - Each similarity evaluation computation between two music (MIDI) pieces requires $O(|cp|)$ since:
    - Computing the similarity between all features, except for *chord progression*, requires constant (near-zero) $O(1)$ time, since they consist in comparing constant size scalar vectors (e.g., comparing *tempo* values, *note density* values, *note onset density* values) while *key* comparison is also done in $O(1)$ by identifying the shortest path within the *circle of fifths* (cf. Section 4.3.2) which is of constant size.
    - As for *chord progression* similarity, it is evaluated using the TPSD algorithm (cf. Section 4.3.2), which runs in average linear time w.r.t. the smallest length between the two *chord progressions* (of the two music pieces) being compared, i.e., $O(|cp|)$ time.

[1] We consider this strategy to be similar to the way some human composers usually write music: producing multiple candidate (trial) pieces, slicing and mixing them up, developing them and making them evolve until reaching a final pool of best candidates, from which the single best candidate is usually adopted as the actual final piece.

- The algorithm compares the input target piece's feature vector with the vectors of every one of the T pieces in the training set, while selecting the top k pieces with the highest similarity scores to compute the target piece's fuzzy sentiment scores accordingly, requiring $O(|cp|\times T\times k)$ time.

Given a fixed k parameter, the algorithm's complexity simplifies to $O(|cp|\times T)$, which in turn comes down to $O(N\times T)$ since chord progression size grows by 1 chord every generation (i.e., $|cp|$ linearly follows N).

- *Music Composition* (*MC*): simplifies to $O\left(N\times\left(\left(B\times S\times T\times N\right)+S^2\right)\right)$ where N represents the number of generations, B the branching factor, S the population size, and T the size of the training set:
  - The *population initialization* phase requires $O(S)$, where S individuals are randomly initialized.
  - The *evolution* phase requires $O(S\times N)$ time: i) *atomic evolution* requires constant $O(1)$ time, whereas ii) *thematic evolution* requires $O(S\times N)$ time since it iterates over every individual among S, and executes once for every added gene, where the length of an individual grows by 1 gene (chord) per generation.
  - The *mutation* phase requires $O(|op|\times S)$, where $|op|$ designates the number of mutation operators in *KB*, while applying every individual operator requires constant $O(1)$ time. Since the number of operators $|op|$ is fixed in *KB*, complexity simplifies to $O(S)$.
  - The *trimming* phase simplifies to $O\left(N\times\left(\left(B\times S\times T\times N\right)+S^2\right)\right)$ time:
    - The *fitness trimming* phase requires $O(B\times S\times T\times N)$, where $B\times S$ individuals (offspring) are evaluated by the *SL* module to predict their estimated sentiment vector, requiring $O((B\times S)\times(N\times T))$ time.
    - The *variability trimming* phase requires $S_{Fit}=\left(\left(B-\left(R\times(B-1)\right)\right)^2\times S^2\right)$ where R is the *fitness-to-variability ratio*, and practically simplifies to $O(S^2)$ since R is supposed to be high (by design, cf. Section 4.4.5) such that most offspring are trimmed during the *fitness trimming* phase[1].

  The *fitness* phase, including both *fitness* and *variability trimming* executed consecutively, is repeated for every generation, leading to an overall $O\left(N\times\left(\left(B\times S\times T\times N\right)+S^2\right)\right)$ time.

6. Experimental Evaluation

Following the implementation of our MUSEC framework[2], we conducted a large battery of tests to assess the performance of its different modules and components. Experiments were carried out on a Toshiba Qosmio X70-A, Intel® Core i7 – 4700 MQ processor with 2.40 GHz clock rate and 32 GB of RAM. A total of 250 music pieces were utilized in the experimental evaluation, including: 100 pieces for feature parsing and similarity evaluation, 120 for sentiment learning (i.e., the training set), and 30 pieces for music composition evaluation. Pieces were assigned to every evaluation task following the different properties that need to be evaluated (e.g., pieces with different lengths in number of notes and number of beats were used to evaluate MUSEC's *FP* module, whereas sentiment-expressive and sentiment-diverse pieces were utilized to train and evaluation the *SL* and *MC* modules). The experimental dataset and test results are available online[2].

6.1. Feature Parsing Evaluation

MUSEC's feature parsing (*FP*) module produces seven (symbolic and frequency domain) features, building a representative feature vector for every input music (MIDI) piece (cf. Section 4.1) to be later processed for sentiment analysis. We considered a sample of 100 music pieces (50 real pieces, and 50 synthetic pieces composed by MUSEC) to evaluate *FE*'s effectiveness and efficiency. All seven features were extracted from every piece and evaluated by three music experts.

6.1.1. Feature Parsing Effectiveness

In terms of effectiveness, results show that five of *FP*'s seven features were parsed correctly all the time for all test pieces, highlighting the deterministic nature of their extraction processes (cf. Section 4.1.1), whereas *dominant key* and *chord progression* feature parsing did not guarantee 100% correctness. Test results showed that *dominant keys* were identified with 93% accuracy (i.e., they were correctly identified in 93 of the 100 of the test pieces), particularly when considering non-modulating music, while *chord progressions* were correctly parsed with 85% accuracy, particularly when considering well-structured music.

By taking a closer look at the pieces where the latter two features were not parsed correctly, we realized that their parsing heuristics did not perform well for atonal music, or for particularly unstructured or non-rhythmic pieces. Yet knowing that *dominant key* and *chord progression* parsing remain open problems in the literature (Demopoulos R.J. et

---

[1] We adopted a ratio $R = 0.7$ in our current study, so that 70% of the offspring would be subject to *fitness trimming*, whereas only 30% would undergo variability trimming.

[2] Available online at: http://sigappfr.acm.org/Projects/MUSEC

al. 2007; Kyogu L. 2008; Zenz V. 2007) (cf. Section 4.1) we consider that our present heuristics perform well enough even in these situations (since providing improved feature parsing techniques is outside of the scope of this study).

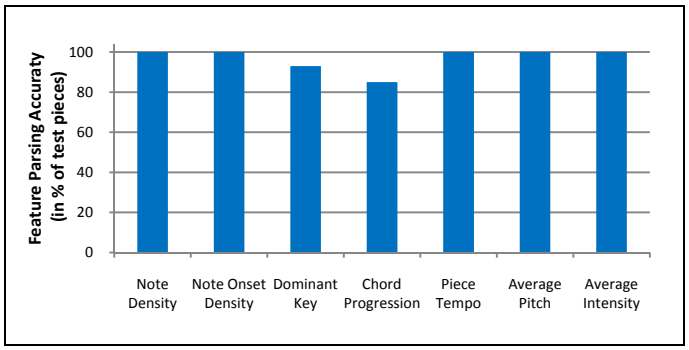
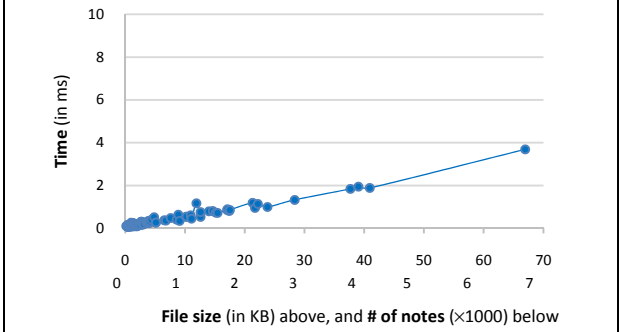


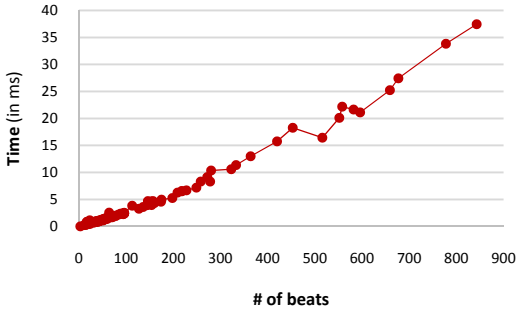
Fig. 11. Feature parsing accuracy, considering *FP*'s seven (symbolic and frequency-based) features

6.1.2. Feature Parsing Efficiency

Results in Fig. 12 highlight: i) aggregated feature parsing time for all MUSEC features (excluding *chord progression*) which varies with the MIDI piece size (Fig. 12.a), while ii) *chord progression* parsing occurs after all the other features have been extracted, and its time varies with the number of beats in a piece (plotted in Fig. 12.b).



a. Aggregate feature parsing time (excluding *chord progression*) w.r.t. MIDI file size and the number of notes in a piece[1]



c. *Chord progression* parsing time w.r.t. the number of beats in a MIDI piece

Fig. 12. Feature extraction time

Results highlight *FP*'s linear complexity in terms of music piece size $|p|$ (which is linear w.r.t. the number of notes n) and the number of beats b in a piece, i.e., $O(|p| + b)$, simplifying to $O(n + b)$. The *chord progression* parsing process is clearly more expensive than its counterparts, due to the inherent (music-theoretic) complexity of detecting and identifying chord progressions (cf. Section 4.1.2), which time is linearly dependent on the number of beats in the music piece[2] (cf. Fig. 12.b).

6.2. Similarity Computation Evaluation

Assessing the quality of similarity computation in comparing two musical pieces is not a straightforward task. Oftentimes, music comparison has no unique correct answer, and usually depends on the listener's judgment. To handle the "subjectivity" of assessing music similarity, we confine our comparison approach to the seven music features considered in MUSEC, and therefore restrict the scope of our tests to the direct application of our aggregate similarity evaluation function (cf. Formula 1). In this experiment, we considered 30 pairs of musical pieces, randomly sampled from the subset of 50 real pieces considered for feature parsing evaluation. Every pair was processed for similarity evaluation using several (weighted) combinations of our individual feature-level similarity functions, in order to find the most suitable and best performing feature similarity combination. The 30 pairs were rated by three music experts, producing an integer value between 0 (for minimum similarity) and 10 (for maximum similarity). Average expert ratings were utilized as a benchmark for this experiment.

---

[1] The number of notes in a MIDI piece is roughly $1/10^{th}$ the size of the corresponding file.

[2] Note that the number of beats in a piece is naturally less than the number of notes. While there is no straightforward relationship between the two, they can be paralleled to *sentences* and *words* in flat text: where beats represent music sentences, and notes represent the sentences' words. In our sample test dataset of 100 pieces, the number of beats was on average 4-to-8 times less than the number of notes.

6.2.1. Similarity Computation Effectiveness

To select a good weighted combination of feature-level similarity functions, and given the infinite number of possible combinations, we limited our testing to the following seven functions which we deemed most interesting to test. Given two music pieces p_1 and p_2, where FV_1 and FV_2 represent their feature vectors:

- ***Tempo_Only*** **Similarity**: Evaluated solely based on the two pieces' *tempo* similarity, where the weight of the *tempo* feature is assigned maximum value: w_{Tempo}=1, and all other features are assigned zero weights, i.e., $Sim(p_1, p_2) = Sim_{Tempo}(FV_1, FV_2) \in [0, 1]$
- ***Chord_Only*** **Similarity:** Evaluated solely based on the two pieces' *chord progression* similarity, where the weight of the *chord progression* feature is assigned maximum value: $w_{ChordProg}$=1, and all other features are assigned zero weights i.e., $Sim(p_1, p_2) = Sim_{ChordProg}(FV_1, FV_2) \in [0, 1]$
- ***Key_Tempo*** **Similarity:** Considering the average of *dominant key* and *tempo* similarities, where the latter features' weights are assigned equal values: $w_{DomKey} = w_{ChordProg} = 0.5$, and all other features are assigned zero weights, i.e., $Sim(p_1, p_2) = 0.5 \times Sim_{DomKey}(FV_1, FV_2) + 0.5 \times Sim_{Tempo}(FV_1, FV_2) \in [0, 1]$
- ***Key_Chord*** **Similarity :** Considering the average of *dominant key* and *chord progression* similarities, and all other features are assigned zero weights, i.e., $Sim(p_1, p_2) = 0.5 \times Sim_{DomKey}(FV_1, FV_2) + 0.5 \times Sim_{ChordProg}(FV_1, FV_2) \in [0, 1]$
- ***High_Level*** **Similarity**: Assigning higher weights to high-level (symbolic) music features, namely *dominant key* and *chord progression*: each assigned a normalized weight of $w_{DomKey} = w_{ChordProg} = 2/7$, and *tempo* assigned $w_{Tempo} = 1/7$, while all other features are assigned equal weights of 1/14 each. Weight values were identified empirically after multiple experimental runs, i.e., $Sim(p_1, p_2) = 2/7 \times Sim_{DomKey}(FV_1, FV_2) + 2/7 \times Sim_{ChordProg}(FV_1, FV_2) + 1/7 \times Sim_{Tempo}(FV_1, FV_2) + \sum_{f \in remaining} 1/14 \times Sim_f(FV_1, FV_2) \in [0,1]$
- **All_But_Chord Similarity:** Considering the average of all feature similarities excluding *chord progression*, i.e., $\sum_{f \neq ChordProg} 1/6 \times Sim_f(FV_1, FV_2) \in [0,1]$
- **Uniform_Average Similarity:** Considering the average of all feature similarities, i.e., $\sum_f 1/7 \times Sim_f(FV_1, FV_2) \in [0,1]$

Similarity scores for all seven test functions applied on all 30 pairs of pieces were computed and compared with the experts' average ratings, using *Pearson Correlation Coefficient* (*PCC*) and *Mean Square Error* (*MSE*). *PCC* is a correlation measure and evaluates the dependence between vector shapes ($\in$[-1, 1], i.e., 1 for maximum correlation, 0 for no correlation, and -1 for negative correlation[1]), whereas *MSE* is a distance measure evaluating the separation between vectors (as their average Euclidian distance $\in [0, \infty[$). A high quality similarity evaluation function would naturally produce: i) high *PCC* scores: which means that the system generated similarity values are closely correlated with the user (expert) ratings; ii) and low *MSE* scores: meaning that the system generated similarity values are not distant from the expert ratings. Results in Table 7 show that *tempo* seems like a relevant feature for computing similarity between two musical pieces, producing high correlation (*PCC*=0.5604) w.r.t. human similarity ratings, compared with other feature combinations. This falls in line with humans' subconscious way of comparing pieces, where fast versus slow pieces are often viewed as (more or less) dissimilar. However, *tempo* used alone produced a high error distance score (*MSE*=0.1126), which means that pairs considered individually (one by one) were evaluated differently w.r.t. human ratings[2]. Surprisingly, *chord progression*, used on its own, produced relatively low correlation (*PCC*=0.2254) and high error distance (*MSE*=0.1021) w.r.t. human ratings, and thus does not seem like a descriptive feature in evaluating music similarity (despite its sophistication and the complexity of its parsing). The highest correlation results obtained with relatively low error distances were produced by combining more than one feature, namely using the *Uniform_Average* aggregate test function (*PCC*=0.6677), followed by *All_But_Chord* (*PCC*=0.6447). These results highlight the importance of aggregating multiple features in evaluating music similarity.

As a result, we adopt the *Uniform_Average* test function as the default aggregate similarity function in the remainder of our experimental evaluation.

---

[1] $PCC = \delta XY/(\delta X \times \delta Y)$ where: x and y designate user and system generated similarity values respectively, δX and δY denote the standard deviations of x and y respectively, and δXY denotes the covariance between the x and y variables. The values of $PCC \in$ [-1, 1] such that: -1 designates that one of the variables is a decreasing function of the other variable (i.e., music pieces deemed similar by human testers are deemed dissimilar by the system, and vice versa), 1 designates that one of the variables is an increasing function of the other variable (i.e., pieces are deemed similar/dissimilar by human testers and the system alike), and 0 means that the variables are not correlated.

[2] *MSE*, computed as an average Euclidian distance measure, is a good indication of how close similarity scores are to human ratings: one by one (for every pair of pieces), whereas *PCC* compares the behavior of the vector of similarity ratings (for all pairs or pieces) as a whole.

Table 7. Average *PCC* and *MSE* scores for all similarity functions under test

Similarity / Metrics	Tempo_Only	Chord_Only	Key_Tempo	Key_Chord	High_Level	All_But_Chord	Uniform_Average
Avg. ***PCC***	0.5604	0.2254	0.4330	0.2784	0.4973	0.6447	0.6677
Avg. ***MSE***	0.1126	0.1021	0.0616	0.0619	0.0714	0.0887	0.0894

6.2.2. Similarity Computation Efficiency

We also conducted efficiency tests to assess the running time of our aggregate similarity computation function. Time results in Fig. 13 show that aggregate similarity computation time comes down to *chord progression* feature similarity time, where *chord progression* similarity is evaluated using the TPSD algorithm (cf. Section 4.3.2). TPSD runs in average linear time w.r.t. the smallest length between the two *chord progressions* (of the two music pieces) being compared, i.e., $O(|cp|)$, whereas all other features require constant (near-zero) computation times, i.e., $O(1)$, since they consist in comparing scalar values (e.g., comparing two *tempos*, or two *note densities*) or values of constant sizes (i.e., comparing two keys within the constant size *circle of fifths* graph).

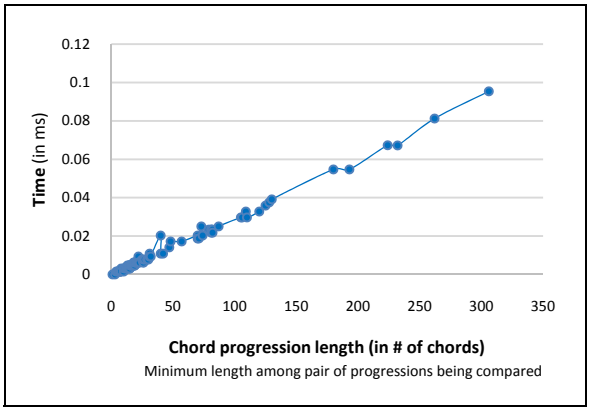


Fig. 13. TPSD running time w.r.t. minimum input chord progression length

6.3. Sentiment Learning Evaluation

To test the performance of our music sentiment learner (*SL*) module, we built a training set of 120 labeled musical pieces using online surveys and human tester training. This set was constructed following multiple experiments and modifications, aiming to produce high-quality training data which evenly represents all sentiment categories. To our knowledge, this is the first significant set of sentiment-labeled MIDI pieces, which is available online[1] as a benchmark to promote future research in this area. In the following, we first describe the training set construction process in Section 6.3.1, and then present and discuss *SL*'s effectiveness and efficiency test results in Sections 6.3.2-to-6.3.4.

6.3.1. Training set Construction

Initially, we started with a small dataset of 24 pieces, ranging from classical to contemporary, each 30 seconds to 2 minutes long, which we assembled into a survey[2], where respondents were asked to rate each piece in terms of the six basic sentiments considered in MUSEC (i.e., anger, fear, joy, love, sadness, surprise), on an integer scale from 0 (sentiment is not expressed) to 10 (sentiment is fully expressed). Testers were explicitly informed that they could rate any of the 24 pieces, and abstain from rating others if they felt they could not infer the sentiments confidently[3]. Every tester could stop the survey and submit the results at any stage of the process, to ensure consistency and high confidence in the results. Pieces were sequentially shuffled and sent to testers in a round-robin fashion to ensure an even distribution of responses. We also dealt with inconsistencies in piece ratings by eliminating scores with negative inter-tester correlation scores. In total, 359 responses were retained, with every piece receiving over 30 responses, the average of which was used to train the system. At that stage, the learning component scores produced an average *PCC*=0.53 using 3-fold cross validation (with 20 training pieces, and 10 testing pieces). Considering that the result was unsatisfactory, we proceeded to increase the size of the training set to 100 pieces by producing 76 "synthetic" pieces using MUSEC's sentiment-based music composition (*MC*) module. These pieces were added to the system's training set using the lifelong learning feature. Using 10-fold cross validation (with 90 training pieces, and 10 testing pieces), we obtained an average *PCC*=0.67, which was a major improvement over the 0.53 figure obtained previously.

---

[1] Available online at: http://sigappfr.acm.org/Projects/MUSEC

[2] http://sigappfr.acm.org/Projects/MUSEC, *SL* survey form #1 (first part, 24-pieces), #2 (second part, 8-pieces), and #3 (third part, 8-pieces).

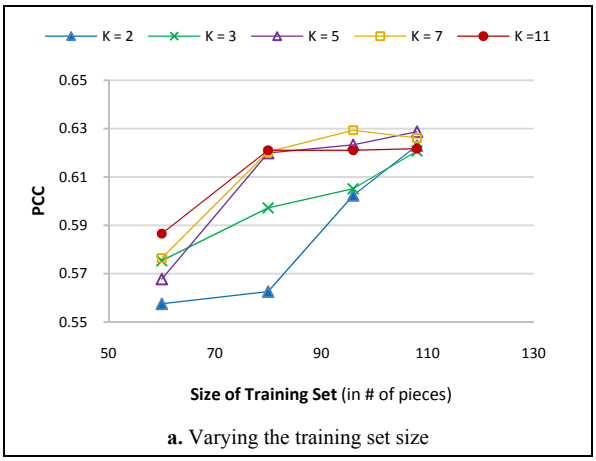
[3] While we could have asked the testers to provide a confidence score associated with every sentiment score, yet, we felt this would complicate things for non-expert testers, especially that our objective was to capture their inherent feelings when listening to the music pieces, rather than have them "rationalize" their ratings by adding confidence scores. Nonetheless, considering tester rating confidence is an interesting factor that we plan to evaluate in a future study.

After closely analyzing the data and the results, we identified another issue with our training set: it seemed *biased* toward certain emotions. Indeed, our dataset was mostly made of joyful and sad pieces, while angry, fearful, and surprising pieces were almost nonexistent. To remedy this situation, we added an additional 16 real pieces to the training set, expressing mostly anger and fear. These pieces were selected based on human sentiment scores obtained by averaging the results of two other online surveys[2] designed in a similar format to the first survey (which produced the first 24 pieces). A total of 240 responses were retained, with every piece receiving over 30 responses. Finally, we tackled the bias issue further by removing 16 sad and joyful pieces from the 76-piece synthetic set, whilst replacing them with 20 more evenly distributed synthetic pieces. The resulting dataset, when looked at in a crisp manner (i.e., assigning a piece to the crisp sentiment category corresponding to the maximum sentiment score), consists of the following distribution (in number of pieces): anger (17), fear (17), joy (26), love (18), sadness (25), and surprise (17).

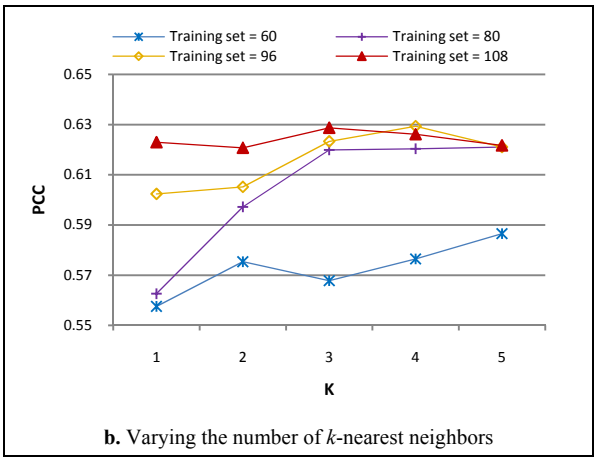
For this final training set, we obtained an average *PCC*=0.63 using 10-fold cross validation (with 108 training pieces, and 12 testing pieces), which proved to perform better on a wider range of musical pieces. Though scoring lower than the 0.67 correlation score obtained using the initial 100-piece training set, results showed that the system was in fact doing a better overall job and performing better in terms of detecting anger and fear, whereas the previous dataset's performance was due to over-fitting[1]. Therefore, we adopted the 120-piece dataset described above as our training set, and proceeded to test our system using this dataset in terms of both effectiveness and efficiency.

6.3.2. Sentiment Learner Fuzzy Scoring Effectiveness

To assess the quality of our system's sentiment learning effectiveness, we evaluated the sentiment learner (*SL*) module's fuzzy scoring ability in correctly inferring sentiments from music pieces, using *PCC* and *MSE* w.r.t. user provided sentiment scores. The experimental evaluation was conducted while varying: i) training set size from 60, 80, 96, to 108 pieces (using 2, 3, 5, and 10-fold cross validation) and ii) *k*-nearest neighbor (*k*) parameter score from 2, 3, 5, 7, to 11. Results are shown in Fig. 14 and Fig. 15, and are averaged in Table 8.

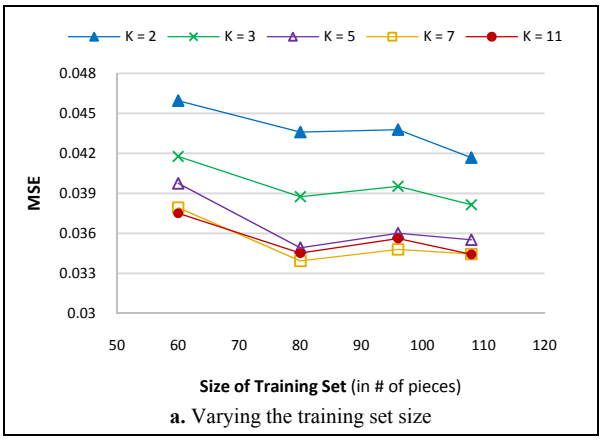


a. Varying the training set size

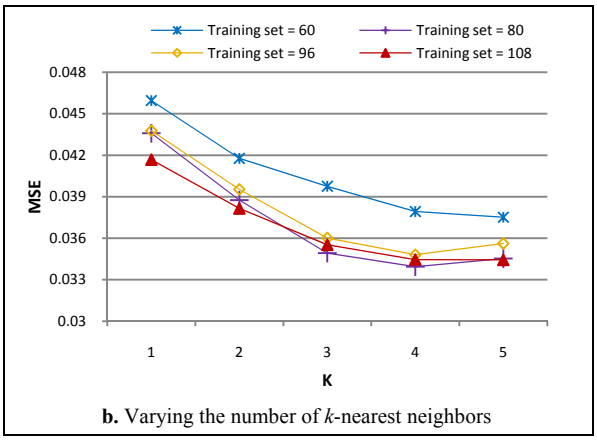


b. Varying the number of *k*-nearest neighbors

Fig. 14. PCC results obtained while varying the training set size (a), and the number of *k* nearest neighbors (b)



a. Varying the training set size



b. Varying the number of *k*-nearest neighbors

Fig. 15. MSE results obtained while varying the training set size (a), and the number of *k* nearest neighbors (b)

[1] With the 100-piece training set, the system had "less" to learn since it was training on a more or less homogeneous training set, and thus over-fitted fitted w.r.t. the well represented sentiments, namely joy and sadness, but was less successful in inferring less represented sentiments like anger and fear.

Table 8. Average *PCC* and *MSE* wr.t. training set size and *k*-neireast neighbors parameters

	Training set size[1]				***k* parameter**[2]				
	60	80	96	108	2	3	5	7	11
Avg. ***PCC***	0.5728	0.6042	0.6163	0.6249	0.5864	0.5996	0.6099	0.6131	0.6126
Avg. ***MSE***	0.0406	0.0371	0.0380	0.0368	0.0438	0.0396	0.0366	0.0353	0.0355

From these results, we can make three observations. First, we clearly see that system performance improves as the size of the training set increases, considering both *PCC* (steadily increasing) and *MSE* (steadily decreasing). This shows that *SL*'s ability to extract sentiments from musical pieces improves as it is exposed to and as it learns more and more pieces. Second, we also notice that *PCC* tends to increase as the number of nearest neighbors *k* increases (except with training set size =108 where it tends to stabilize), while *MSE* steadily drops with the increase of *k*. Following our analysis of fuzzy *k*-NN, as we increase the number of neighbors, the training vectors used for score computation tend to become more diverse and less similar to the target piece's vector (increasing the learner's training set variety, and thus increasing its resistance to noise when performing classification). Hence, training vectors become more normally distributed, which in turn reduces and normalizes the system generated sentiment vectors. Third, we can also notice that while *PCC* values tend to increase with the increase of *k* (despite slightly decreasing with $k > 5$), however, *MSE* consistently drops as *k* increases. Here, we make a distinction between the two measures. *PCC* is a correlation measure which compares the behaviors of the vectors, whereas *MSE* is a distance measure that measures their average Euclidian distance. Both are good measures for computing similarity. *PCC* reflects the *overall* similarity of a predicted sentiment vector w.r.t. the target vector, whereas *MSE* is only a good indication of how close scores are to target sentiments one by one[3]. As we increase the value of *k*, the training vectors used for score computation become more diverse and less similar to the target piece (and can be considered as "noise" to the learning algorithm). They become more normally distributed, which in turn reduces and normalizes the predicted sentiment characteristics. In other words, with higher *k*, sentiment scores lose their shape towards a more even sentiment distribution in which relative differences among sentiments drop. This change is detected through *PCC*, which drops due to the change in the overall vector shape. However, this "normalization" in scores tends to draw them closer to a mean sentiment score, which is reflected in a lower *MSE* distance error.

6.3.3. Sentiment Learner Crisp Classification Effectiveness

In addition to evaluating *SL*'s fuzzy sentiment scores, we also evaluated its crisp scores, or so-called crisp classification quality. To do so, we used well-established *precision*, *recall*, and *f-value* measures from the literature (Baeza-Yates R. et al. 2011; Ghosh A. et al. 2003). High *precision* denotes that the classification task achieved high accuracy, grouping together music pieces that actually correspond to the same sentiment (class). High *recall* means that very few pieces are not associated to the appropriate sentiment (class) where they should have been. Hence, high *precision* and *recall*, and thus high *f-value* (indicating in our case excellent classification quality) characterize a good crisp sentiment extraction method.

To perform crisp testing, we first had to convert *SL*'s fuzzy scores into crisp ones. This was done by considering the sentiment with the highest score as the representative sentiment for the piece. Human tester scores were also made crisp in this manner. In other words, while training and testing were performed based on the initial fuzzy training and testing scores, it was only at the final evaluation phase that the fuzzy-to-crisp conversion was made. The testing protocol used for crisp evaluation is described as follows:

1) Compute a fuzzy sentiment score for every piece in the training set using cross-validation,
2) Compute the system generated crisp sentiment (system generated class), and match it with the human tester crisp sentiment (human expected class),
3) For each of the 6 sentiments, compute the number of true positives and true negatives, false positive and false negatives, and use the latter to compute sentiment-level *precision*, *recall* and *f-value* scores.

Results in Fig. 16 and Fig. 17 show *SL*'s crisp classification results, when varying i) training set size from 60, 80, 96, to 108 pieces (using 2, 3, 5, and 10-fold cross validation, cf. Fig. 16) and ii) *k*-nearest neighbor (*k*) parameter value from 2, 3, 5, 7, to 11 (cf. Fig. 17). Results are summarized in Table 9.

---

[1] *PCC* and *MSE* are averaged over all *k*-nearest neighbor variations, for every training set size.

[2] *PCC* and *MSE* are averaged over all training set size variations, for every *k* nearest neighbor value.

[3] To help illustrate this concept, let's consider the following example, consisting of three vectors: $V_1 = (0.8, 0.6)$, $V_2 = (0.95, 0.45)$, and $V_3 = (0.65, 0.75)$. Let V_1 be our target vector and let V_2 and V_3 be our system estimate vectors. Upon first inspection, it is obvious that V_2 is a better representative of V_1 than V_3, since it more or less exhibits the same behavior as V_1 (higher first term). This similarity in behavior is visible through *PCC*, where $PCC(V_1,V_2)=1$ and $PCC(V_1,V_3)=-1$. However with *MSE*, we obtain $MSE(V_1,V_2) = MSE(V_1,V_3) = 0.0225$. This shows that *MSE* is only a good indication of how close scores are to target sentiments one by one, while *PCC* reflects the *overall* similarity of a predicted sentiment vector to the target vector as a whole.

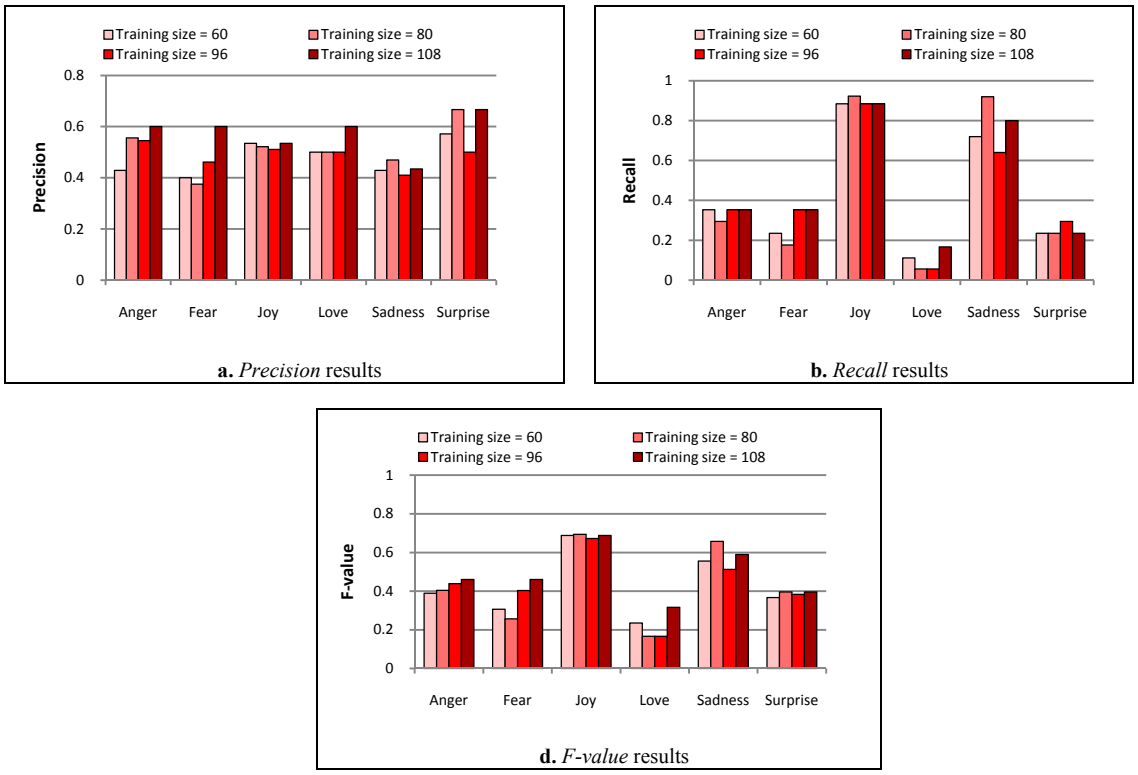


Fig. 16. *Precision*, *recall*, and *f-value* scores for all 6 sentiments, when varying the training set size (number of folds), while fixing the number of *k* nearest neighbors (i.e., *k*=5)

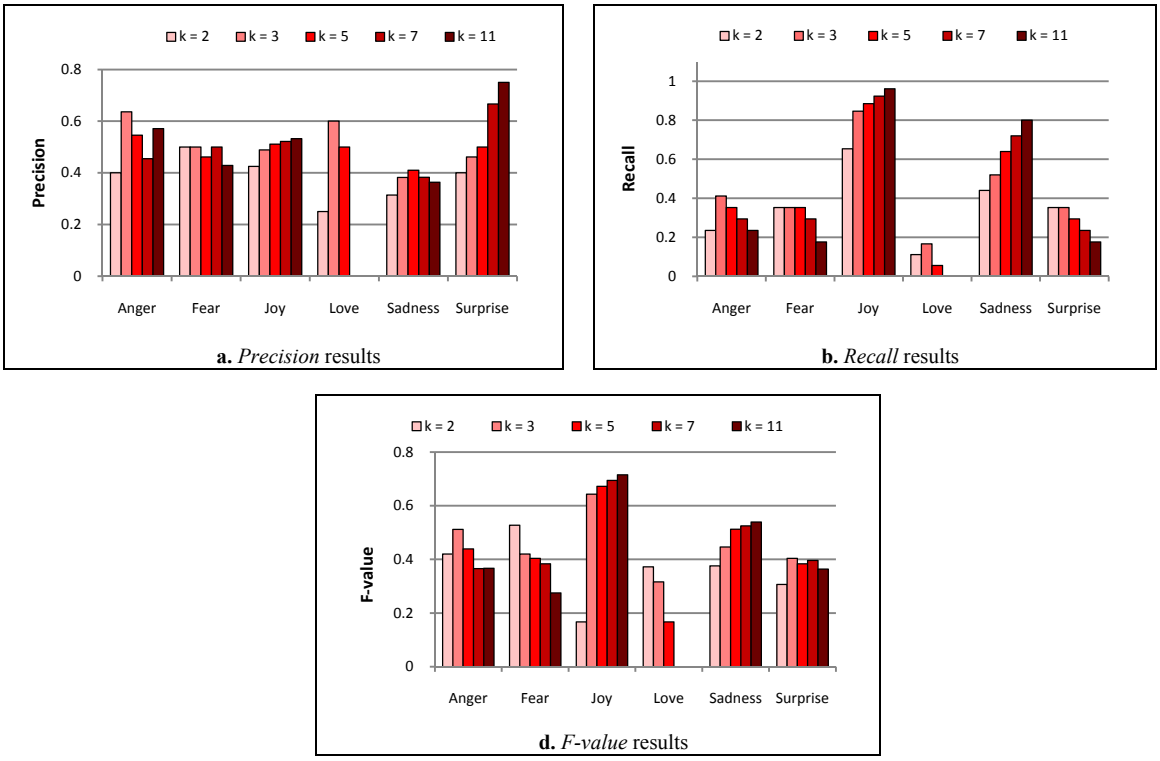


Fig. 17. *Precision*, *recall* and *f-value* scores for all 6 sentiments, when varying the number of *k* nearest neighbors, while fixing the training set size (= 96, i.e., number of folds = 5)

Results confirm our intuition concerning training set bias. On one hand, joy and sadness, which are represented with more pieces in the training set (i.e., 26 and 25 pieces respectively) compared with other sentiments (represented by 17 or 18 pieces each), benefitted little from the increased size of the training set (cf. Fig. 16) since they already had

sufficient training sample representations even with small training sets. However, they clearly benefitted more from the increased number of nearest neighbors *k* (including more relevant samples in their decision neighborhoods increases their *precision* and *recall* levels, cf. Fig. 17). On the other hand, less represented sentiments, like anger and fear, benefitted more from the increased training set size (cf. Fig. 16), allowing the learning algorithm to acquire enough "experience" to better learn these sentiments. Nonetheless, they benefitted less from the increase of *k*, which rather had a negative effect on their classification quality (increasing *k* without having enough training samples would include more noise in the decision neighborhoods, thus reducing classification *precision* and *recall* accordingly, as shown in Fig. 17 for the anger and fear sentiments).

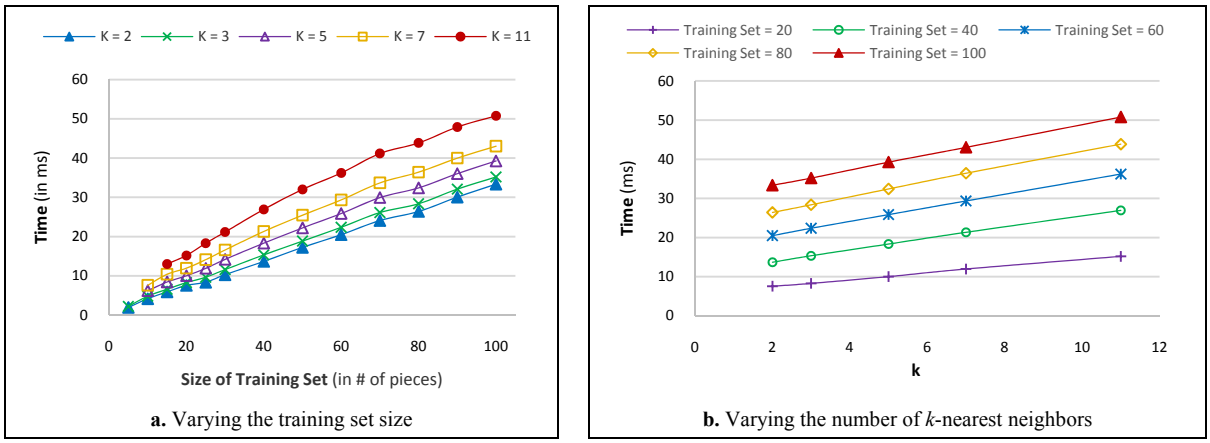
Table 9. Average *f-value* scores wr.t. training set size (number of folds), and *k*-nearest neighbor parameters

	Training set size[1]				***k* nearest neighbors**[2]				
	60	**80**	**96**	**108**	**2**	**3**	**5**	**7**	**11**
Anger	0.3958	0.3579	0.3980	0.4373	0.3536	0.4512	0.4230	0.3648	0.3936
Fear	0.3176	0.2538	0.3805	0.4717	0.3997	0.3917	0.3570	0.3645	0.2666
Joy	0.6547	0.6880	0.6504	0.6597	0.5887	0.6632	0.6855	0.6814	0.6972
Love	0.1626	0.1368	0.1299	0.2254	0.2256	0.2739	0.2213	0.0975	0.0000
Sadness	0.5422	0.5522	0.4789	0.5428	0.4265	0.4833	0.5787	0.5867	0.5699
Surprise	0.3907	0.3970	0.3845	0.4147	0.4176	0.4489	0.3856	0.3894	0.3424
Avg.	**0.4106**	**0.3976**	**0.4037**	**0.4586**	**0.4019**	**0.4520**	**0.4418**	**0.4140**	**0.3783**

All in all, considering all crisp sentiments put together, results in Table 9 show that average *f-value* scores tend to increase with the increase of the size of the training set, highlighting overall improved system performance, while stabilizing and even slightly decreasing with the increased number of nearest neighbors *k* (especially with less represented sentiments). We expect learner performance to improve even further as the training set size further increases (beyond our 120 piece dataset, allowing to better represent all sentiments).

6.3.4. Sentiment Learner Efficiency

Following Section 5, our *SL* component's time complexity comes down to $O(|cp|\times T\times k)$ where $|cp|$ designates the smallest length chord progression between two pieces being compared, T the number of pieces in the training set, and k the number of nearest neighbors considered when producing the fuzzy sentiment scores. Linear dependency w.r.t. $|cp|$ comes down to the complexity of our similarity computation function, i.e., $O(|cp|)$, which was empirically evaluated in Section 6.2.2 (cf. Fig. 13). Results in Fig. 18 confirm the linear relationship between *SL*'s execution time and the training set size (Fig. 18.a), as well as the number of *k* nearest neighbors (Fig. 18.b).



a. Varying the training set size

b. Varying the number of *k*-nearest neighbors

Fig. 18. *SL* running time, when varying the training set size (T), and the number of nearest neighbors (k)

6.4. Music Composer Evaluation

Assessing the quality of the music composer (*MC*) module can be done in multiple ways. It can be done in terms of music theory, where the composed pieces are checked to confirm whether they meet music-theoretical criteria. Yet, given the nature of our composer and its integral music-theoretic validation procedures (maintained using our music-theoretic knowledge base - *KB*, cf. Section 4.2), we found such testing to be of secondary importance, since all synthetic pieces inherently comply with all music theoretic rules considered in the *KB* module. Instead, we opted to test the effectiveness of our composer in terms of whether it can produce human-like, genuinely interesting, and sentiment expressive music. Note that not all theoretically-correct music is appealing, is deemed beautiful by humans, or expresses their desired feelings. Most crucially, we evaluated whether our composer truly hits the target sentiments

[1] *F-value* scores are averaged over all *k*-nearest neighbor variations, for every training set size.

[2] *F-value* scores are averaged over all training set size variations, for every *k* nearest neighbor value.

it is given as input, when producing compositions as output. To that purpose, we used a smaller set of 15 real pieces, and compared them with 15 synthetic MUSEC compositions, considering both non-expert surveys and expert user feedback. In the following, we first describe the composer evaluation dataset construction process in Section 6.4.1, and then present and discuss *MC*'s effectiveness and efficiency test results in Sections 6.4.2-to-6.4.5.

6.4.1. Evaluation Criteria and Dataset Construction

We designed our experiments to test three criteria:

i. *Composer's Nature*: it consists of an integer score from 0 to 10, where 0 indicates absolute certainty that the piece is composed by a computer, and 10 indicates absolute certainty that the piece is composed by a human. This comes down to a form of Turing test[1] , where an overall score ≈ 5 indicates uncertainty about the nature of the composer, i.e., the human listener cannot tell if the piece comes from a human composer or a synthetic composer. MUSEC passes the Turing test if its pieces produce an average score ≥ 5, either confusing the listener as to its pieces' nature, or tricking the listener into believing its pieces were composed by a human.

ii. *Appeal-Enjoyment*: it consists of an integer score from 0 to 10, where 0 indicates a total lack of appeal or enjoyment by the listener, and 10 denotes extreme appeal or enjoyment,

iii. *Sentiment expressiveness*: it consists of a vector of six integer values, from 0 to 10, representing the piece's expressiveness of the six sentiments considered in MUSEC (anger, fear, joy, love, sadness, surprise), where for every sentiment, 0 indicates that the sentiment is not expressed, and 10 indicates that the sentiment is fully expressed (similarly to the user sentiment vectors produced in the *SL* evaluation process, cf. Section 6.3).

Table 10. Set of 15 synthetic MUSEC compositions utilized in *MC*'s experimental evaluation

Piece #	Target Sentiment Scores						Mutation Distribution/ Configuration
	Anger	*Fear*	*Joy*	*Love*	*Sadness*	*Surprise*	
S1	0.5	0.2	0	0	0.2	0	Standard (all mutation probabilities=0.1)
S2	0.3	0.5	0.2	0.2	0.4	0.4	Standard
S3	0	0	0.8	0.6	0	0.2	Standard
S4	0.4	0.5	0	0	0.3	0.1	Standard
S5	0.3	0.3	0	0	0.7	0	Standard
S6	0	0	0.7	0.3	0	0.3	Standard
S7	0.4	0.2	0	0	0.4	0	Standard
S8	0.7	0.3	0	0	0.7	0	Standard
S9	0.3	0.7	0	0	0.7	0	Standard
S10	0	0	0.4	0.8	0	0	Standard
S11	0	0	0.6	0.6	0	0	*Silence* mutation probability = 0.05, restricted divisions and repetitions to even split
S12	0	0.3	0.3	0.3	0.3	0.6	Standard
S13	0	0	1	0.5	0	0	*Silence* probability = 0.05, restricted divisions and repetitions to even split
S14	1	0	0	0	0	1	*Silence* probability = 0, restricted divisions and repetitions to even split
S15	1	0	0	0	0	0	*Trille*, *Appoggiatura* and *Double Appoggiatura* probabilities are set to 0.2

To evaluate the above criteria while ensuring unbiased results, we built our experimental dataset with the assistance of an expert: professional composer and music instructor Anthony Bou Fayad[2]. Together, we produced a dataset consisting of 30 pieces: 15 real and 15 synthetic (produced by MUSEC's *MC*), each up 2 minutes long. To our knowledge, and following existing literature on algorithmic music composition evaluation (cf. Section 3.2), the latter would be the first dataset of MIDI pieces used for sentiment-based algorithm music composition evaluation[3]. Initially, we used MUSEC*'s MC* module to generate 15 compositions[4], each between 30 and 90 seconds in length, such that the overall composition set evenly portrays MUSEC's six primary sentiments. The pieces were presented to the expert for feedback. While he found several compositions to be "interesting" and "new", he highlighted a lack of rhythmic "logic" in some of the compositions, largely due to unusual divisions (3 to 1, rather than an even split), and repetitions produced by certain mutation operators such as *Repeat* and *Compress* (cf. Section 4.4.4). He also suggested that mutation probabilities be tweaked between pieces so that the system can compose pieces reflecting similar sentiments *in a different style*. As a result, 4 of the initial 15 MUSEC compositions were replaced with new compositions arising from different mutation probabilities and configurations. Table 10 lists the 15 final MUSEC compositions, their target sentiment scores, as well as the mutation configurations used during their creation.

---

[1] The Turing test was proposed by Alan Turing in 1950, designed to test the ability of a machine to exhibit intelligent behavior that is equivalent to or indistinguishable from that of a human. It was originally used to evaluate machines mimicking human conversation (originally referred to as the "imitation game"). A machine passes the Turing test if, after a number of questions, the human tester (asking questions) cannot know if the answers come from a human or a machine (Epstein R. *et al.* 2009).

[2] Anthony Bou Fayad is a processional composer, pianist, and music instructor in the Antonine University's School of Music, located in Baabda, Mont Lebanon. He also holds a Master's of Computer Engineering, specializing in multimedia data processing, which allowed him to easily understand the context and purpose of our study, helping us set up the experimental process. Mr. Bou Fayad was partly remunerated for his efforts, mainly for playing and digitally recording all pieces, while volunteering his consulting services.

[3] Some music composition systems provide sample pieces online, e.g., (Diaz-Jerez G. 2011; SACEM 2016), yet none of them are sentiment-based.

[4] Using a population size $S = 50$, a generation size N varying between 50 and 80, a branching factor $B = 10$ and a fitness-to-variability ratio $R = 0.7$. All mutation probabilities were set to 0.1

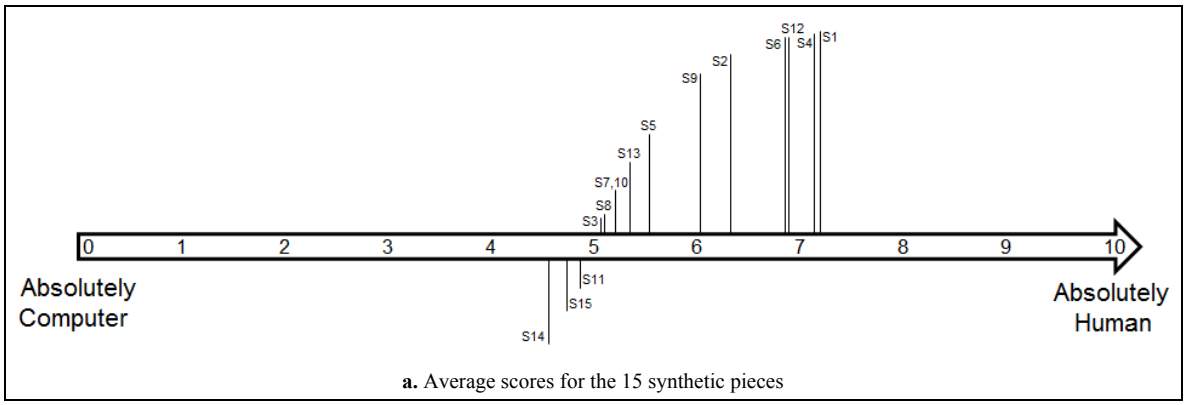
Then, with the help and suggestions of the expert, we selected 15 human compositions of similar sophistication and musical standard to the considered MUSEC compositions, thus producing our experimental 30-piece dataset (cf. Table 11). Anthony Bou Fayad, as a professional pianist, performed all thirty pieces and recorded them using the same digital piano. We also asked that he represents MUSEC composition themes loyally and to reflect their chord progressions accurately. Given that MUSEC only produces static arrangements for its main melodies, all thirty pieces, real and synthetic, were performed with similar arrangement sophistication, to ensure that all pieces would be judged for their melodies. The pieces' music charts, digital recordings, and experimental results are available online.

Table 11. Set of 15 real pieces used in *MC*'s experimental evaluation

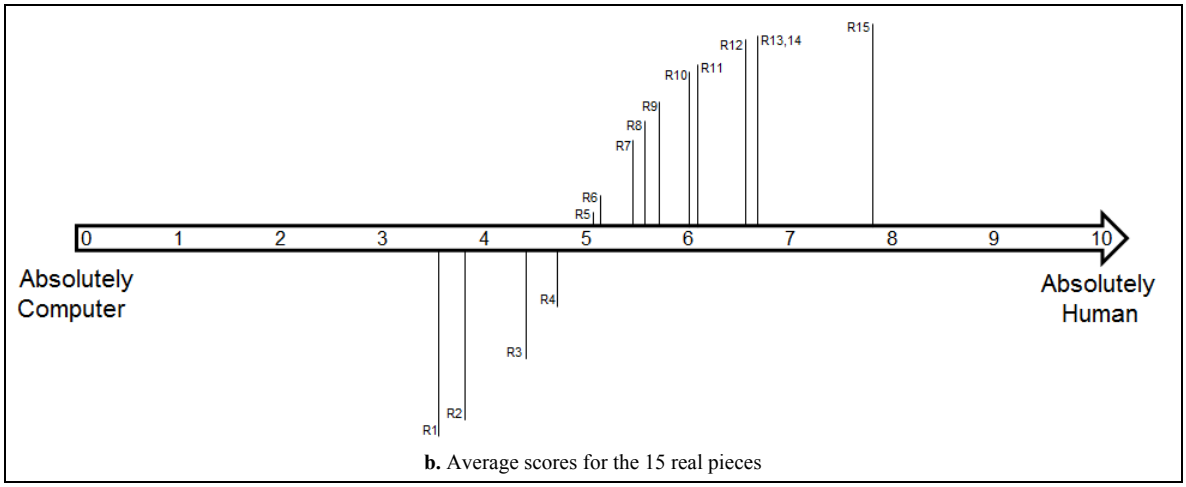
Piece #	Piece Name	Piece Composer
R1	Prélude No 2	Frédéric Chopin
R2	Désir, Op 57 No 1	Aleksandr Scriabin
R3	Le Gibet	Maurice Ravel
R4	5 Piano Pieces	Franz Liszt
R5	Prélude No 10	Sergei Rachmaninoff
R6	Prélude No 4	Aleksandr Scriabin
R7	Suite in G Major	Henry Purcell
R8	Ziemlich langsam, Albumblatter, Op. 99	Robert Schumann
R9	Sicilienne	Gabriel Fauré
R10	Prélude No 8	Johann Sebastian Bach
R11	Gnoissienne No 1	Éric Saté
R12	Impromptus Op 142 No 2 in A flat Major	Franz Schubert
R13	Melody	Arno Babajanian
R14	Andantino	Aram Khachaturian
R15	Lebewohl	Anthony Bou Fayad

6.4.2. Composer Effectiveness

Similarly to MUSEC's *SL* evaluation process, the 30 pieces were randomly assembled into a survey[1], where respondents were asked to rate the pieces in terms of: i) their nature, ii) appeal, and iii) sentiment expressiveness. Pieces were sequentially shuffled and sent to testers in a round-robin fashion to ensure an even distribution of responses. We also dealt with inconsistencies in ratings by eliminating entries with negative inter-tester correlation scores. In total, 348 responses were retained, with every piece receiving at least 30 responses.



a. Average scores for the 15 synthetic pieces



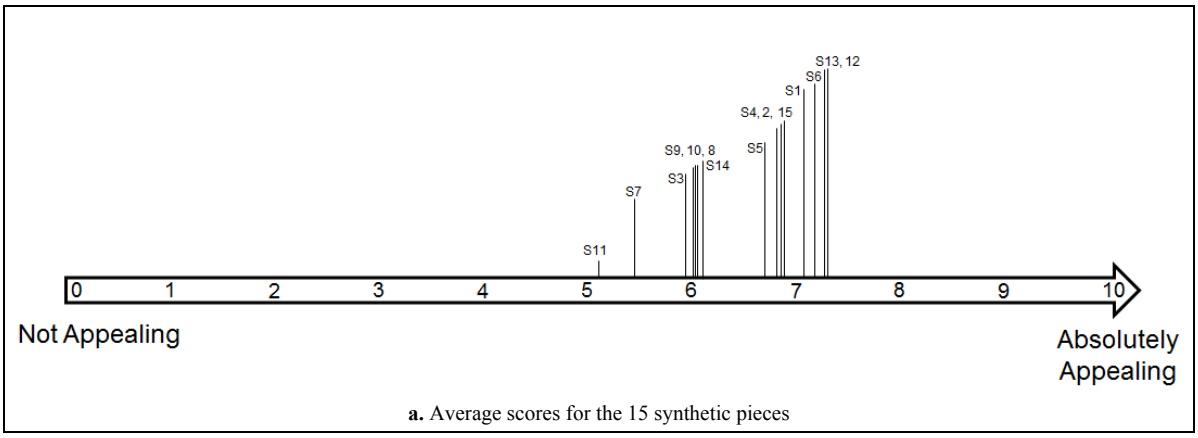
b. Average scores for the 15 real pieces

Fig. 19. *Composer's nature* average scores for both synthetic and real pieces considered in *MC*'s evaluation

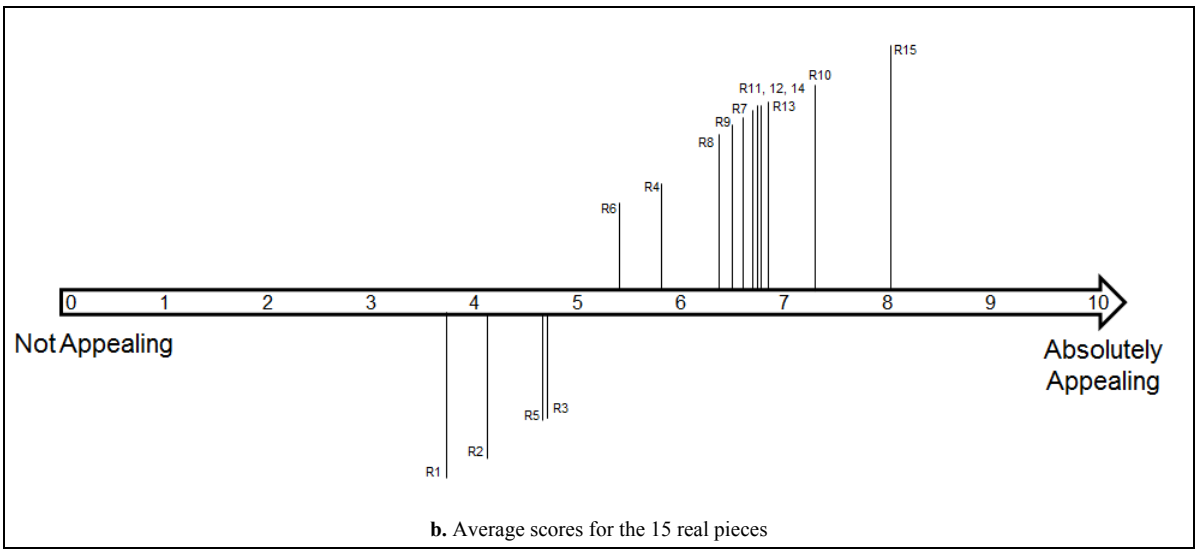
[1] Available online at: http://sigappfr.acm.org/Projects/MUSEC, *MC* survey forms #1-to-#10.

i) Composer's Nature: average composer nature scores for all 30 pieces are shown in Fig. 19 and are provided in Table 12.a. Results were very encouraging, as MUSEC compositions seemed to perform slightly better than their human counterparts, earning an overall average *composer nature* score of **5.736** compared with **5.548** for real pieces. Results show that 13 out of 15 synthetic pieces produced a score ≥ 5, meaning that listeners were either confused about the nature of the pieces (e.g., with scores ≈ 5 such as with pieces S3, S7, S8, and S10) or considered the latter to be composed by humans (compared with 12 out of 15 real pieces having scores ≥ 5). Real piece R15 was deemed "most human" by the listeners (with average score = 7.795), followed by two MUSEC compositions: S1 and S4 (with scores = 7.189 and 7.113 respectively). This shows that listeners had difficulty discerning between the real pieces and MUSEC's synthetics. Considering the above results, we can say that MUSEC passed this Turing test, where most of its synthetic pieces were either confusing to listeners as to their nature, or simply tricked them into believing they were composed by humans.

ii) Appeal-Enjoyment: Results are provided in Fig. 20 and Table 12.b. MUSEC compositions seemed to outperform their human counterparts in terms of appeal and enjoyment by the listeners, producing an average score of **6.498**, compared with **6.035** obtained with real pieces. All synthetic pieces received appeal-enjoyment scores ≥ 5 (i.e., they scored "above average") whereas 4 real pieces (R1-3 and R5) received scores < 5 (i.e., they were considered as rather not appealing-enjoyable by listeners). We also found that *composer nature* and *appeal-enjoyment* scores shared a very strong positive correlation, producing an overall *PCC* score of **0.856** for all 30 pieces. This correlation can be clearly seen in Fig. 21, and demonstrates that listeners might have associated the appeal-enjoyment of a piece with the human nature of its composer, revealing a potential implicit assumption: *human listeners seemed to consider that computer systems cannot produce enjoyable pieces* (or that *enjoyable music can only be created by human composers*). The obtained results however clearly contradict the latter assumption (many "curious" testers who requested to know the true nature of the pieces they enjoyed, after completing the test surveys, were "amazed" to learn that many of those pieces were synthetic).



a. Average scores for the 15 synthetic pieces



b. Average scores for the 15 real pieces

Fig. 20. *Appeal-Enjoyment* average scores for both synthetic and real pieces considered in *MC*'s evaluation

iii) Sentiment Expressiveness: We computed the average sentiment scores for all 15 synthetic pieces and correlated the latter with their initial target sentiment scores provided as input at composition time. Results are provided in Table 13. We realized that MUSEC generally performs well for most of its composition tasks in expressing the intended target sentiments, obtaining low (negative) *PCC* scores for only 3 out the 15 synthetic pieces (i.e., S1, S4, and S5). The latter compositions for which MUSEC did not perform well seem to have: i) high *love* and *joy* scores when not required to, or ii) a high *surprise* score when an *angry* piece is requested from the system. The high *love* and *joy* scores are particularly apparent in pieces S1, S4 and S5, which are written in a minor key (normally used to reflect sadness rather than joy and love), while the high *surprise* score of an *angry* composition is apparent in pieces S15 and S8. All in all, MUSEC was able to quite accurately express the correct sentiments in 12 out of the 15 pieces, particularly those conforming to standard musical logic (major keys generally portray positive emotions, whereas minor keys generally portray negative emotions), producing an overall *PCC*=0.483 (where *PCC*=0.724 when disregarding negative correlation pieces, i.e., S1, S4, and S5). MUSEC however seemed to struggle when users produced "unconventional" sentiment scores, i.e., when they found positive/negative sentiments in otherwise minor/major key pieces respectively (namely in S1, S4, and S5). Here, we also note that the latter pieces (having unconventional user sentiment scores) produced some of the lowest inter-tester correlation scores (0.033 for S5, 0.1509 for S1, and 0.233 for S4), compared with the remaining pieces where inter-tester correlation is generally > 0.3 (cf. Table 13). In other words, even human listeners themselves tend to diverge greatly in rating the sentiments expressed in the latter pieces, which could also explain MUSEC's low correlation scores.

Table 12. Average human ratings evaluating: the *composer's nature* (a) and *appeal-enjoyment* (b) for both synthetic and real pieces considered in *MC*'s evaluation

a. *Composer nature* scores

Synthetic piece #	*Score*	Real piece #	*Score*
S1	7.189	R1	3.548
S2	6.324	R2	3.838
S3	5.067	R3	4.400
S4	7.133	R4	4.706
S5	5.526	R5	5.064
S6	6.853	R6	5.135
S7	5.216	R7	5.448
S8	5.093	R8	5.567
S9	6.029	R9	5.711
S10	5.206	R10	6.000
S11	4.861	R11	6.088
S12	6.886	R12	6.558
S13	5.353	R13	6.674
S14	4.564	R14	6.684
S15	4.743	R15	7.795
Average:	**5.736**	**Average:**	**5.548**

b. *Appeal-Enjoyment* scores

Synthetic piece #	*Score*	Real piece #	*Score*
S1	7.135	R1	3.774
S2	6.892	R2	4.135
S3	5.933	R3	4.629
S4	6.867	R4	5.829
S5	6.789	R5	4.613
S6	7.286	R6	5.432
S7	5.405	R7	6.633
S8	6.093	R8	6.414
S9	6.057	R9	6.533
S10	6.088	R10	7.333
S11	5.108	R11	6.735
S12	7.371	R12	6.774
S13	7.353	R13	6.86
S14	6.179	R14	6.789
S15	6.914	R15	8.051
Average:	**6.498**	**Average:**	**6.035**

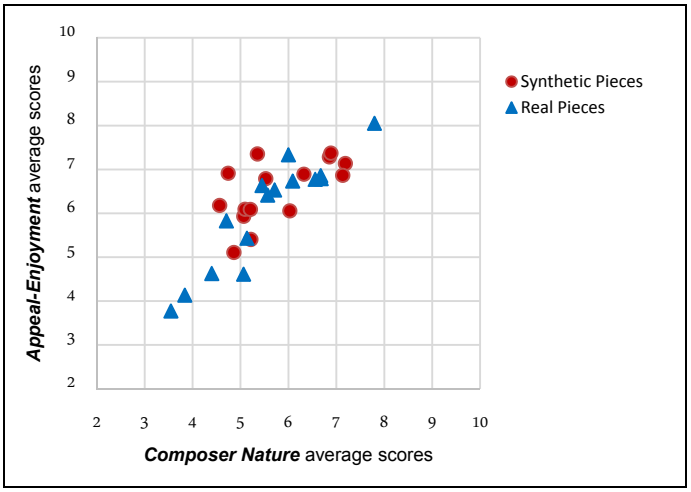


Fig. 21. Scatter plot highlighting correlation between *composer nature* (Turing) and *appeal-enjoyment* scores

Looking forward, we will attempt to further train the system to incorporate the *surprise* side-effect into its own computations when creating *angry* pieces, and to account for potential positive/negative emotions appearing implicitly in otherwise minor/major-oriented compositions.

6.4.3. Expert Feedback

In addition to non-expert surveys, we also acquired the feedback of 3 music experts. First, Anthony Bou Fayad (mentioned previously) helped up set-up, prepare, and record the experimental music dataset, suggesting that mutation probabilities be tweaked between pieces so that the system can compose pieces reflecting similar sentiments *in a*

different style (as discussed previously). The 4 pieces composed with different mutation configurations following the expert's recommendations (i.e. pieces S11, S13, S14 and S15) performed well in terms of sentiment portrayal (average *PCC*=0.781) despite their differing styles, and seemed to be appealing/enjoyable by listeners (with average *appeal-enjoyment* = 6.388). However, their *composer nature* (Turing test) performance was not satisfactory, where pieces S11, S14 and S15 were the only three MUSEC compositions to score a *composer nature* score < 5 (i.e., identified by listeners as non-human compositions), while piece S13 only scored 5.353. In fact, their average *composer nature* was only = 4.880. From this, we can infer that these pieces, composed differently from the other 11 MUSEC pieces, managed to reflect the sentiments fed into the system whilst also being appealing/enjoyable by users, but, due to their different and unfamiliar styles, were perceived to be written by a computer composer (i.e., failing the Turing test). These results confirm the expert's initial feedback: that the system's probabilistic parameters, if properly tweaked, can produce *novel* compositions expressing the same emotions but in *different styles*.

Table 13. Sentiment expressiveness results for the 15 synthetic pieces produced by MUSEC's *MC* module

MUSEC Piece #		*Anger*	*Fear*	*Joy*	*Love*	*Sadness*	*Surprise*	PCC	MSE	Inter-tester correlation
S1	Target	0.5	0.2	0	0	0.2	0	-0.767	0.143	0.151
	Result	0.284	0.281	0.551	0.565	0.368	0.395			
S2	Target	0.3	0.5	0.2	0.2	0.4	0.4	0.469	0.041	0.360
	Result	0.616	0.486	0.230	0.439	0.694	0.427			
S3	Target	0	0	0.8	0.6	0	0.2	0.949	0.037	0.267
	Result	0.187	0.153	0.563	0.540	0.303	0.313			
S4	Target	0.4	0.5	0	0	0.3	0.1	-0.657	0.115	0.233
	Result	0.173	0.290	0.413	0.580	0.479	0.340			
S5	Target	0.3	0.3	0	0	0.7	0	-0.030	0.119	0.033
	Result	0.474	0.346	0.445	0.427	0.466	0.494			
S6	Target	0	0	0.7	0.3	0	0.3	0.825	0.063	0.468
	Result	0.249	0.206	0.669	0.691	0.340	0.380			
S7	Target	0.4	0.2	0	0	0.4	0	0.726	0.075	0.318
	Result	0.381	0.465	0.247	0.354	0.681	0.335			
S8	Target	0.7	0.3	0	0	0.7	0	0.641	0.091	0.129
	Result	0.509	0.388	0.344	0.298	0.470	0.488			
S9	Target	0.3	0.7	0	0	0.7	0	0.680	0.098	0.136
	Result	0.446	0.431	0.386	0.443	0.637	0.383			
S10	Target	0	0	0.4	0.8	0	0	0.894	0.078	0.342
	Result	0.188	0.282	0.591	0.658	0.397	0.365			
S11	Target	0	0	0.6	0.6	0	0	0.906	0.045	0.399
	Result	0.121	0.235	0.570	0.581	0.373	0.247			
S12	Target	0	0.3	0.3	0.3	0.3	0.6	0.384	0.049	0.343
	Result	0.269	0.224	0.671	0.560	0.254	0.497			
S13	Target	0	0	0.8	0.5	0	0	0.794	0.078	0.636
	Result	0.197	0.165	0.779	0.579	0.176	0.603			
S14	Target	1	0	0	0	0	1	0.943	0.148	0.1154
	Result	0.492	0.349	0.362	0.321	0.344	0.603			
S15	Target	1	0	0	0	0	0	0.481	0.179	0.067
	Result	0.514	0.357	0.417	0.303	0.382	0.551			
Average:								**0.483**	**0.091**	**0.266**
Average (without negative PCCs)								**0.724**	**0.082**	**0.301**

We also conducted two separate interviews with two other experts, Robert Lamah, senior piano instructor at the Lebanese National Higher Conservatory of Music[1], and Joseph Khalifé, senior composer and musician-in-residence at the Lebanese American University[2]. After listening to MUSEC's compositions, both experts reported that the pieces' music quality was "very good", and described the compositions as being "beautiful" and "interesting". Ms. Lamah also enjoyed what he called MUSEC's composition "eccentricity". He suggested enhancing MUSECs composition functionality by offering musical experts the ability to suggest and teach musical modifications to the system so that it can better reach its target sentiments. As for Joseph Khalifé, he particularly stated that he would have had difficulties discerning between the MUSEC and human pieces we presented to him, had he not known the real pieces in advance. He also suggested extending MUSEC's composition functionality, to compose music that, not only expresses feelings, but rather "tells a story" related to those feelings and their evolution over the course of the composition.

However, both experts expressed concerns regarding the potential applications of the system (cf. Section 7), and the possibility of such systems replacing human composers in the future. Joseph Khalifé explained that "while a computer can compose music based on a specific and brief set of inputs, it probably cannot simulate the passion that goes into the music composition process".

6.4.4. Comparative Study

An experimental study comparing the effectiveness of MUSEC with existing approaches would have been interesting, and would have allowed us to further evaluate our method. Nonetheless, the diversity of related studies in the

[1] http://www.conservatory.gov.lb/disciplines/discipline/21

[2] http://comm.lau.edu.lb/joseph-khalife

literature (cf. Section 3), which significantly differ in their objectives, premises, and techniques used to perform composition, and most importantly the lack of a benchmark of sentiment-based compositions from these methods (few authors provide sample compositions by their methods, while none of them provide sentiment ratings), makes it difficult to perform a comparative empirical study. Hence, we currently settle for a qualitative comparison, depicting the main characteristics, commonalities, and differences between our approach and related methods.

Table 14 summarizes the main differences between our method and related solutions. On one hand, MUSEC: i) accepts as input a vector of scaled ($\in [0, 1]$) sentiment scores, or a piece of (MIDI) music that can be processed to extract a scaled sentiment vector, ii) adopts a categorical sentiment model (consisting of six basic sentiments) which is intuitive and simple to understand by users who will use it to express their input sentiments accordingly, iii) produces pieces that reflect a target crisp sentiment (e.g., love) or a collection of fuzzy sentiments (e.g., 65% happy, 20% sad, and 15% angry), iv) consists of an evolutionary composition module integrating a music sentiment-based machine learning module as its fitness evaluation function, in order to ensure flexibility and variability of MUSEC's compositions for similar input sentiment requests as the system gains experience and dynamically adapts to its user's particular inclinations, v) utilizes an extensible set of 18 different music-theoretic mutation operators (*trille*, *staccato*, *repeat*, *compress*, etc.), stochastically orchestrated within the evolutionary process, to add atomic and thematic variability to the compositions, vi) composes polyphonic pieces while adhering to "correct" music structure and coherence brought about the music-theoretic rules embedded within the *KB* module.

On the other hand, most existing solutions; i) accept different kinds of inputs (e.g., crisp scores (Hoeberechts M. et al. 2009), gestures (Morreale F. et al. 2016), texts (Kirke A. et al. 2017), or EEGs (Kirke A. et al. 2011)) to describe sentiments following the valence/arousal dimensional model, which are not always intuitive or easy to produce by non-expert users, ii) produce as output crisp-only or two-dimensional (*valence*/*arousal*) sentiment scores which are less descriptive in their sentiment expressiveness compared with the categorical model[1], iii) utilize *translation-based* models (Hoeberechts M. et al. 2009; Huang C. et al. 2013; Livingstone S. R. et al. 2010), or *mathematical model*-based techniques (Kirke A. et al. 2011; Kirke A. et al. 2017; Morreale F. et al. 2016), creating relatively simpler or less creative music where the main challenge lies in selecting appropriate inputs and converting them into music, iv) are mostly static and heavily reliant on predefined rules or rule-based models, v) produce mostly monophonic music (Hoeberechts M. et al. 2009; Kirke A. et al. 2017), while few approaches produce polyphonic music (Kirke A. et al. 2011; Morreale F. et al. 2016) although using author-developed heuristics to extend an initial monophonic melody into polyphonic music.

Table 14. Comparing our solution with existing approaches

Approach	Input	Sentiment Model	Sentiment Analysis	Composition Approach	Music Texture type
AMEE (Hoeberechts M. et al. 2009)	Crisp ($\in \{0, 1\}$) sentiment scores	Categorical (10 emotions)	None[2]	Rule-based, Autonomous	Monophonic music
TRAC (Kirke A. et al. 2017)	Movie Script	Valence/Arousal/Dominance	Syntactic structure and word-level analysis[3]	Rule-based, Assisted	Monophonic music
EEG System (Kirke A. et al. 2011)	EEG readings	Valence/Arousal	Rule-based	Rule-Based with heuristics, Assisted,	Polyphonic music
Huang C. & Lin E. (Huang C. et al. 2013)	Scaled ($\in [0, 1]$) Valence/Arousal scores	Valence/Arousal	None[2]	Rule-Based, Autonomous	Not specified
ROBIN (Morreale F. et al. 2016)	User gestures in music room	Valence/Arousal	User gesture analysis	Rule-based with heuristics, Autonomous	Polyphonic music
MUSEC	Scaled ($\in [0, 1]$) sentiment scores, or musical piece	Categorical (6 emotions)	Machine Learning (for musical inputs)	Evolutionary with embedded musical rules, Autonomous	Polyphonic music

6.4.5. Composer Efficiency

Following our complexity analysis in Section 5, *MC*'s time complexity simplifies to $O\left(N\times\left(\left(B\times S\times T\times N\right)+S^2\right)\right)$ where N represents the number of generations, B the branching factor, S the population size, and T the learner's training set size. Linear dependency w.r.t. training set size T comes down to the complexity of the *SL* module (which performs *fitness trimming* in the evolutionary process), and has been empirically evaluated in Section 6.3.4 (Fig. 18).

Results in Fig. 22 confirm *MC*'s: i) quadratic dependency on the number of generations N (Fig. 22.a), ii) almost linear dependency on the branching factor B (Fig. 22.b), and iii) almost quadratic dependency on population size S (Fig. 22.c).

---

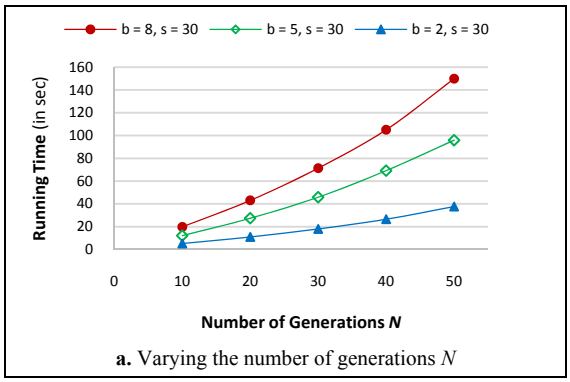
[1] Recall that states where both valence and arousal dimensions converge (e.g., both valence and arousal are high, or both are low) occur more often than states were they diverge, indicating a potential bias or ambiguity in the model (as stated by the model's creator in (Russell J. 1980)).

[2] Target sentiments (to be expressed by the composed piece) are manually provided by the user, where no automatic sentiment inference is involved.

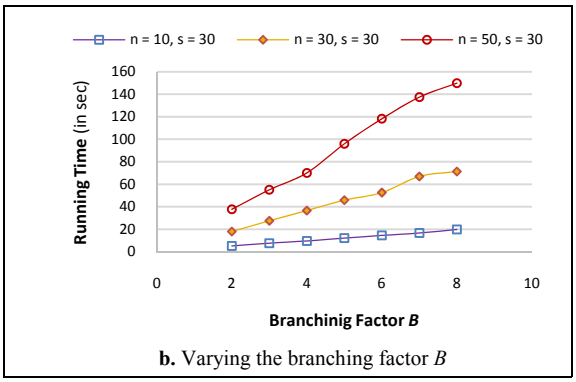
[3] Target sentiments are inferred from text parsed from a movie script, to compose an affective movie soundtrack.

7. Applications

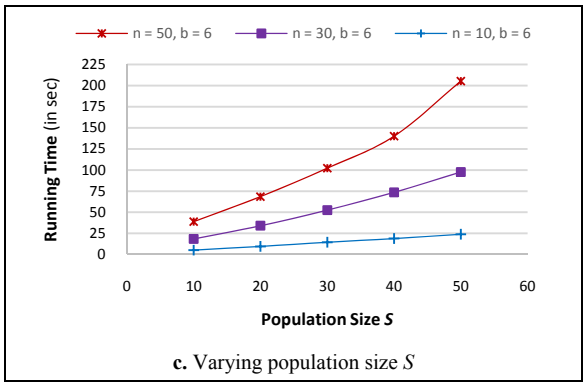
In this section, we discuss some of the main application scenarios which can benefit, in one way or another, from the MUSEC framework and its modules, ranging over: i) information retrieval, ii) music composition, iii) assistive music therapy, and iv) social intelligence.



a. Varying the number of generations *N*



b. Varying the branching factor *B*



c. Varying population size *S*

Fig. 22. *MC* component's running time w.r.t. the number of generations *N*, branching factor *B*, and population size *S*[1]

7.1. Information Retrieval

Sentiment-Based Music Retrieval: MUSEC's *SL* (sentiment learning) module can be utilized as a core component toward building a sentiment-based music retrieval engine, allowing to search for musical pieces based on the sentiments they express, versus traditional feature-based music retrieval approaches.

Searching for music based on traditional features: i) metadata and context features such as lyrics (Panda R. et al. 2013; Xiao H. et al. 2010), user or the artist profiles (Lin C.L. et al. 2016; Schedl M. et al. 2013), or ii) intrinsic music features, i.e., symbolic and frequency-domain features (Demopoulos R.J. et al. 2007; Schedl M. et al. 2014) such as the ones extracted by MUSEC*'s FP* module, though useful and powerful in their own right, rarely offer the chance to identify/discover new/original/unexpected musical pieces/genres. Searching for a song by its lyrics or by the artist information would return as a result songs having similar lyrics or songs written by the same artist, while searching based on low-level music features would return as a result musically-similar pieces. However, with sentiment-based music retrieval, users would simply state their target sentiments as a query, and then the system would find pieces that match their sentiment needs, i.e., pieces that could be very dissimilar in style/genre/music content but that would make them feel a certain way (happy, exited, etc.). In other words, users would not be thinking about the lyrics, the artist, or any musical features in formulating their queries, but would rather directly state their sentiment needs, while the system retrieves relevant musical pieces accordingly.

***Universal Retrieval System*:** MUSEC's *SL* module, combined with other sentiment analysis tools targeting different kinds of data (e.g., text, images, videos) could also serve as the building blocks of a universal sentiment-based retrieval system.

Most existing information retrieval (IR) systems are geared toward specific kinds of data: text, images, videos, music, etc. The most prominent IR engines are text-based (Baeza-Yates R. et al. 2011; L'Hadj L.S. et al. 2016), while recent efforts are leading toward image retrieval (Ayadi M.G. et al. 2016; Iakovidou C. et al. 2014), video retrieval (Chivadshetti P. et al. 2015; Mühling M. et al. 2016), and music retrieval systems (Hauger D. et al. 2013; Schedl M.

[1] All tests were run while considering a fixed *fitness-to-variability* ratio $R = 0.7$, and using all of *SL*'s experimental dataset for training, i.e., T=120 pieces.

et al. 2014) based on their intrinsic features. These systems, each used for their own goals, are divergent by design, since their target data are described using different features, and thus are mostly unrelated. In this context, a major challenge would be developing a full-fledged IR system handling multiple document types without it being a mere concatenation of their independent components. By integrating MUSEC's *SL* for musical sentimental analysis, with other relevant text/image/video-based sentiment analysis tools, all data types could then be queried with a single *sentiment query* (expressing the user's sentiment needs, in terms of crisp or fuzzy sentiment scores), producing data-type independent results which are based solely on the data's sentiment scores. In this context, an integrated sentiment analysis tool would allow to bridge the gap between data-types, and to create a universal sentiment-based retrieval system which returns any (type of) document answering the user's target sentiments. The users could provide as input a sentiment query, or an object (e.g., text, image, music) from which sentiment scores could be automatically inferred. Sentiment vectors would then be used as a common referential (mediator) space, bridging the gap between native (data-type specific) feature vectors describing every type of object. After matching the user's input sentiment vectors with those of objects in the repository, the most sentiment-similar objects would be returned to the user, ranked following their sentiment relevance (or any other ranking function deemed relevant by the user).

7.2. Music Composition

***Sentiment-based Music Composer*:** This is a straightforward usage of MUSEC: providing an algorithmic composer that uses sentiment predictions as a guide in its compositions, much like human composers writing music to reflect their state of mind and their emotions. This usage becomes more relevant when MUSEC is asked to compose new pieces based on an existing piece's expected emotional response. Using its sentiment learner (*SL*) module, MUSEC can accept as input a certain music piece, analyze the sentiments it expresses, and then use the produced sentiment vector to compose new music accordingly. In other words, users can not only input their target sentiment vectors, but they can also provide MUSEC with a certain piece as input, and then ask it to compose a completely different one such that the new composition reflects the same sentiments as the input piece. This can be useful for professional composers as well, who can require the system to generate a piece that is similar (in the sentiments it expresses) to one of their own compositions, which they can later utilize as a new source of inspiration for their future compositions.

***Automatic Composer Assistant*:** In addition to functioning alone as a full-fledged music composer, MUSEC could provide preliminary motifs and themes that would be further developed/completed by human composers, forming the basis of full-fledged compositions. This could provide composers with the inspiration needed to help them get out of their so-called *composer's block*, i.e., a mental block to getting music projects started or completed, which is a very familiar experience with composers (as well other forms of creative tasks, such as writing, painting, sculpting, etc.) (Yiu R. 2013). It would provide them with the motivation and insight to look outside of the box of their inherent composition styles, providing them with inspiration toward producing new and original compositions.

Another scenario would be for a human composer to write the beginning or parts of a piece, and then ask an automated composer assistant to compose the rest, while adhering to the themes and sentiments expressed in the initial parts. This scenario was suggested by renowned Lebanese composer and musician Jean Marie Riachi[1], after participating in a live tutorial event[2] during which we demonstrated and tested the different functionalities of MUSEC. Such functionality, he added, could help him gain valuable time in the composition process, focusing on the innovative tasks of the process, while helping him increase his productivity.

7.3. Assistive Music Therapy

Music sentiment analysis and composition could also be useful is assistive music therapy, a novel discipline where music is used within a therapeutic relationship to address emotional, cognitive, and social needs of individuals (Dell A.G. et al. 2011; McChord K.A. 2004). After assessing the needs of each patient, the music therapist provides the indicated treatment including creating, singing, moving to, and/or listening to music. Music therapy has been shown effective in various areas such as: physical rehabilitation, increasing people's motivation to become engaged in their treatment, helping restore patients' memories and enhance their moods, and providing an outlet for expressing feelings (especially when dealing with autistic children) (Wan C.Y. et al. 2011; Whipple J. 2004).

One area which can specifically benefit from these techniques is experimental psychology, where an automated music-based sentiment analysis tool would allow the psychologist to analyze the primary emotions of their patients when listening to certain pieces of music, selecting musical pieces in line with a therapeutically-appropriate mood or emotional state, in order to characterize their psychological states accordingly (e.g., analyzing whether the patient's sentiment perceptions match the expected sentiments, and highlighting unexpected sentiment perceptions which could help diagnose the case or prescribe treatment accordingly). This could be especially helpful in treating patients with Autism Spectrum Disorder (ASD), by learning their emotional states and helping them express their emotions through music, e.g., (Kim J. et al. 2009; See C.M. 2012; Whipple J. 2004). This is especially useful with low functioning ASD

[1] https://sv.wikipedia.org/wiki/Jean-Marie_Riachi

[2] http://www.lau.edu.lb/news-events/news/archive/music_composers_face_off_with_/. The event included an active participation from a live audience of LAU students, faculty, staff, and friends, who helped rate MUSEC's compositions and evaluate its sentiment scoring accuracy.

patients who have serious difficulties in writing and reading texts (traditional means of communication), and tend to be more interactive to images and music. The psychologist could ask MUSEC to compose different pieces expressing different emotions she wishes to evaluate with her ASD patients, and then ask the patients to listen to the pieces and identify the feelings expressed: either verbally (if possible), or by pinpointing other music pieces expressing the same sentiments. If the selected target piece's scores are close to the reference piece's scores, this means that the patient has a good perception of sentiments in music. Else if the scores are quite different from those of the reference piece, this means that the patient is not able to recognize the proper feelings behind the music piece, and would need to be followed accordingly by the psychologist. The same process could be applied by combining other non-verbal stimuli such as images or videos.

7.4. Social and Emotional Intelligence

Another area which can benefit from sentiment-based music analysis is *emotional intelligence*: understanding human emotions and reacting accordingly (Berrett L.F. 2017). More specifically, emotional intelligence is defined as the ability to identify, evaluate, and control one's own emotions, the emotions of others, and those of a group or people (Goleman D. 2005). It is a central building block of the broader area of *social intelligence*: the understanding and managing of others (Gkonou C. et al. 2017). While mainstream artificial intelligence (AI) research and applications traditionally emphasized the simulation of human cognitive aspects such as problem-solving and memory management, simulating emotional intelligence through automated sentiment analysis and affective computing has been receiving increased attention in the past couple of years, e.g., (Berrett L.F. 2017; Hovy 2015; Ravi K. et al. 2015). The main motivation for this research area is to simulate empathy: where an artificial agent would recognize, understand, and consider human emotions, and would adapt its behavior accordingly to give an appropriate response to those emotions.

While early affective computing approaches mainly focused on text-based analysis (Ravi K. et al. 2015), image and more recently music sentiment-based analysis are attracting increasing attention, with potential applications spanning many domains, including:

- *Social media applications*: allowing sentiment-based content analysis and human-like suggestions. While most current social sites focus on text and images, mainstream sound and music sentiment analysis in social sites could be around the corner, with early solutions focused on discussing social perspective on music (Hauger D. et al. 2013; Iren D. et al. 2016), and evaluating listening behavior and composition styles (following the interactions and collective perceptions of members of a society) (McAndrew S. et al. 2015; Zangerle E. et al. 2014),
- *Customer reviews analysis*: on products and services, such as movie and song reviews matched with their emotional contents, where movies and songs both include music that can also be processed for sentiment analysis (e.g., comparing the actual emotions expressed by listeners/viewers, versus the emotions intended by the song/movie creators, and the latter's success rates, target audience preferences, etc.) (Ravi K. et al. 2015; Xiao H. et al. 2010),
- *Information and tutoring tools*: developing digital instructors and modeling tutoring environments (Barrett F.S. et al. 2010; Rahim A. et al. 2015), including music as a teaching or monitoring medium to invoke memory or personality traits,
- *Analyzing voter moods* and mood swings (Ravi K. et al. 2015): considering not only voters' text posts, but also the music they are listening to during a certain period of time, to infer their sentiments accordingly,
- *Pervasive AI applications*: sensor-based linguistic, facial, gesture, behavioral, and specifically sound and music-based sentiment detection, analysis, and interaction (Holland S. et al. 2013; Yuanyuan W. 2014), e.g., allowing robotic systems to analyze different cues (through different input sensors) and automatically adapt their behaviors accordingly. Such a system could be implemented in a smart home, capturing sensory inputs from its residents and deciding on the right ambient music to play (joyful and relaxing, romantic, etc.). A similar solution could be applied to a smart automobile, which can collect sensory inputs from its driver and passengers, and then suggest or play music accordingly, e.g., cheering up the driver is she's feeling sad, calming the children down if they seem excited, or stirring and motivating passengers on their ride to a competitive sports game.

8. Conclusion

This paper introduces a novel Music Sentiment-based Expression and Composition framework titled MUSEC, to produce polyphonic and thematic MIDI musical pieces that express human emotions, while being appealing and enjoyable by listeners. MUSEC can serve many application domains ranging over sentiment-based music retrieval, music composition, assistive music therapy, and social intelligence. It consists of four main modules: i) MIDI music feature parser (*FP*), ii) music theory knowledge base (*KB*) including operations and rules to produce "correct" music, iii) music sentiment learner (*SL*) consisting of a non-parametric fuzzy classifier that learns to infer sentiments in music, and iv) music sentiment-based composer (*MC*) consisting of an evolutionary-developmental framework with specially tailored genetic operators to produce new, diversified, and sentiment-expressive music compositions. Experimental results reflect our approach's effectiveness in music sentiment learning and sentiment-based

composition. Time analyses underline the impact of the training set size, the number of generations, the branching factor, as well as the population size considered (among other factors) on learning and composition time. Our experimental prototype and results have been made available online, including a benchmark of 70 sentiment annotated pieces: the first significant dataset of sentiment-labeled MIDI music, to promote future research in this area.

Several MUSEC improvements lay ahead in the near future. First, we aim to further extend the *FP* module in order to consider a wider range of adapted low-level (spectral) (Costa Y. et al. 2017) and high-level (symbolic) music features (Panda R. et al. 2013), from which users can select and utilize the ones that best fit their needs (e.g., using the dominant key to describe well structured classical pieces, versus a 24-vector fuzzy key distributions with atonal pieces, to better portray the latter's key variations). Re-evaluating the importance of the chord progression feature and improving on its parsing heuristics, e.g., (Demopoulos R.J. et al. 2007; Kyogu L. 2008), would be especially interesting, in order to ensure that the full benefits of such a sophisticated feature can be reaped. Second, we plan to improve the design of our similarity evaluation function used within the *SL* module, morphing it into a learning (similarity-based) classifier on its own (Chen Y. et al. 2009), aiming to improve *SL*'s accuracy beyond the 0.63 *PCC* score it produced in our recent experiments. Exploring parametric deep learning models and comparing their performance with the non-parametric paradigm used in the current agent is yet another direction for improving *SL*'s performance. Third, we also aim to strengthen *MC*'s design, by leveraging machine learning or optimization techniques, e.g., (Hopfield J. and Tank D. 1985; Liu J. et al. 2010), and recent generative adversarial reasoning approaches, e.g., (Cai Z. et al. 2018; Cao Y. et al. 2019; Troiano L. et al. 2017), in order to learn *MC*'s optimal parametric configuration (e.g., chord distribution, mutation probabilities, modulation keys, etc., which are currently static or chosen manually by the user) producing compositions that not only reflect a target sentiment vector, but also reflect the "style" of the training compositions. For instance, the composer can be trained to "learn" a particular way to perform a chord from its training corpus, allowing mutating chords not only by using music theory, but also by using original "learned" ways of playing a chord (similarly to modern music compositions, where non-conventional chord expressions are becoming all too common). An intriguing possibility of this hybrid crossover between machine learning and evolutionary music composition is to allow the composer to learn and mimic the style of existing composers, e.g., mimicking the great Mozart for instance by learning all his compositions and then generating/evolving new compositions accordingly, which is both a challenging and a thought-provoking prospect.

Acknowledgments

This study is partly funded by the National Council for Scientific Research - Lebanon (CNRS-L), by LAU, as well as the Fulbright Visiting Scholar program (sponsored by the US Department of State). Special thanks go to music experts: Anthony Bou Fayad, Robert Lamah, and Joseph Khalifé, who helped evaluate the synthetic compositions, as well as Jean Marie Riachi for his participation in a live demonstration of the system. We would also like to thank the non-expert testers (including LAU students, faculty, staff, and friends) who volunteered to participate in the experimental evaluation.

Compliance with Ethical Standards

- Disclosure of potential conflicts of interest
  1. Funding: *The study was partly funded by:*
     - *National Council for Scientific Research – Lebanon, CNRS-L (grant number: NCSR-LAU#887)*
     - *Lebanese American University, LAU (grant number: SOERC1516R003)*
     - *Fulbright Visiting Scholar program, sponsored by the US Department of State (grant number: PS00232737)*
  2. Conflict of Interest: *The authors declare that they have no conflict of interest.*
- Research involving human participants and/or animals
  1. Statement of human rights: *Ethical approval: For this type of study formal consent is not required.*
  2. Statement on the Welfare of Animals: *Ethical approval: This article does not contain any studies with animals performed by any of the authors.*
- Informed consent: *Additional informed consent was obtained from all individual participants for whom identifying information is included in this article.*

References

Abbasi A., Chen H., Thoms S. and Fu T. (2008). Affect Analysis of Web Forums and Blogs Using Correlation Ensembles. IEEE Transactions on Knowledge and Data Engineering 20(9):1168-1180.

Abu Arqub O. (2017). Adaptation of Reproducing Kernel Algorithm for Solving Fuzzy Fredholm-Volterra Integrodifferential Equations. Neural Computing and Applications 28(7): 1591-1610.

Abu Arqub O. and Abo-Hammour Z. S. (2014). Numerical Solution of Systems of Second-Order Boundary Value Problems using Continuous Genetic Algorithm. Information Sciences 279: 396-415.

Abu Arqub O., Al-Smadi M., Momani S. and Hayat T. (2016). Numerical Solutions of Fuzzy Differential Equations using Reproducing Kernel Hilbert Space Method. Soft Computing 20(8): 3283-3302.

Adiloglu K. and Alpaslan F.N. (2007). A Machine Learning Approach to Two-Voice Counterpoint Composition. Knowledge-Based Systems 20(3):300-309.

Ayadi M.G., Bouslimi R. and Akaichi J. (2016). A Medical Image Retrieval Scheme with Relevance Feedback through a Medical Social Network. Social Netw. Analys. Mining 6(1): 53:1-53:23.
Baeza-Yates R. and Ribeiro-Neto B. (2011). Modern Information Retrieval: The Concepts and Technology behind Search. ACM Press Books, Addison-Wesley Professional, 2nd Ed. p. 944.
Barrett F.S., Grimm K. J., Robins R. W. , Wildschut T. and Sedikides C. (2010). Music-Evoked Nostalgia: Affect, Memory, and Personality. Emotion 10(3):390–403.
Bas De Haas W., Veltkamp R.C. and Wiering F. (2008). Tonal Pitch Step Distance: a Similarity Measure for Chord Progressions. International Society of Music Information Retrieval (ISMIR) pp. 51-56.
Bas De Haas W., Wiering F. and Veltkamp R.C. (2013). A Geometrical Distance Measure for Determining the Similarity of Musical Harmony. International Journal of Multimedia Information Retrieval 2(3):189-202.
Berrett L.F. (2017). How Emotions are Made: The Secret Life of the Brain. Macmillan, United kingdom 425 pages.
Boden M. A. (1994). "Precis of The creative mind: Myths and mechanisms. Behavioral and Brain Sciences 17(3):519-570.
Bradley M. and Lang P. (1999). Affective Norms for English Words (ANEW): Instruction Manual and Affective Ratings. Technical Report C-1 University of Florida: Center for Research in Psychophysiology.
Burton A. R. (1998). A Hybrid Neuro-Genetic Pattern Evolution System Applied to Musical Composition. Ph.D. thesis, University of Surrey, UK.
Cai Z. and Hu H. (2018). Session-Aware Music Recommendation via a Generative Model Approach." Soft Computing 22(3): 1023-1031.
Cao Y., Jia L., Chen Y, Lin N., Yang C., Zhang B., Liu Z., Li X. and Dai H. (2019). Recent Advances of Generative Adversarial Networks in Computer Vision. IEEE Access 7: 14985-15006.
Carnie A. (2013). "Syntax: A Generative Introduction. 3rd edition. Malden, MA: Wiley-Blackwell p. 519.
Chen Y., Garcia E., Gupta M., Rahimi A. and Cazzanti L. (2009). Similarity-based Classification: Concepts and Algorithms. Journal of Machine Learning Research 10:747-776.
Chivadshetti P., Sadafale K. and Thakare K. (2015). Content based Video Retrieval using Integrated Feature Extraction and Personalization of Results. International Conference on Information Processing (ICIP'15) DOI: 10.1109/INFOP.2015.7489372.
Cormen T.H., Leiserson C.E., Rivest R.L. and S. C. (2009). Introduction to Algorithms (3rd Ed.). MIT Press and McGraw-Hill. .
Costa Y., Oliveira L. and Silla C. Jr. (2017). An Evaluation of Convolutional Neural Networks for Music Classification using Spectrograms. Applied Soft Computing 52:28-38.
Danhauser A. (1994). "Theory of Music (French)." Paris: Henri Lemoine p. 128, Original edition published in 1950.
Dell A.G., Newton D.A. and Petroff J.G. (2011). "Assistive Technology in the Classroom: Enhancing the School Experiences of Students with Disabilities." Pearson, 2nd Ed. p. 384.
Demopoulos R.J. and Katchabaw M.J. (2007). "Music Information Retrieval: A Survey of Issues and Approaches." Technical Report #677, Department of Computer Science, University of Western Ontario, Ca p. 71.
Di Nunzio A. (2014). Illiac Suite for String Quartet. http://www.musicainformatica.org/topics/illiac-suite.php Accessed July 2017.
Diaz-Jerez G. (2011). Composing with Melomics: Delving into the Computational World for Musical Inspiration. MIT Press Journals 21:3-14.
Dubois R.L. (2003). Applications of Generative String-Substitution Systems in Computer Music. Ph.D. Dissertation, Columbia University.
Ekman P. (1993). Facial Expression of Emotion. American Psychologist 48:384-392.
Epstein R., Roberts G. and Beber G. (2009). Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer. Springer, 2009 Ed., 517 p.
F. Amin, A. Fahmi, S. Abdullah, A. Ali, R. Ahmed and F. Ghani (2017). Triangular Cubic Linguistic Hesitant Fuzzy Aggregation Operators and their Application in Group Decision Making. Journal of Intelligent and Fuzzy Systems 32(2): 1217-1228 34:2401-2416.
Fahmi A., Abdullah S., Amin F., Ali A. and Khan W. A. (2018). Some Geometric Operators with Triangular Cubic Linguistic Hesitant Fuzzy Number and their Application in Group Decision-Making. Journal of Intelligent and Fuzzy Systems 35(2): 2485-2499.
Fahmi A., Abdullah S., Amin F. and K. M.: (2019). Trapezoidal Cubic Fuzzy Number Einstein Hybrid Weighted Averaging Operators and its Application to Decision Making. Soft Computing 24(14): 5753-5783.
Fahmi A., Abdullah S., Amin F., S. N. and Ali A. (2017). Aggregation Operators on Triangular Cubic Fuzzy Numbers and its Application to Multi-Criteria Decision Making Problems. Journal of Intelligent and Fuzzy Systems 33(6): 3323-3337.
Fernandez J. and Vico F. (2013). AI Methods in Algorithmic Composition: A Comprehensive Survey. Journal of Artificial Intelligence Research 48:513–582.
Fleischman M.B. and Deb K. R. (2013). Displaying Estimated Social Interest in Time-based Media. U.S. Patent No. 8,516,374. 20 Aug. 2013.
Freeman J. (2015). Survey of Music Technology. Coursera. Available: https://www.coursera.org/learn/music-technology, (Accessed Jul. 2017).
Ghosh A. and Strehl J. (2003). Cluster Ensembles—a Knowledge Reuse Framework for Combining Multiple Partitions. The Journal of Machine Learning Research 3:583-617.
Gkonou C. and Mercer S. (2017). Understanding Emotional and Social Intelligence among English Language Teachers. ELT Research Papers 17.03, British Council, ISBN 978-0-86355-842-9.
Goldberg D. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 432 pages.
Goleman D. (2005). Emotional Intelligence: Why It Can Matter More Than IQ. Bantam Books 10th Anniversary edition 384 pages.
Hauger D., Schedl M., Kosir A. and Tkalcic M. (2013). The Million Musical Tweet Dataset: What We Can Learn From Microblogs. In Proc. of the 14th International Society for Music Information Retrieval Conference (ISMIR'13).
Hevner K. (1935). The Affective Character of the Major and Minor Modes in Music. The American Journal of Psychology 47(1):03-118.
Hiller L. (1970). Music Composed with Computers: a Historical Survey in The Computer and Music. Edited by Harry B. Lincoln, Cornell University Press pp. 42 – 97.
Hiller L. and Isaaccson L. (1959). Experimental Music: Composition with an Electronic Computer. McGraw-Hill, New York p. 218.
Hoeberechts M. and Shantz J. (2009). Realtime Emotional Adaptation in Automated Composition. Proceedings of Audio Mostly pp. 1-8.
Holland S., Wilkie K., Mulholland P. and Seago A. (2013). Music and Human-Computer Interaction. Springer Series on Cultural Computing ISBN-10: 144712989X, 292 pages.
Hopfield J. and Tank D. (1985). Neural Computation of Decisions in Optimization Problems. Biological Cybernetics 52(3):52–141.
Hovy, E. (2015). What are Sentiment, Affect, and Emotion? Applying the Methodology of Michael Zock to Sentiment Analysis. N. Gala et al. (eds.), Language Production, Cognition, and the Lexicon, Text, Speech and Language Technology, 48:13-24.
Huang C. and Lin E. (2013). An Emotion-based Method to Perform Algorithmic Composition. The 3rd Inter. Conference on Music & Emotion, pp. 244-247.
Husarik S. (1983). John Cage and LeJaren Hiller: HPSCHD, 1969. American Music 1(2):1-21
Iakovidou C., Anagnostopoulos N., Kapoutsis A., Chatzichristofis Y. and Boutalis Y. (2014). Searching images with MPEG-7 (& MPEG-7-like) Powered Localized dEscriptors: The SIMPLE answer to effective Content Based Image Retrieval. 12th International Workshop on Content-Based Multimedia Indexing (CBMI) pp. 18-20.
Iren D., Liem C., Yang J. and Bozzon A. (2016). Using Social Media to Reveal Social and Collective Perspectives on Music. International ACM Conference on Web Sciencs (WebSci'16) pp. Pages 296-300, Hannover, Germany.
Katayose H., Kato H., Imai M. and Inokuchi S. (1989). An Approach to an Artificial Music Expert. International Computer Music Conference pp. 138-146.
Keller J.M., Gray M.R. and Givens J.A. (1985). A Fuzzy k-Nearest Neighbor Algorithm. IEEE Transactions on Systems, Man, and Cybernetics, 4:580-585.
Kim J., Wigram T. and Gold C. (2009). Emotional, Motivational and Interpersonal Responsiveness of Children with Autism in Improvisational Music Therapy. Autism 13(4), 389-409. PMID: 19535468.
Kirke A. and Miranda E. (2011). Combining EEG Frontal Asymmetry Studies with Affective Algortihmic Composition and Expressive Performance Model. In International Computer Music Conference, Huddersfield p. 4.
Kirke A. and Miranda E. (2017). Aiding Soundtrack Composer Creativity through Automated Film Script-profiled Algorithmic Composition. Journal of Creative Music Systems Vol. 1, Issue 2.
Kirke A. and Miranda E.R. (2009). A Survey of Computer Systems for Expressive Music Performance. ACM Computing Surveys (CSUR), 42(1), 3.
Kotsiantis S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica 31:249-268.
Kyogu L. (2008). A System for Acoustic Chord Transcription and Key Extraction from Audio using Hidden Markov Models Trained on Synthesized Audio. Diss. Department of Music, Stanford University.

L'Hadj L.S., Boughanem M. and Amrouche K. (2016). Enhancing Information Retrieval through Concept-based Language Modeling and Semantic Smoothing. Journal of the Association for Information Science and Technology (JASIST) 67(12): 2909-2927.
Lin C.L., Shih Y.H., Tzeng G.H. and Yu H.C. (2016). A Service Selection Model for Digital Music Service Platforms using a Hybrid MCDM Approach. Applied Soft Computing 48:385-430.
Liu J., Zhong W. and Jiao L. (2010). A Multiagent Evolutionary Algorithm for Combinatorial Optimization Problems. IEEE Transactions on Systems, Man, and Cybernetics 40(1):229-240
Livingstone S. R. et al. (2010). Changing Musical Emotion: A Computational Rule System for Modifying Score and Performance. Computer Music Journal 34(1):41-64.
Manousakis S. (2006). Musical L-systems. Koninklijk Conservatorium, The Hague Master Thesis.
Marques M., Oliveira V., Vieira S. and Rosa A.C. (2000). Music Composition using Genetic Evolutionary Algorithms. Proceedings of the IEEE Conference on Evolutionary Computation IEEE Press, New York, NY.
Matic D. (2010). A Genetic Algorithm for Composing Music. Yugoslav Journal of Operations Research 20(1):157-177
McAndrew S. and Everett M. (2015). Music as Collective Invention: A Social Network Analysis of Composers. Cultural Sociology Vol 9, Issue 1, British Sociological Association.
McChord K.A. (2004). Moving Beyond "That's all I can do": Encouraging Musical Creativity in Children with Learning Disabilities. Bulletin of the Counsil of Research in Music Education 159:23-32.
McCormack J. (1996). Grammar-based Music Composition. Complex Systems pp. 321-336.
Molina A., Daniel D., Moya J.C. and Vico F.J. (2016). An Evo-Devo System for Algorithmic Composition that Actually Works. Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion. ACM pp. 37-38.
Morreale F. and d. Angeli A. (2016). Collaborating with an Autonomous Agent to Generate Affective Music. ACM Transactions on Computers in Entertainment (ACM CIE) 14(3):1-21.
Mühling M. et al. (2016). Content-based Video Retrieval in Historical Collections of the German Broadcasting Archive. International Conference on Theory and Practice of Digital Libraries (TPLD'16) pp 67-78.
O'Connor B., Balasubramanyan R., Routledge B.R. and Smith N.A. (2010). From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media pp. 122-129.
Orio N. (2006). Music Retrieval: A Tutorial and Review. Foundations and Trends in Information Retrieval 1(11), p. 90.
Ozcan E. and Erçal T. (2008). A Genetic Algorithm for Generating Improvized Music. Lecture Notes in Computer Science, Springer Heidelberg, 4926.
Panda R., Malheiro R., Rocha B., Oliveira A. and Paiva R. P. (2013). Multi-Modal Music Emotion Recognition: A New Dataset, Methodology and Comparative Analysis. 10th International Symposium on Computer Music Multidisciplinary Research (CMMR) pp. 1-13.
Papadopoulos G. and Wiggins G. (1999). AI Methods for Algorithmic Composition: A survey, a Critical View and Future Prospects. AISB Symposium on Musical Creativity pp. 110-117.
Pavlov S., Olsson C., Svensson C., Anderling V., Wikner J. and Andreasson O. (2014). Generation of music through genetic algorithms. Bachelor's Thesis, University of Gothenburg, Sweeden.
Prusinkiewicz P. and Lindenmayer A. (1990). The Algorithmic Beauty of Plants. Springer-Verlag New York P. 228.
Rahim A., Civelek I. and Liang F.H. (2015). A Model of Department chairs' Social Intelligence & Faculty Members' Turnover Intention. Intelligence 53:65–71.
Ravi K. and Ravi V. (2015). A Survey on Opinion Mining and Sentiment Analysis: Tasks, Approaches and Applications. Knowledge-Based Systems 89:14–46.
Reimer M. A. and Garnett G. E. (2014). A Hierarchical System for Autonomous Musical Creation. Tenth Artificial Intelligence and Interactive Digital Entertainment Conference pp. 45-49.
Russell J. (1980). A Circumplex Model of Affect. Journal of Personality and Social Psychology 6(39):1161-1980.
SACEM (Society of Auhors, C., and Editors of Music) (2016). AIVA: Artificial Intelligence Virtual Artist. Available online at: http://www.aiva.ai/about Accessed in May 2018.
Sandred O., Laurson M. and Kuuskankare M. (2009). Revisiting the Illiac Suite–a Rule-based Approach to Stochastic Processes. Sonic Ideas/Ideas Sonicas 2 pp. 42-46.
Schank R. C. and Cleary C. (1995). Making Machines Creative. In: S Smith, T B Ward & R A Finke (eds) The Creative Cognition Approach. MIT Press pp. 229-247.
Schedl M., Flexer A. and Urbano J. (2013). The Neglected User in Music Information Retrieval Research. Journal of Intelligent Information Systems (JIIS) 41(3): 523–539.
Schedl M., Gómez E. and Urbano J. (2014). Music Information Retrieval: Recent Developments and Applications. Foundations and Trends in Information Retrieval 8(2-3):127-161.
See C.M. (2012). The Use of Music and Movement Therapy to Modify Behaviour of Children with Autism. Pertanika J. Soc. Sci. & Hum., 20(4): 1103-1116.
Serra M. H. (1993). Stochastic Composition and Stochastic Timbre: Gendy3 by Iannis Xenakis. Perspectives of New Music pp. 236-257.
Shang W. et al. (2005). An Improved kNN Algorithm–Fuzzy kNN. Computational Intelligence and Security, pp. 741-746.
Song Y., Dixon S. and Pearce M. (2012). A Survey of Music Recommendation Systems and Future Perspectives. 9th International Symposium on Computer Music Modeling and Retrieval pp. 395-410.
Subasic P. and Huettner A. (2001). Affect Analysis of Text Using Fuzzy Semantic Typing. IEEE Trans. Fuzzy Systems 9(4):483-496.
Temperley D. (2002). A Bayesian Approach to Key-Finding. International Conference on Music and Artificial Intelligence, LNAI 2445 pp. 195-206.
Troiano L., Birtolo C. and Armenise R. (2017). Modeling and Predicting the User Next Input by Bayesian Reasoning. Soft Computing 21(6): 1583-1600.
Verbeurgt K., Fayer M. and Dinolfo M. (2004). A Hybrid Neural-Markov Approach for Learning to Compose Music by Example. Conference of the Canadian Society for Computational Studies of Intelligence pp. 480-484.
W. Bas de Haas, R. C. V., Frans Wiering (2008). Tonal Pitch Step Distance: A Similarity Measure for Chord Progressions. ISMIR: 51-56.
Wan C.Y. et al. (2011). Auditory-Motor Mapping Training as an Intervention to Facilitate Speech Output in Non-Verbal Children with Autism: A Proof of Concept Study. PLoS ONE 6(9): e25505. doi:10.1371/journal.pone.0025505.
Whipple J. (2004). Music in Intervention for Children and Adolescents with Autism: a Meta-Analysis. Journal of Music Therapy 41(2):90-106. PubMed PMID: 15307805.
Whitley D. and Sutton A.M. (2012). Genetic Algorithms — A Survey of Models and Methods. Handbook of Natural Computing pp 637-671.
Wohlfahrt-Laymanna J. and Heimbürgerb A. (2017). Content Aware Music Analysis with Multi-Dimensional Similarity Measure. Information Modelling and Knowledge Bases XXVIII, pp. 292-303.
Wolfram Tones Inc. (2005). WolframTones: How It Works - Scientific Foundations. http://tones.wolfram.com/about/how-it-works Accessed June 2019.
Worth P. and Stepney S. (2005). Growing Music: Musical Interpretations of L-Systems. Workshop on Applications of Evolutionary Computing, pp. 545-550.
Xiao H. and Downie S.J. (2010). "Improving Mood Classification in Music Digital Libraries by Combining Lyrics and Audio. Proceedings of the 10th annual Joint Conference on Digital libraries. ACM pp. 159-168.
Yiu R. (2013). A Composer's Imagining of Musical Tradition and The Reinvention of Heritage. Doctoral Thesis, City, University of London Institutional Repository, p. 104.
Yuanyuan W. (2014). Music Emotion Cognition Model and Interactive Technology. IEEE Workshop on Electronics, Computer and Applications DOI: 10.1109/IWECA.2014.6845608, Ottawa, Canada.
Zangerle E., Pichl M., Gassler W. and Specht G. (2014). #nowplaying Music Dataset: Extracting Listening Behavior from Twitter. In Proc. of the First International Workshop on Internet-Scale Multimedia Management (WISMM), ACM Multimedia Orlando, Florida, USA, 2014.
Zentner M. and Eerola T. (2010). Self-Report Measures and Models. Handbook of Music and Emotion: Theory, Research, Applications pp.187-221.
Zenz V. (2007). Automatic Chord Detection in Polyphonic Audio Data. Master's Thesis University of Wien, Austria.

Appendix

Appendix I. Pseudo-code for *Chord_Realizations* Function

The pseudo-code for the *chord-Realizations* recursive function is shown in Fig. 23. The *isValid* method mentioned in this pseudo-code refers to a method from *MUSEC*'s *KB* module which verifies that the progression from one chord to another is music-theoretically valid (i.e., conforming to all the rules built into *KB*, such as: no consecutive fifths, resolution of sensible tone, etc.).

```
Algorithm: Chord_Realizations

Inputs:  New chord Chroma Set: ChromaSet          // chromas that make up the chord to be realized, provided by the KB module
         Last Chord's frontier: frontier          // set of MIDI pitches to resolve
         Note array: notes                        // including notes to be used in realizations
         New Chord ID (root and type): chordID    // provided by the KB module
         Last Chord currently in individual. previousChord

Output: A Set of valid note realizations for the chord: validSet
Begin
   Initialize a list of valid chord realizations: validSet
   If chromaSet ≠ ∅                               // Some notes in the new chord are yet to be realized      1
   {                                                                                                        2
      For every Chroma chroma in ChromaSet                                                                  3
      {                                                                                                     4
         For every MIDI Pitch pitch in frontier                                                             5
         {                                                                                                  6
            Clone frontier to create frontierNew                                                            7
            Remove pitch from frontierNew                                                                   8
            Clone notes and ChromaSet to create notesNew and chromaSetNew respectively                      9
            Remove chroma from chromaSetNew                                                                 10
                                                                                                            11
                        // At this point, the algorithm has decided to resolve a pitch from the old chord into chroma in new chord
                        // It will now determine pitches through which chroma can be realized in the new chord

            Compute difference as being (pitch – chroma) % 12
                                                                                                            12
                        //MIDI pitches >=20 for piano, so difference is positive. This difference

            Compute MIDIPitch1 = pitch – difference      // First realization of chroma (dropping transition)
            Compute MIDIPitch2 = pitch + 12 – difference // Second realization of chroma (rising transition) 13
            Add MIDIPitch1 to notesNew                   // 1st realization                                   14
            Add all valid chords from new recursive call on chromaSetNew, frontierNew, notesNew, ChordID to validSet;  15
                                                         // Add answers from 1st recursive call               16

            Reset notesNew as being a new clone of notes
            Add MIDIPitch2 to notesNew                   // 2nd realization                                   17
            Add all valid chords from new recursive call on chromaSetNew, frontierNew, notesNew, ChordID to validSet;  18
                                                         // Add answers from 2nd recursive call               19
         }
      }
      Return validSet                                                                                       20
   }                                                                                                        21
   else                                                                                                     22
   {                                                 // All chromas processed, terminate recursion here and create the new chord  23
                                                                                                            24
      Instantiate the new chord newChord                                                                    25
      Set newChord's key to the individual's current key;
      Set newChord's chord type to ChordID                                                                  26
      Convert notes MIDI Pitch array to note objects and add them to newChord's note List                   27
      Define newChord's frontier based on ChordID and notes                                                 28
                                                      //Validate progression using Knowledge Base         29
      If isValid(newChord, previousChord)             //Validate music-theoretically                      30
      {                                                                                                     31
         Add newChord to validSet                                                                           32
      }                                                                                                     33
     Return validSet;                                                                                       34
   }                                                                                                        35
End
```

Fig. 23. Pseudo-code of *Chord_Realizations* function

Appendix II. Mutation Operators

II.1. *Trille* operator

The *trille* mutation operator affects the highest note, in terms of MIDI pitch, played in the chord. Its overall operation is visualized in Fig. 24. Based on the current key of the mutated chord, this operator retrieves the next note above the previously mentioned note in the key, using MUSEC's *KB* module. It then proceeds to alternate rapidly between the two aforementioned notes over the first half-beat of the chord being mutated. This mutation mainly increases overall piece note density and note onset density. For more variability, a random decision is made at the time of the mutation's execution to decide the range in which the alternating notes are performed. The following outcomes are possible: i) first quarter-beat, ii) second quarter-beat, and iii) full half-beat. Alternatively, based on the previous mutations that have affected the chord being mutated, the *trille* operator can also affect the final half-beat rather than the first half-beat of the given chord (i.e., at the end of a chord's execution rather than at the beginning).

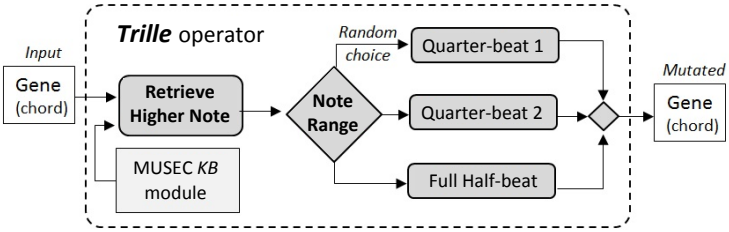


Fig. 24. Simplified activity diagram describing the *trille* mutation operator

II.2. *Staccato* operator

This mutation affects all the notes being played as part of a chord. It mainly alters the way they are performed. In music theory, a "staccato" refers to a note being played in a manner detached and separated from the others (such that its own duration is very short). This operator reduces the duration of every note to an eighth beat so as to emulate this effect.

II.3. *Repeat* operator

This mutation operator repeats the notes being played as part of a chord's realization a second time within the current duration of the chord. Fig. 25 provides a simplified activity diagram describing the process. It basically divides the current chord duration into two parts based on a random decision, takes all the notes currently being played, and then puts a copy of all the notes in both divisions. In order to maximize variability while maintaining musical structure, three divisions are allowed: i) 0.75/ 0.25 , where the first duplicate receives three quarters of the total chord duration and the latter duplicate receives the remaining quarter, ii) 0.5/0.5, an equal split of duration amongst the two duplicates, and ii) 0.25/0.75, as the inverse of the first division. To avoid overlap between notes, the copies are compressed to fit their new total duration (i.e., the individual note duration and onset time are scaled down to fit the smaller duplicate size). Any notes that have too short a duration following compression are discarded.

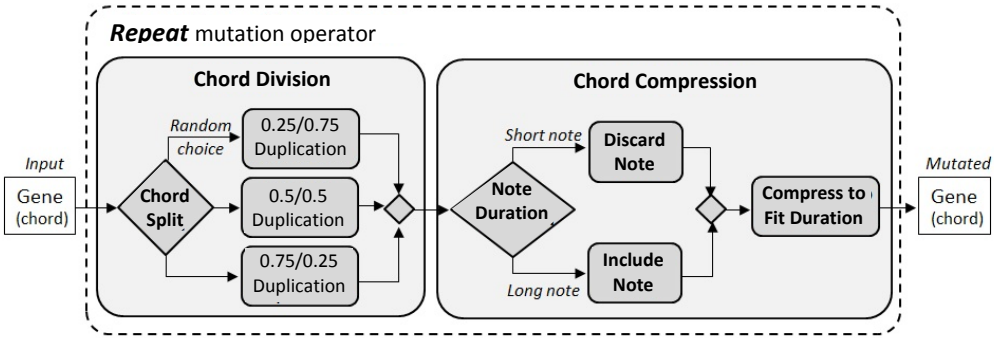


Fig. 25. Simplified activity diagram describing the *repeat* mutation operator

II.4. *Compress* operator

This mutation affects the duration of a chord, and aims to raise overall piece note and note onset density. Unlike the *repeat* operator where the notes are duplicated, compressed, and then repeated across the whole chord duration, the compress operator only performs compression, thereby shrinking the chord's overall duration. It halves a chord's duration, and applies to the chord's notes following the same compression process used with the *repeat* mutation operator.

II.5. *Extend* operator

The *extend* operator, as its name suggests, extends the length of a chord. Unlike the *compress* and *repeat* operators, this operator aims to lower the piece note and note onset densities. It first randomly decides an extension value in beats: either half a beat, or a full beat. Then, it identifies the notes that are played (i.e., that are audible) at the end of the chord's duration and increases their duration by the extension value, whilst also increasing overall chord duration.

II.6. *Silence* operator

Similar to the *extend* operator, the *silence* operator lowers overall note and note onset density by extending the mutated chord's duration. However, this operator does not add or extend any notes, instead creating a silence at the end of the chord. This mutation emulates the "rest" concept in music theory.

II.7. *Single Suspension* operator

The *single suspension* operator affects the notes that make up the chord's definition (i.e., its root, third, and fifth notes) as specified in its frontier. This mutation identifies the note realizations of the frontier notes then randomly chooses one of them and delays its entry by a quarter-beat, thus increasing note onset density. Note that since no new notes are added through this mutation, and no changes are made to a chord's duration, note density is preserved. However, note onset density increases since notes that would otherwise be played together are now played separately.

II.8. *Progressive Entrance* operator

This mutation, like the *single suspension* mutation, also increases note onset density. Unlike the latter operator however, *progressive entrance* rather affects all but one of the frontier notes' onsets. A simplified activity diagram highlighting its behavior is shown in Fig. 26. It randomly chooses a starting distribution, spreading over a half-beat duration, indicating the beat timing at which every frontier note should be played. For structural and musical purposes, the smallest beat timing unit used for this process is the eighth beat. This process produces 20 possible distributions, three of which are shown in the activity diagram for illustration purposes. Due to this distribution's decision process, some frontier notes could be dropped. This occurs when the duration distribution assigns zero values for certain frontier notes, which would subsequently produce unexpected and musically diverse results, whilst also lowering note density and note onset density in a novel way. Finally, the operator plays the surviving frontier notes sequentially (from lowest to highest pitch) following the chosen distribution.

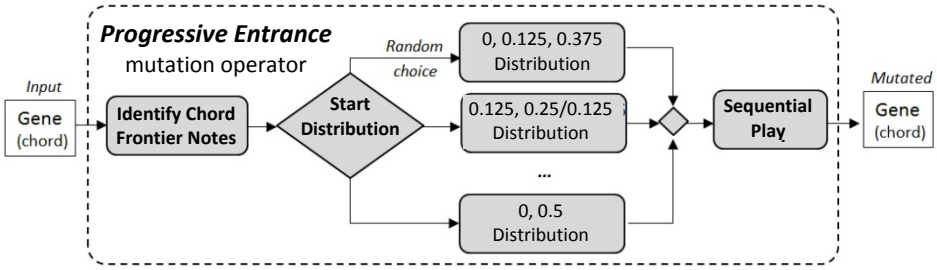


Fig. 26. Simplified activity diagram describing the *progressing entrance* mutation operator

II.9. *Nota Cambiata* operator

The *nota cambiata* operator emulates the music-theoretical principle of *nota cambiata*, and is used to decorate the highest note of a chord. In a typical *nota cambiata* realization, the decorated note is preceded by three other notes in its key. In order, these are the notes: i) a third above, ii) a second above, and ii) a second below, it in its chord's key. The operator assigns a random duration to each of these notes following the same logic as the one described with the *progressive entrance* operator (i.e., eighth beat time step, half beat total duration), meaning that notes amongst the decorative notes could also be dropped. It also delays the decorated note's onset by half a beat so as to accommodate the decoration notes. As a result, this operator increases note density and note onset density in the given musical piece.

II.10. *Appoggiatura* operator

Another music-theoretically inspired operator, the *appoggiatura*, precedes the decorated note with an adjacent note in its key, typically the note a second above it or a second below it in the given key, analogously to the music-theoretic "appoggiatura" decoration. The operator first identifies the highest note in the chord, then retrieves both of its adjacent notes in the chord's key using MUSEC's *KB* module, and randomly chooses one of them to add to the first half beat of the piece. Similarly to the *nota cambiata* operator, the decorated note is delayed by half a beat to accommodate the new decoration. This operator increases the piece's note density and note onset density.

II.11. *Double Appoggiatura* operator

A more sophisticated version of the *appoggiatura* mutation, the *double appoggiatura* precedes the decorated note with both its adjacent notes, in random order. A simplified activity diagram describing its behavior is shown in Fig. 27. It first identifies the decorated note and its adjacent notes using MUSEC's *KB* module. Yet unlike the *appoggiatura* operator, *double appoggiatura* does not select one of the two adjacent pitches, but rather chooses an order (i.e., which note is played first) and a duration distribution (using eighth beat time units) for these two notes over the half-beat they are allocated, following which these notes are sequentially added and the decorated note is delayed by half a beat to fit the decoration. To avoid redundancy, the distribution in this case cannot include zero values, so that this operator, when applied, does not boil down to the *appoggiatura* operator described earlier. Following this constraint, three possible duration distribution are possible: (0.375, 0.125), (0.25, 0.25), and (0.125, 0.375), as shown in Fig. 27.

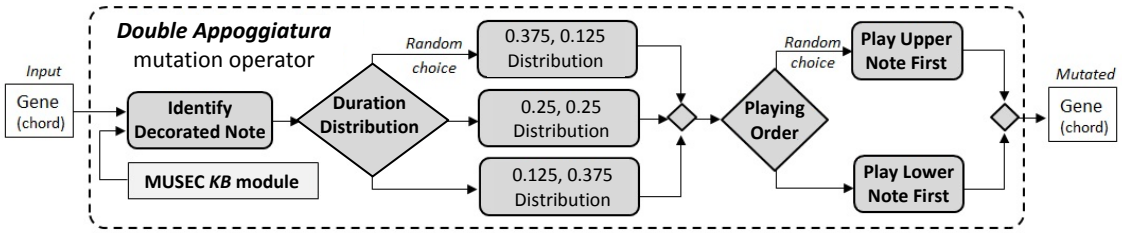


Fig. 27. Simplified activity diagram describing the *double appoggiatura* mutation operator

II.12. *Octava* operator

The *octava* operator affects the composition's average MIDI pitch by shifting a chord's notes' MIDI pitches up or down by an octave (i.e., it adds/subtracts 12 to the said notes' MIDI pitches). The choice of octave jump (up or down) is stochastically governed by the current average pitch of the chord such that chords with a lower average pitch are likelier to be shifted up by an octave, and chords with higher average pitch are likelier to be shifted down an octave.

II.13. *Tempo Steal* operator

The *tempo steal* operator, unlike all previous operators, affects two chords, rather than just one. In this mutation, two consecutive chords are selected such that one "steals" a certain duration in beats from the other. The steal value used in MUSEC is half a beat. This mutation would not take place should the chord that is "stolen" be less than a beat long. Essentially, this operation extends a chord by half a beat using the *extend* operator described previously, and shrinks the other by half a beat. *Shrinking* works using a similar logic to *extending*, where the notes at the beginning of the shrunken chord are shortened by half a beat. This mutation was introduced to break the static duration distribution among chords, and to make compositions more rhythmically diverse.

II.14. *Passing Notes* operator

This mutation also runs on two adjacent chords, by adding notes to the first chord based on the higher note in the following chord. A simplified activity diagram describing the *passing notes* operator is shown in Fig. 28. It checks the highest notes in both chords. It then checks if both chords are in the same key and whether both highest notes are less than an octave apart so as to ensure a reasonable number of notes is subsequently added. If these conditions are verified, MUSEC's *KB* module is called to identify all intermediary notes between the two higher notes based on the common key. Finally, these notes are added in sequence (over a total duration of half a beat) to the end of the first chord following a duration distribution. The latter is decided by randomly allocating duration "chunks" equal to the total duration divided by the number of passing notes.

As with the *progressive entrance* and *nota cambiata* duration distributions, zero values are possible and notes shorter than an eighth-beat are discarded, which adds variability to this operator's results. In total, $\binom{2N-1}{N-1}$ distributions are possible for the addition of n passing notes, resulting in an increase in piece note density and note onset density.

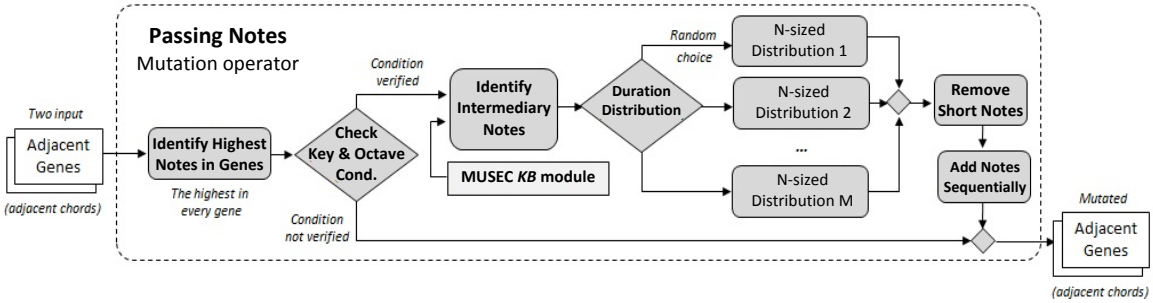


Fig. 28. Simplified activity diagram describing the *passing notes* mutation operator

II.14. *Anticipation* operator

Also applied to two consecutive chords within the piece, the *anticipation* operator checks the highest notes of the second chord, and then inserts it in the final half-beat of the first chord. This operator emulates the music-theoretical concept of "anticipation" and increases both note density and note onset density.

II.15. *Tempo Change* operator

The *tempo change* operator targets the piece's overall tempo. It changes tempo value in increments or decrements of 4 BPM (beats per minute). The increase/decrease decision is stochastically governed by in the individual's current tempo, where pieces that are slower are likelier to speed up following this mutation, and vice-versa.

II.16. *Intensity Change* operator

The *intensity change* operator changes the piece's current intensity value (i.e., MIDI Velocity) in steps of 20, such that pieces that are too quiet are likelier to become louder and vice versa, thereby producing an effect similar to a composer's dynamics. This is the only operator affecting a piece (individual)'s average velocity.

II.17. *Modulation/Demodulation* operator

This is the most music-theoretic intensive mutation implemented in MUSEC, changing a piece's current key to a new key. In theory, a piece can change to any key of the 23 possible other keys. For the sake of simplicity, MUSEC was artificially restricted to modulate only to its neighbor keys, i.e., keys with which it shares an edge (direct connection) in the *circle of fifths* (c.f. Section 4.3.2). Note that we adopt a transient approach to modulations in MUSEC: using a common chord between the source and destination key to modulate (other modulation approaches, such as abrupt modulation, are not yet included in the current version of the system).

A simplified activity diagram describing the operator's behavior is shown in Fig. 29. *Modulation* checks the last chord in the individual and identifies potential destination keys using MUSEC's *KB* module. In the event that many alternatives are possible, it randomly selects a destination key. Alternatively, when no destination keys are compatible, the mutation is aborted. To announce the modulation, the operator also appends two chords to the modulated individual: i) the new key's dominant chord, and ii) the root chord, thereby producing a perfect cadence. Finally, the piece's current key is changed to the new key.

Demodulation occurs when the individual's main key is different from the current key, and follows an analogous procedure to that of *modulation*.

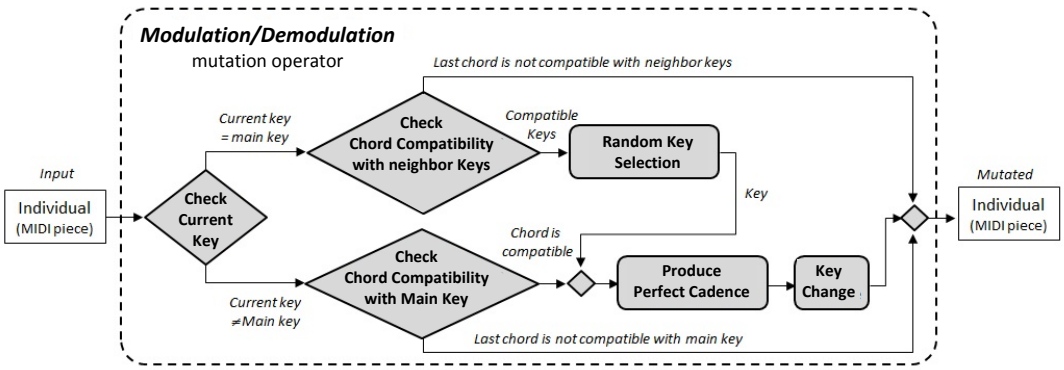


Fig. 29. Simplified activity diagram describing the *modulation/demodulation* mutation operator