MetaCompose: A Compositional Evolutionary Music Composer

Marco Scirea[1], Julian Togelius[2], Peter Eklund[1], and Sebastian Risi[1]

[1] IT University of Copenhagen, Denmark
msci@itu.dk
WWW homepage: http://www.itu.dk/msci
[2] New York University, USA

Abstract. This paper describes a compositional, extensible framework for music composition and a user study to systematically evaluate its core components. These components include a graph traversal-based chord sequence generator, a search-based melody generator and a pattern-based accompaniment generator. An important contribution of this paper is the melody generator which uses a novel evolutionary technique combining FI-2POP and multi-objective optimization. A participant-based evaluation overwhelmingly confirms that all current components of the framework combine effectively to create harmonious, pleasant and interesting compositions.

Keywords: Evolutionary computing, genetic algorithm, music generator

1 Introduction

Computer music generation is an active research field encompassing a wide range of approaches [1]. There are many reasons for wanting to build a computer system that can competently generate music. One is that music has the power to evoke moods and emotions – even music generated algorithmically [2]. In some cases the main purpose of a music generation algorithm is to evoke a particular mood. This is true for music generators that form part of highly interactive systems, such as those supporting computer games. In such systems a common goal of music generation is to elicit a particular mood that dynamically suits the current state of the game play. Music generation for computer games can be seen in the content of an experience-driven procedural content generation framework (*EDPCG*) [3], where the game adaptation mechanism generates music with a particular mood or affect expression in response to player actions.

While the medium-term goal of our system (described in Section 3) focuses on this kind of affect expression, this paper describes work on the core aspects of music generation, without expressly considering affective impact.

In games, unlike in traditional sequential media such as novels or movies, events unfold in response to player input. Therefore, a music composer for an interactive environment needs to create music that is dynamic while also being

non-repetitive. This applies to a wide range of games but not all of them; for example rhythm games make use of semi-static music around which the gameplay is constructed. The central research question in this paper is how to create music that is dynamic and responsive in real-time, maintaining fundamental musical characteristics such as harmony, rhythm, etc. We describe a component-based framework for (i) the generation of a musical abstraction and (ii) real-time music creation through improvisation. Apart from the description of the method the main research question of this paper is **validating the music generation algorithm: do all the components of the system add to the music generated?** To that end we present and discuss the results of a participant-based evaluation study.

2 Background

2.1 Procedurally generated music

Procedural generation of music is a field that has received much attention over the last decade. The approaches taken are diverse, they range from creating simple sound effects, to avoiding repetition when playing human-authored music, to creating more complex harmonic and melodic structures [4]. A variety of different approaches to procedural music generation have been developed, which can be divided into: *transformational* and *generative* algorithms [5]. MetaCompose falls in the latter category as it creates music without having any predefined snippets to modify or recombine.

Similar work to ours can be found in the system described by Robertson [6], which focuses on expressing fear. There are some parallels with this work, as the representation of the musical data through an abstraction (in their case the CHARM representation [7]), yet we claim our system has a higher affective expressiveness since it aims to express multiple moods via music. There are many examples of using evolutionary approaches to generating music, two example are the work of Loughran *et al* [8] and Dahlstedt's evolution of piano pieces [9], many more can be found in the *Evolutionary Computer Music* book [10].

Other examples of real-time music generation can be found in some patents: a system that allows the user to play a solo over some generative music [11] and a system that can create complete concerts in real time [12]. An interesting parallel between the second system and ours is the incorporation of measures of "distance" between music snippets to reduce repetition. Still, both these approaches present no explicit affective expression techniques.

The work of Livingstone [13] in trying to define a dynamic music environment where music tracks adjust in real-time to the emotions of the game character (or game state). While this work is interesting, it is still limited (in our opinion), by the usage of predefined music tracks for affective expression. Finally *Mezzo*[14], a system designed by Daniel Brown that composes neo-Romantic game soundtracks in real time, creates music that adapts to emotional states of the character, mainly through the manipulation of *leitmotifs*.

2.2 Multi-Objective Optimization

Multi-Objective Optimization (MOO) is defined as the process of simultaneously optimizing multiple objective functions. In most multi-objective optimization problems, there is no single solution that simultaneously optimizes every objective. In this case, the objective functions are said to be partially conflicting, and there exists a (possibly infinite) number of Pareto optimal solutions. A solution is called nondominated, Pareto optimal, Pareto efficient or noninferior, if none of the objective functions can be improved in value without degrading some of the other objective values. Therefore, a practical approach to multi-objective optimization is to investigate a set of solutions (the best-known Pareto set) that represent the Pareto optimal set as much as possible [15]. Many Multi-Objective Optimization approaches using Genetic Algorithms (GAs) have been developed. The literature on the topic is vast; Coello lists more than 2000 references on this topic in his website[3].

Our approach builds on the successful and popular NSGA-II algorithm [16]. The objective of the NSGA-II algorithm is to improve the adaptive fit of a population of candidate solutions to a Pareto front constrained by a set of objective functions. The population is sorted into a hierarchy of sub-populations based on the ordering of Pareto dominance. Similarity between members of each subgroup is evaluated on the Pareto front, and the resulting groups and similarity measures are used to promote a diverse front of non-dominated solutions.

2.3 Feasible/Infeasible 2-Population Genetic Algorithm

Many search/optimization problems have not only one or several numerical objectives, but also a number of constraints – binary conditions that need to be satisfied for a solution to be valid. The approach we adopted for melody generation contains such strong rules, that are described in detail in Section 5.2. A number of constraint handling techniques have been developed to deal with such cases within evolutionary algorithms. FI-2POP [17] is a constrained evolutionary algorithm that keeps two populations evolving in parallel, where feasible solutions are selected and bred to improve their objective function values while infeasible solutions are selected and bred to reduce their constraint violations. Each generation, individuals are tested to see if they violate the constraints; if so they are moved to the 'Infeasible' population, otherwise they are moved to the 'Feasible' one. An interesting feature of this algorithm is that the infeasible population influences, and sometimes dominates, the genetic material of the optimal solution. Since the infeasible population is not evaluated by the objective function it cannot get stuck in a sub-optimal solution, but it is free to explore boundary regions, where the optimum is most likely to be found.

[3] http://www.cs.cinvestav.mx/~constraint/papers/

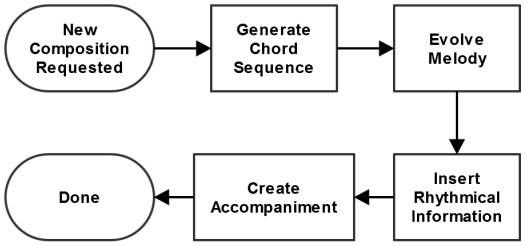

Fig. 1. Steps for generating a *composition*.

3 MetaCompose

The presented system is composed of three main components: (i) composition generator, (ii) real-time affective music composer and (iii) an archive of previous compositions. The modular nature of the system allows components to be easily swapped for other components or augmented with further components. The archive (iii) maintains a database of all the previous compositions connected to the respective levels/scenes of the game. The archive allows persistence of compositions for later reuse, but also allows us to compute a measure of novelty for future compositions compared with what has already been heard. This database could also be extended to connect compositions to specific characters, events, game levels, etc. The real-time affective music composer is the component that transforms a composition in the final score according to a specific mood or affective state. The system is designed to be able to react to game events, such events depending on the effect desired, examples of responses to such events include a simple change in the affective state, a variation of the current composition or an entirely new composition.

4 Non-dominated Sorting Feasible-Infeasible 2 Populations

Usually, when dealing with constrained optimization problems, the solution adopted is the introduction of penalty functions to act as constraints. This approach favors the feasible solutions to the infeasible ones, potentially removing infeasible individuals that might lead to an optimal solution, and getting the solutions stuck at a local optimum. There have been many examples of a constrained multi-objective optimization algorithms [18] [19] [20] [21].

Presented here is a combination of FI-2POP and NSGA-II dubbed Non-dominated Sorting Feasible-Infeasible 2 Populations (NSFI-2POP), uniting the benefits of keeping an infeasible population, which is free to explore the solution space without being dominated by the objective fitness function(s), and finding the Pareto optimal solution for multiple objectives. The algorithm takes the

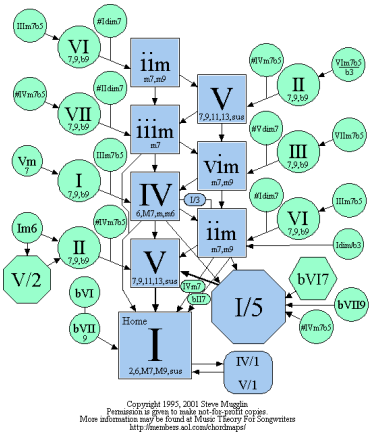

Fig. 2. Common chord progression map for major scales, created by Steve Mugglin [22].

structure of FI-2POP, but the objective function of the feasible function is substituted with the NSGA-II algorithm. In section 5.2 is described an application of this approach to the evolution of melodies.

5 Composition Generation

Composition in this paper is a *chord sequence*, a *melody* and an *accompaniment*. It is worth noting that the *accompaniment* is only an abstraction and not a complete score of a possible accompaniment, which is described in detail in Section 5.3 below. The main reason for the deconstruction of compositions is that we want a general structure (an abstraction) that makes music recognizable and gives it identity. Generating abstractions, which themselves lack some information that one would include in a classically composed piece of music, e.g. tempo, dynamics, etc, allows the system to modify the music played in real-time depending on the affective state the game-play wishes to convey. The generation of compositions is a process with multiple steps: (i) create a chord sequence, (ii) evolve a melody fitting this chord sequence, and (iii) create an accompaniment for the melody/chord sequence combination (see Fig. 1).

5.1 Chord Sequence Generation

The method for generating a chord sequence works as follows: we use a directed graph of common chord sequences and preform random walks on this graph (see Fig. 2) starting from a given chord. As can be seen from the Fig. 2, the graph does not use a specific key, but rather 'degrees': in music theory, a degree (or scale degree) is the name given to a particular note of a scale to specify its position relative to the 'tonic' (the main note of the scale). The tonic is considered to be the first degree of the scale, from which each octave is assumed to begin.

The degrees in Fig. 2 are expressed in Roman numerals and, when talking about chords, the numeral in upper-case letters symbolizes a major chord, while lower-case letters (usually followed by an m) express a minor chord. Other possible variations on the chord are generally expressed with numbers and other symbols, which we don't list for the sake of brevity. So, if we consider the D major scale, the $Dmajor$ chord would correspond to a I degree, while a $iiim$ degree would be a $F\sharp minor$. Various parameters of this sequence can be specified, such as sequence length, first element, last element, chord to which the last element can resolve properly (e.g., if we specify that we want the last chord to be able to resolve in the V degree, the last element might be a IV or a iim degree).

An interesting aspect of this graph is that it also shows common resolutions to chords outside of the current key, which provide a simple way of dealing with key changes. Each chord can be interpreted as a different degree depending on which key is considered, so if we want a key change we can simply: (i) find out which degree the last chord in the sequence will be in the new key and (ii) follow the graph to return to the new key. This produces harmonious key changes which do not sound abrupt.

5.2 Melody Generation

Melodies are generated with an evolutionary approach. We define a number of features to both include and avoid in melodies based on classical music composition guidelines and personal musical practice. These features are divided into constraints and objective functions. Accordingly, we use a Feasible/Infeasible two-population method (*FI-2POP* [17]) with multi-objective optimization [23] for the Feasible population. Given a chord sequence, a variable number of notes is generated for each chord, which will evolve without duration information. Once the sequence of notes is created, we generate the duration of the notes randomly.

Genome Representation The evolutionary genome consists of a number of values (the number of notes to be generated) that can express the notes belonging to two octaves of a generic key (i.e. 0-13). Here, we do not introduce notes that do not belong to the key, effectively making the context in which the melodies are generated strictly diatonic. Alterations will appear in later stages, in the real-time affective music composer module, when introducing variations of the composition to express affective states or chord variations.

Constraints We have three constraints: a melody should: (i) *not have leaps between notes bigger than a fifth*, (ii) *contain at least a minimum amount of leaps of a second* (50% in the current implementation) and (iii) *each note pitch should be different than the preceding one*.

$$\text{Feasibleness} = -\sum_{i=0}^{n-1}(\text{Second}(i, i+1) + \text{BigLeap}(i, i+1) + \text{Repeat}(i, i+1)) \quad \text{where } n \text{ is the genome length} \quad (1)$$

The three functions comprising equation 1 are all boolean functions that return either 1 or 0 depending if the two notes at the specified indexes of the genome satisfy the constraint or not. As can be seen, this function returns a number that ranges from (potentially) $-\infty$ to 0, where reaching the 0 score determines that the individual satisfies all the constraints and, consequently, can be moved from the unfeasible population to the feasible one.

On the constraints in eqn 1, leaps larger than a fifth do appear in music but they are avoided here, as experience suggest they are too hard on the ear of the listener. Namely, if the listener is not properly prepared, leaps larger than a fifth can easily break the flow of the melody. We also specify a minimum number of intervals of a second (the smallest interval possible considering a diatonic context such as this one, see the *Genome Representation* section) because if the melody has too many larger leaps it feels more unstructured, and not something that we would normally hear or expect a voice to sing. Finally the constraint on repetition of notes is justified by the fact that these will be introduced in the real-time interpretation of the abstraction.

Fitness Functions Three objectives build to compose the fitness functions: the melody should (i) *approach and follow big leaps (larger than a second) in a counter step-wise motion (explained below)* (eqn 2), (ii) where the melody presents big leaps the *leap notes should belong to the underlying chord* (eqn 3) and finally (iii) *the first note played on a chord should be part of the chord* (eqn 4).

$$\text{CounterStep} = \frac{\sum_{i=0}^{n-1} [\text{IsLeap}(i, i+1)(\text{PreCounterStep}(i, i+1) + \text{PostCounterStep}(i, i+1))]}{leapsN} \tag{2}$$

$$\text{ChordOnLeap} = \frac{\sum_{i=0}^{n-1} [\text{IsLeap}(i, i+1)(\text{BelongsToChord}(i) + \text{BelongsToChord}(i+1))]}{leapsN} \tag{3}$$

$$\text{FirstNoteOnChord} = \frac{\sum_{i=0}^{n} (\text{IsFirstNote}(i) \times \text{BelongsToChord}(i))}{chordsN} \tag{4}$$

First, we remind the reader that the definition of an interval in music theory is the **difference between two pitches**. In Western music, intervals are mostly differences between notes belonging to the diatonic scale (for example, considering a *Cmajor* key, the interval between C and D is a *second*, the interval between C and E is a *third* and so on).

To clarify what *counter step-wise motion* means: if we examine a leap of a fifth from C to G as in Fig. 3 (assuming we are in a *Cmajor* key), this is a upward movement from a lower note to a higher one, a counter step-wise approach would mean that the C would be preceded by a higher note (creating a downward movement) with an interval of a second, so a D. Likewise following

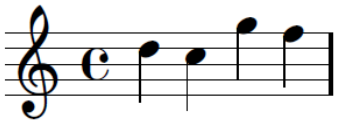

Fig. 3. Example of counter step-wise approach and departure from a leap (C-G).

the leap in a counter step-wise motion would mean that we need to create a downward movement of a second after the G, so we need an F to follow. The reason we introduce this objective is that this simple technique makes leaps much easier on the listener's ear, otherwise they often sound too abrupt by suddenly changing the range of notes the melody is playing. The `PreCounterStep` and `PostCounterStep` functions are boolean functions that respectively check if the note preceding and following the leap approaches or departs with a contrary interval of a second.

The reason for having the leap notes – the two notes that form a leap bigger than a second – be part of the underlying chord is that leaps are intrinsically more interesting than a step-wise motion, this means that the listener unconsciously considers them more meaningful and pays more attention to them. If these leaps contain notes that have nothing to do with the underlying chord, even if they do not present real dissonances, they will be perceived as dissonant because they create unexpected intervals with the chord notes. Trying to include them as part of the chord gives a better sense of coherence that the listener will consider as pleasant. The last objective simply underlines the importance of the first note after a chord change, by playing a note that is part of the chord we reinforce the change and make it sound less discordant.

Note that these objectives, by the nature of multi-objective optimization, will generally not all be completely satisfied. This is fine, because satisfying all objectives might make the generated music sound too mechanical, while these are "soft" rules that we want to enforce only to a certain point (contrary to the constraints of the infeasible population, which always need to be satisfied).

5.3 Accompaniment Generation

Accompaniment is included in the composition because, not only chords and melody give identity to music, but also rhythm. The accompaniment is divided into two parts: a **basic rhythm** (a collection of note duration) and a **basic note progression** (an *arpeggio*). We can progress from the accompaniment representation to a score of the accompaniment by creating notes with duration from the basic rhythm and pitches from the progressions (offset on the current underlying chord).

Accompaniments are generated through a stochastic process involving combinations and modifications (inversions, mutations, etc) of some elements taken from a small archive of basic rhythms. Specifically we have four basic rhythm patterns and two basic arpeggios (see Figs. 4 & 5).

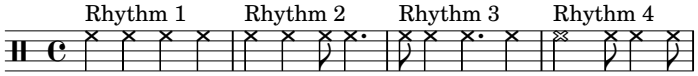

Fig. 4. Basic rhythms.

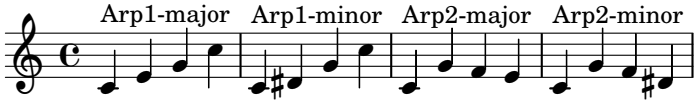

Fig. 5. Basic arpeggios. These are represented as if they were played under a C major or C minor chord, but are transposed depending on what chord they appear underneath. Also the rhythmic notation of the arpeggio is dependent on the rhythmic structure.

The algorithm performs the following steps:

1. choose a basic rhythm and basic arpeggio;
2. shuffle the elements of the basic rhythm;
3. shuffle the elements of the arpeggio;
4. increase the basic rhythm: with a probability of 0.5, either the biggest (longest duration) or a random element is split in two elements of half the size of the original duration. This function is called recursively with linearly decreasing probability;
5. increase the arpeggio to match the new size of the basic rhythm: this is done by introducing at a random index of the arpeggio a new random pitch that already belongs to the arpeggio.

The rhythm presented in the final music will be modified by the real-time affective music composer for variety or for affect expression, while still maintaining a rhythmic and harmonic identity that will be characteristic of the composition.

6 Experiment design

We performed an extensive quantitative study to validate our music generation approach. The main objective is to investigate the contribution of each component of the framework to the quality of the music created. To do this, we systematically switched off components of our generator and replaced them with random generation. From these random "broken" compositions and the complete algorithm we created various pair-wise samples to test against each other. (This method was inspired by the "ablation studies" performed by e.g. Stanley [24].) As the quality of music is a subjective matter, we conducted a survey where participants are asked to prefer one of two pieces of music presented to them (one generated by the complete algorithm and one from a "broken" generator with one component disabled) and evaluate them according to four criteria: *pleasantness*, *randomness*, *harmoniousness* and *interestingness*. Using these four criteria presents a good overview of the preference expressed by the

participant. Note that no definition of these terms is offered in the survey, so we have no guarantee they might not be interpreted differently by individual test subjects.

Pleasantness measures how pleasing to the ear the piece is, but this alone is not sufficient to describe the quality of the music produced. There are countless pieces of music that do not sound pleasant, but may nonetheless be considered by the listener as "good" music. In fact, in music, it is of course common to introduce uncommon (and even discordant) chord sequences or intervals to express different things like affect, narrative information and other effects. Also note that some alterations or passages can be specific of a music style. Moreover, discordant intervals is more and more accepted to the ear the more repeated they are (see dodecaphonic music, for example).

Interestingness is introduced to overcome the just described limitations of the pleasantness criteria: in this way we are able to test if one of our "broken" versions might introduce something that would result in something considered interesting to the listener, even when the composition is not as pleasant or harmonic. Note that this is a very subjective measure, as most people would find different things interesting.

On the other hand, *harmoniousness* might be confused with pleasantness, but we hope that it will be seen as a more objective measure: less of a personal preference and more of an ability to recognize the presence of dissonances and harmonic passages.

Finally, *randomness* gathers a measure of how structured the music appears to the listener. It is not only a measure of dissonance (or voices being off-key), but also of how much the music seems to have a cohesive quality and internal structure. Examples of internal structure are: (i) voices working together well (ii) coherent rhythmic structure (iii) the chord sequence presents tension building and resolution.

An online survey was developed with HTML and PHP, using a MySQL database to hold the data collected. Participants were presented with pairswise music clips and asked to evaluate them using the previously described four criteria. Each criteria has a multiple choices question structured as:

> **Which piece do you find more pleasing?** *"Clip A"/"Clip B"/"Neither"/ "Both Equally"*

Where the last word (e.g. "pleasing") is dependent on the criteria. We also include the more neutral answers "Neither" and "Both Equally" to avoid randomness in the data from participants who cannot decide which clip satisfies the evaluation criteria better or worse. Other benefits of doing this are: avoiding the participant getting frustrated and giving us some possibly interesting information on cases where the pieces are considered equally good/bad.

Note that for the first five questions in the survey instrument the pair-wise clips always included one clip from the complete generator. After five trials, the clip pairs are picked at random between all the groups. In this way, we hoped to collect enough data to be able to make some observations between the four

"broken" generators. The motivation behind this design choice is that our main question is evaluating the complete generator against all possible alternatives, so attention to the complete architectural music generator has priority. This also has a practical justification in the fact that, with the number of groups we have (five), testing all possible combinations and gather enough data would be near impossible. The survey has no pre-defined end: the user is able to continue answering until he/she wants, and can close the online survey at any time or navigate away from it without data loss. In the preamble, we encouraged participants to do at least five comparisons.

6.1 Music clip generation

The five groups were examined (as dictated by the architecture of our generator):

A. Complete generator: normal composition generator as described in Section 5;
B. Random chord sequence: the chord sequence module is removed and replaced with a random selection of chords;
C. Random unconstrained melody: the melody generation evolutionary algorithm is replaced with a random selection between all possible notes in the melody range (two octaves);
D. Random constrained melody: the melody generation evolutionary algorithm is replaced with a random selection between all possible notes belonging to the key of the piece in the melody range (two octaves). We decided this was necessary as (by design) our melody evolution approach is restricted to a diatonic context;
E. Random accompaniment: the accompaniment generation is replaced by a random accompaniment abstraction (we remind the reader that an accompaniment abstraction is defined by a basic rhythm and note sequence).

For each of these 5 groups, 10 pieces of music are created. For the sake of this experiment the affect expression has been kept to a neutral state for all the groups and we used the same algorithms to improvise on the composition abstraction. There is therefore no test of the music generators affect generation but rather of the music quality form the complete architecture compared to the architectural alternatives. The clips for the various groups can be accessed at http://msci.itu.dk/evaluationClips/

7 Results and analysis

The data collected amounts to a total of 1,291 answers for each of the four evaluation criteria from 298 participants. Of the survey trials generated, 1,248 contained a clip generated with our complete music generator algorithm (A). Table 1 shows how many responses were obtained for each criteria and how many neutral answers were collected.

Table 1. Amounts of correct, incorrect and neutral answers to our criteria for the complete generator against all the "broken" generators (B-E) combined. Keep in mind that in the case of the random criteria the listener is asked to select the clip that he/she feels the most random, so it is entirely expected that a low number of participants choose the random clip (E) against the complete generator (A).

Choice	**Pleasing**	**Random**	**Harmonious**	**Interesting**
Choose the complete generator (A)	654	197	671	482
Choose a "broken" generator (B-E)	240	633	199	327
A neutral answer	197	261	221	282
Total non-neutral answers	894	830	870	809
Binomial test p-value	7.44E-21	2.75E-77	2.05E-29	7.81E-02

For now we only consider definitive answers (the participant chooses one of the music clips presented), we will look at the impact of the neutral answers at the end of this section. Under this constraint, the data becomes boolean: the answers are either *"user chooses the clip from the complete generator (A)"* or *"user chose the clip from the broken generator (B-E)"*. To analyze this data we use a two-tailed binomial test, which is an exact test of the statistical significance of deviations from a theoretically expected random distribution of observations into two categories. The null hypothesis is that both categories are equally likely to occur and, as we have only two possible outcomes, that probability is 0.5.

7.1 Complete Generator against all other groups

Firstly, let us consider the combined results of all the "broken" groups (B-D) against the complete generator (A): as can be seen from Tab. 1, we have statistically highly significant differences for the *pleasing*, *random* and *harmonious* categories, while we have a p-value of 0.078 for the *interesting* category. This means that we can refuse the null hypothesis and infer a difference in distribution between choosing the music generated by the complete algorithm (A) and the "broken" ones (B-E).

We can affirm that our generator (A) ranked better than all the other versions (B-E) for three of our four criteria, with the exception of *interestingness*, where there is no statistically significant difference. Interestingness is clearly a very subjective measure, and this may explain the result. Moreover, examining the ratio of neutral answers obtained for this criteria, it can be inferred that it is almost 26%, so a much higher neutral response that for the other criteria. This suggests that in a higher number of cases participants could not say which composition they found more interesting. A possible explanation is that, as the affect expression (which also includes musical features such as tempo and intensity) is held in a neutral state, equal for all pieces, after hearing a number of clips the listener does not find much to surprise him/her. Also the duration of the generated pieces (30 s) might not allow sufficient time to determine interestingness.

Table 2. Answers and results of the binomial test for pairs comprised of the full generator and the one with random chord sequences.

A versus B	pleasing	random	harmonious	interesting
successes	121	71	112	98
failures	93	117	84	98
totals	214	188	196	196
Binomial test p-value	3.23E-02	4.90E-04	2.68E-02	5.28E-01

Table 3. Answers and results of the binomial test for pairs comprised of the full generator (A) and the one with unconstrained random melody (C).

Table 4. caption

A versus C	pleasing	random	harmonious	interesting
successes	221	21	236	144
failures	26	221	19	72
totals	247	242	255	216
Binomial test p-value	5.15E-40	1.44E-43	4.11E-49	5.46E-07

7.2 Complete Generator against random chord sequence generation

If we only consider the pairs that included the complete generator (A) and the one with random chord sequences (B) (Tab. 2) we, again, obtain statistically significant differences in the distribution of the answers for the *pleasing*, *random* and *harmonious* criteria. In this case we have a very high p-value for *interestingness* (more than 0.5), in fact we have the same amount of preference for the complete generator (A) and the "broken" one (B). We can explain this by considering that the disruptive element introduced by this modification of the algorithm is mitigated by the fact that the rest of the system tries to create as pleasing music as it can based on the chord sequence given. So, for most of the time, the music will not have notes that sound out of key or that do not fit well with the chord sequence. Still, we can observe how the listener is capable of identifying that, while the piece does not sound discordant or dissonant, it lacks the structure of tension-building and tension-releasing. This explains how the complete generator (A) is preferred for all other criteria. It is interesting to note how the act itself of presenting the listener with uncommon chord sequences does create an increase in the interestingness of the music.

7.3 Complete Generator against unconstrained melody generation

When we consider the unconstrained melody group we have statistically significant differences for all criteria, with some extremely strong significance (Tab. 3). These results are as we expected, as the melody plays random notes that conflict with both the chord sequence and the accompaniment.

Table 5. Answers and results of the binomial test for pairs comprised of the full generator (A) and the one with constrained random melody (D).

A versus D	pleasing	random	harmonious	interesting
successes	125	81	120	108
failures	100	109	85	94
totals	225	190	205	202
Binomial test p-value	5.47E-02	2.49E-02	8.68E-03	1.80E-01

Table 6. Answers and results of the binomial test for pairs comprised of the full generator (A) and the one with random accompaniment (E).

A versus E	pleasing	random	harmonious	interesting
successes	188	25	203	132
failures	21	186	12	63
totals	209	211	215	195
Binomial test p-value	5.00E-35	6.58E-32	3.01E-46	4.35E-07

7.4 Complete Generator against constrained melody generation

The results given by the constrained random melody generation (D) are more interesting (Tab. 5). First, we notice no statistically significant values for the *pleasing* and *interesting* criteria. This is explained by the fact that the melody never goes off key, so it never presents off-key notes and never sounds abruptly "wrong" to the listener's ear. Yet, the *random* and *harmonious* criteria are statistically significant. Remembering how we described these criteria we notice that the more more objective criteria (*random* and *harmonious*) are those that demonstrate a difference in distribution. We believe this reinforces how, although compositions made in this group never achieve a bad result, the listener is still able to identify the lack of structure (randomness) and lack of consideration of the underlying chords of the melody (harmoniousness). An example of the first case would be a melody that jumps a lot between very different registers; this would make the melody sound more random than the melodies we evolve using (A), which follow more closely the guidelines of a singing voice. Harmoniousness can be influenced by the fact that, over a chord (expressed by the accompaniment), the melody can play notes that create intervals that 'muddle' the clarity of the chord to the listener's ear.

7.5 Complete Generator against random accompaniment generation

Finally, for the last group, the random accompaniment generation (E), gives us very clear statistically significant results on all criteria (Table 6). A lot of the harmony expression depends on the accompaniment generation, and when this is randomized it is no wonder that the piece sounds confusing and discordant. This is reflected in the trial data.

8 Conclusion and future work

This paper describes a new component-based system for music generation based on creating an abstraction for musical structure that supports real-time improvisation. We focus on the method of creating the abstractions ("compositions"), that consists of the sequential generation of (i) chord sequences, (ii) melody and (iii) an accompaniment abstraction. Our novel approach is described to evolve melody in detail: non-dominated sorting with two feasible-infeasible populations genetic algorithm (NSFI-2POP).

Returning to the main question of the paper, (*do all parts of the music generation system add to the music produced?*), we have described an evaluation in which we created music with our generator substituting various components with randomized generators. In particular we observed four broken groups: *random chord sequences*, *random melody constrained* (to the key of the piece), *random melody unconstrained* and *random accompaniment*. An evaluation of the music clips generated by these "broken" versions was compared to music clips created by the complete algorithm according to four criteria: *pleasantness*, *randomness*, *harmoniousness* and *interestingness*.

Analysis of the data supports the assertion that participants prefer the complete system in three of the four criteria: (*pleasantness*, *randomness* and *harmoniousness*) to the alternatives offered. The results for the *interestingness* criteria are however not definitive, but suggest that some parts of our generator have a higher impact in this criteria. It is also noteworthy that there is no statistical significant difference between preferences between *constrained melody group* (C) and the *complete generator* (A) for the *pleasantness* criteria.

Future work will focus on developing and evaluating the affect-expression capabilities of the system, we will probably follow the methodology described by Scirea *et al.* for characterizing control parameters through crowd-sourcing [25]. In summary, we show (i) how each part of our music generation method assists creating music that the listener finds more pleasant and structured and (ii) presented a novel GA method for constrained multi-objective optimization.

References

1. Papadopoulos, G., Wiggins, G.: Ai methods for algorithmic composition: A survey, a critical view and future prospects. In: AISB Symposium on Musical Creativity, Edinburgh, UK (1999) 110–117
2. Konečni, V.J.: Does music induce emotion? a theoretical and methodological analysis. Psychology of Aesthetics, Creativity, and the Arts **2**(2) (2008) 115
3. Yannakakis, G.N., Togelius, J.: Experience-driven procedural content generation. IEEE Transactions on Affective Computing **2**(3) (2011) 147–161
4. Miranda, E.R.: Readings in music and artificial intelligence. Volume 20. Routledge (2013)
5. Wooller, R., Brown, A.R., Miranda, E., Diederich, J., Berry, R.: A framework for comparison of process in algorithmic music systems. In: Generative Arts Practice 2005 — A Creativity & Cognition Symposium. (2005)

6. Robertson, J., de Quincey, A., Stapleford, T., Wiggins, G.: Real-time music generation for a virtual environment. In: Proceedings of ECAI-98 Workshop on AI/Alife and Entertainment, Citeseer (1998)
7. Smaill, A., Wiggins, G., Harris, M.: Hierarchical music representation for composition and analysis. Computers and the Humanities **27**(1) (1993) 7–17
8. Loughran, R., McDermott, J., O'Neill, M.: Tonality driven piano compositions with grammatical evolution. In: IEEE Congress on Evolutionary Computation (CEC), IEEE (2015) 2168–2175
9. Dahlstedt, P.: Autonomous evolution of complete piano pieces and performances. In: Proceedings of Music AL Workshop, Citeseer (2007)
10. Miranda, E.R., Biles, A.: Evolutionary computer music. Springer (2007)
11. Rigopulos, A.P., Egozy, E.B.: Real-time music creation system (May 6 1997) US Patent 5,627,335.
12. Meier, S.K., Briggs, J.L.: System for real-time music composition and synthesis (March 5 1996) US Patent 5,496,962.
13. Livingstone, S.R., Brown, A.R.: Dynamic response: Real-time adaptation for music emotion. In: Proceedings of the 2nd Australasian Conference on Interactive Entertainment. (2005) 105–111
14. Brown, D.: Mezzo: An adaptive, real-time composition program for game soundtracks. In: Proceedings of the AIIDE 2012 Workshop on Musical Metacreation. (2012) 68–72
15. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation **8**(2) (2000) 173–195
16. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. Evolutionary Computation, IEEE Transactions on **6**(2) (2002) 182–197
17. Kimbrough, S.O., Koehler, G.J., Lu, M., Wood, D.H.: On a feasible–infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. Eur. J. Operational Research **190**(2) (2008) 310–327
18. Deb, K., Pratap, A., Meyarivan, T.: Constrained test problems for multi-objective evolutionary optimization. In: Evolutionary Multi-Criterion Optimization, Springer (2001) 284–298
19. Chafekar, D., Xuan, J., Rasheed, K.: Constrained multi-objective optimization using steady state genetic algorithms. In: Genetic and Evolutionary ComputationGECCO 2003, Springer (2003) 813–824
20. Jimenez, F., Gómez-Skarmeta, A.F., Sánchez, G., Deb, K.: An evolutionary algorithm for constrained multi-objective optimization. In: Proceedings of the Congress on Evolutionary Computation, IEEE (2002) 1133–1138
21. Isaacs, A., Ray, T., Smith, W.: Blessings of maintaining infeasible solutions for constrained multi-objective optimization problems. In: IEEE Congress on Evolutionary Computation, IEEE (2008) 2780–2787
22. Mugglin, S.: Chord charts and maps. http://mugglinworks.com/chordmaps/chartmaps.htm Accessed: 2015-09-14.
23. Deb, K.: Multi-objective optimization using evolutionary algorithms. Volume 16. John Wiley & Sons (2001)
24. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2) (2002) 99–127
25. Scirea, M., Nelson, M.J., Togelius, J.: Moody music generator: Characterising control parameters using crowdsourcing. In: Evolutionary and Biologically Inspired Music, Sound, Art and Design. Springer (2015) 200–211