An evolutionary algorithm for harmonic music composition

José Pedro Aguerre, Rodrigo Bayá, Renzo Massobrio, Sergio Nesmachnow

Facultad de Ingeniería, Universidad de la República
Herrera y Reissig 565, Montevideo 11300, Uruguay
{jpaguerre,rbaya,renzom,sergion}@fing.edu.uy

Abstract

This article presents an evolutionary approach for harmonic music composition using information from a database of existing songs from a given musician. The fitness function of the proposed evolutionary algorithm uses score comparison to find harmonies which are similar—but no equal—to those songs in the database. A voice-based version of the composition problem is solved, where each voice contains a melody composed by notes and rests. The generic problem is reduced to compose the musical notes implied in each voice and to compare them to a set of entry sounds, stored in a reference database. Promising results are reported for five sample database instances.

1 Introduction

Music composition is the art whose main goal is the creation of musical pieces. In occidental music, it requires the study of several disciplines, such as harmony, counterpoint, orchestration and musical forms.

From time immemorial human being has long for the automatic execution and creation of music. Nowadays, thanks to computer science, those expectations have become reality [10]. Algorithmic composition is the technique of using formal procedures to create music [6]. Actually, the technique allows composing songs without human intervention, for example by applying stochastic methods and/or using music–dependent knowledge [14].

This article describes the application of a parallel evolutionary algorithm (EA) [1] for harmonic music composition. We consider the optimization problem defined by the evaluation of scores using the comparison to a set of sounds stored in a reference database. Previous works have been developed in the field of harmonic music composition and evolutionary computation for music generation [3]; the main contribution of the proposed parallel EA is related to the efficient and accurate random generation of musical pieces, which evolve using the score comparison as a measurement of its musical aptitude. Efficient and accurate results are reported for a set of databases built as example execution instances.

The article is structured as follows. The problem definition and model are presented in Section 2. A brief description of EAs as problem solvers is presented in Section 3. Section 4 reviews the related work on automatic music composition. After that, the proposed EA for harmonic music composition is described in Section 5. The experimental analysis is reported in Section 6, including a discussion in the experimental results when executing the algorithm using a set of sample databases. Finally, Section 7 presents the conclusions and formulates the main lines of future work.

2 The harmonic music composition problem

A musical score is basically constituted by rests and a set of notes (C, D, E, F, G, A, B, and their respective sharps) with their correspondent octave and duration (e.g., whole, half, quarter). Other concepts, such as compasses and key and time signatures, are not relevant to the problem faced in this article.

Harmonic music composition is the process of defining a set of notes representing melodies and harmonies. In this work, a melody is considered to be a music idea independent of its accompaniment, composed by a set of musical notes and their respective execution times (rhythm). Harmony is understood as a union of two or more simultaneous sounds that generate harmonic intervals. These concepts are outlined in Figure 1.

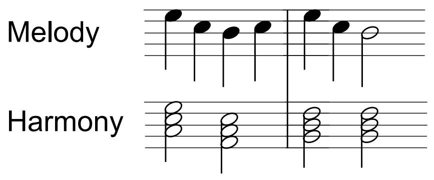


Figure 1: Example of melody and harmony in music

In this article, a score is represented by a set of melodies (also named voices). Therefore, the composition of the notes in each voice results in a harmonic piece of music. Every harmony generated by the voices must be analyzed to obtain good-sounding music. In the problem model, the number of individual voices is limited to 16. Odd duration times, like triplets, are not taken into account.

The mathematical formulation of the problem considers the following elements: *i*) a score s_G to be generated by the proposed optimization method; *ii*) a set of N scores $S = \{s_1, \dots s_N\}$, used as reference to generate and evaluate s_G; and *iii*) a function that measures similarity between s_G and a score in the reference set, $f : \{s_G\} \times S \rightarrow \mathbb{N}$. The goal of the problem is to find a score s_G that maximizes the objective function OF (Equation 1), where $\alpha, \beta \in [0, 1]$ and $\alpha + \beta = 1$.

$$OF = \max\left\{\alpha \times \frac{\sum_{s \in S} f(s_G, s)}{N} + \beta \times \max_{s \in S} f(s_G, s)\right\} \tag{1}$$

The objective function includes two components: the first one evaluates the average similarity between s_G and the scores in the reference set. The second component evaluates the similarity between s_G and the most similar score in the reference set. The linear combination between both components allows capturing the features of each score in the reference set, while not creating exact copies of any particular score. The weights α and β are used to fine-tune the generation process towards solutions that are more diverse or more similar to the scores in the reference set.

3 Evolutionary computation

This section introduces evolutionary computation and the parallel EA applied in this paper.

3.1 Evolutionary algorithms

EAs are stochastic optimization methods that simulate the evolution of species in nature. They have been successfully applied for solving problems underlying many complex real-life applications [2, 16]. An EA is an iterative technique that applies stochastic operators on a population of *individuals*. Each individual encodes a tentative solution of the problem. The goal of the EA is to improve the *fitness* of solutions, a measure related to the objective function. An evaluation function associates a fitness value to the solution encoded by every individual, indicating its suitability to the problem.

In an EA, the initial population is generated at random or by using a specific heuristic for the problem. Iteratively, the EA explores new solutions by applying probabilistic *variation operators*, including the *recombinations* of individuals or random changes (*mutations*) in their contents. The search is guided by a selection-of-the-best technique to tentative solutions of higher quality. The stopping criterion usually involves a fixed number of iterations (*generations*) or execution time, a quality threshold on the best fitness value, or the detection of a stagnation situation. Specific policies are used for selecting individuals to recombine and to determine which new individuals replace the older ones in each new generation. Finally, the EA returns the best solution found, regarding the fitness function values.

3.2 Parallel evolutionary algorithms

Parallel implementations became popular as an effort to improve the efficiency of EAs. By splitting the population or the fitness function evaluation into several processing elements, parallel EAs allow reaching high quality results in a reasonable execution time even for hard-to-solve optimization problems [1].

The parallel EA proposed for harmony composition is categorized within the *distributed subpopulations* model, which splits the population in several *demes*. Each deme runs a serial EA, whose individuals are able to recombine only with other individuals in the deme. An additional *migration* operator is defined: occasionally some selected individuals are exchanged among demes, introducing a new source of diversity in the EA. Figure 2 presents a graphic schema of a parallel EA with distributed subpopulations.

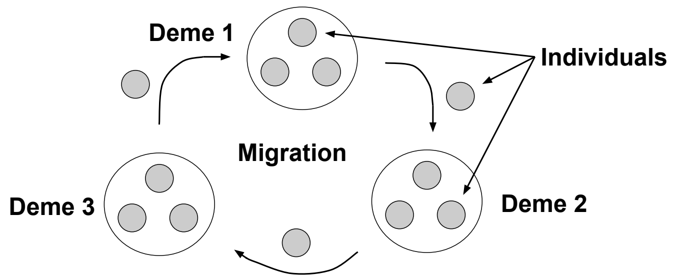


Figure 2: Parallel distributed subpopulations EA

4 Related work: automatic music composition

Studies on automatic music generation date back to the late 1950's. One of the first semi-automatic software methods for musical execution, transcription, and composition was proposed for the ILLIAC suite in 1959 [9]. Tools for automatic music composition use many different techniques, including random numbers, formal grammars, cellular automata, fractals, and evolutionary computation [4, 5, 11]. The first approaches used the human ear as the main detector of musical aptitude. New methods incorporating rules to evaluate musical quality were proposed as the computing power increased. A summary of the state-of-the-art methods on automatic music composition is presented by Miranda [15].

Related works on automatic music composition follow two different approaches: *i*) some works focus on generating compositions considering either *melodies* or *harmonies*; and *ii*) other works are based on *using entry sounds or scores* or *generating completely random music*.

Fujinaga and Vantome [8] applied granular synthesis to develop a music generator that does not depend on an initial score. In the proposal, the grains are short musical strips, which are organized in different patterns until a musically fit sequence is found. The resulting patterns are evaluated according to musical theory criteria. Lo and Lucas [12] presented an evolutionary approach that generates melodies based on N-grams, which allows finding the most commonly used sequences of notes in a set of scores from an author. N-grams were used for determining the similarity between a score and the musical pieces in the set. The article focused on the composition of melodies and on the development of a melody classifier. A training step is required to improve the evaluation function.

The evolutionary method by Wu et al. [19] generates melodies based on several musical theory rules. The EA copies the duration of every note from a given entry score. Some notes from the entry score are also directly copied to the solution. This approach is limited since it is only able to generate scores with the exact same notes duration than the entry score.

Marques et al. [13] developed an EA for evolving harmonic music from an entry musical score. The EA avoids using randomly generated initial individuals through a set of pre-designed rules and produces individuals which satisfy the specified restrictions. Melodic and musical theory concepts are used in the fitness calculation for searching attractive sounds.

The review of related work indicates that some works use entry scores to guide the automatic generation, but none attempts to exploit the similitude between compositions by the same author, in order to extract musical properties that can guide the generation. This article presents a parallel EA for harmonic music that generates completely aleatory scores as a basis but uses an entry score database for the computation of the musical fitness of the generated sounds. This novel approach is our main contribution to the art of automatic music composition, since the database can be composed of many scores by the same author, which will guide the generation into similar music.

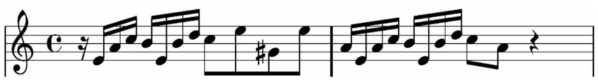
5 An EA for harmonic music composition

This section describes the implementation details of the proposed EA for harmonic music composition.

5.1 Solution encoding

A musical score is represented by a string, where each note in the score is separated by spaces. A note is represented by a set of characters: i) the first character is the note represented using letter notation; ii) the second character is an optional ”#” if the note is sharp; iii) the third character is the number that determines the octave (being C5 the central DO in a piano); and iv) the last character indicates the note duration: *w*–whole, *h*–half, *q*–quartet, *i*–eighth and *s*–sixteenth. A rest is represented by an *R*, followed by its duration. The character *V* is used to represent a new voice, where each note after a new voice belongs to it. This encoding does not take compasses or musical signatures into account. Due to these considerations, the length of each individual in the population is variable.

Figure 3 shows a score consisting of only one voice and the corresponding encoding. The example contains notes varying between two octaves and includes a sharp note and two rests.



"Rs E5s A5s C6s B5s E5s B5s D6s C6i E6i G#5i
E6i A5s E5s A5s C6s B5s E5s B5s D6s C6i A5i Rq"

Figure 3: Score extract taken from J. S. Bach piece Inventio 13 and its string representation.

5.2 Fitness function

The fitness function incorporates several techniques proposed in related works. Each individual in the population is evaluated by comparing the score it encodes against all the other scores in the reference database. Individuals which encode scores that are “musically similar” to the database entries are assigned a better fitness value. Additionally, several penalization functions are included to account for different musical properties in the encoded scores.

5.2.1 Score comparison criteria

Score comparison is implemented using seven functions. These functions are based on evaluating similarities and performing comparisons without voice division (taking harmony into account) and comparisons of separated voices (taking melody into account). Each function receives two scores as input and applies the corresponding comparison technique between them. These functions are described next:

1. *Number of voices*. Measures similarities on the number of voices: two scores with different number of voices are not close in the comparison.

2. *Number of silence notes*. Measures the amount of silences: this function returns the difference between the number of silences of the two scores

3. *String comparison*. Compares individuals using the regular string comparison, a useful function according to preliminary analysis of results.

4. *Harmonic intervals*. Computes the difference between harmonic intervals on the scores, as described in Algorithm 1. In the harmonic intervals calculation, each individual is divided into chords, separating each note into many sixteenth notes (as this is the minimal duration of a note when using the proposed encoding). Notes in every chord are ordered from low to high and the number of intervals for each possible interval length is counted. The intervals are computed between the lower note (known as the bass) and the rest of them. Figure 4 illustrates this concept.

Algorithm 1 Calculation of harmonic intervals

```
S ← minimal transformation of score
voices ← get voices from S
chords_list ← empty_list                         {iterate over notes}
for i ← 0 to score_length do
    chord ← empty_list                           {a chord is a list of notes}
    for all voice in voices do
        if (i < voice→length) then
            chord→add(voice→get(i))
        end if
    end for
    chord ← notes from low to high
    chords_list→add(chord)
end for
N ← 60                                           {number of representable notes}
M ← 16                                           {maximum number of voices}
Mat ← zeros($M \times N$)
for all chord in chords_list do
    for i ← 1 to chord→length do
        interval → (chord→get(0) - chord→get(i))  {subtracting two notes results in an interval}
        Mat(i, interval)++
    end for
end for
return Mat
```

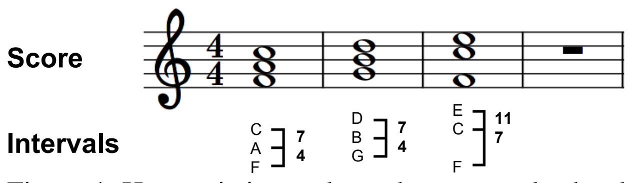


Figure 4: Harmonic intervals on three example chords

For example, a chord composed by $[C + E + G]$ has an interval of length 4 (between C and E) and another of length 7 (between C and G). Thus, a $M \times N$ sparse matrix is created for each score, representing the number of intervals of length $n \in N$ in the position $m \in M$. This sparse matrix is stored in coordinate format [17] to save memory. In our implementation, $N = 60$, given that 5 octaves can be represented ($5 \times 12 = 60$) and $M = 16$, since 16 is the maximum number of voices on a score. Finally, the Euclidean distance between the two matrices is computed.

5. *Voice coordination.* Measures the number of times that a score stops a note execution in two or more voices at the same time (the similarity between the detention of voices in two scores is quantified). Individuals whose voices are naturally coordinated have better fitness values.

6. *Note length similarity.* Calculates the weight of each possible note duration in the whole score, and computing the Euclidean distance between the two generated arrays.

7. *Melodic intervals.* Computes the difference between melodic intervals on the individuals, implementing a similar concept to N-gram [12], but using notes intervals instead of notes. The intervals in each voice are classified by their length. Then, a vector is computed to store the number of times each interval length is used, considering contiguous notes, and notes separated by a another (single) one. The output is the Euclidean Distance between the vectors calculated for two input scores. This criterion penalizes non-uniform melodies and, together with the scale detection function, it allows obtaining non-dissonant (and musically similar to the input scores) melodies. For example, in a voice composed by $[C5\ E5\ B4]$, there is an interval of length 5 and another one of length -6 for adjacent notes, and one of length -1 for separated notes. An example is shown in Figure 5.

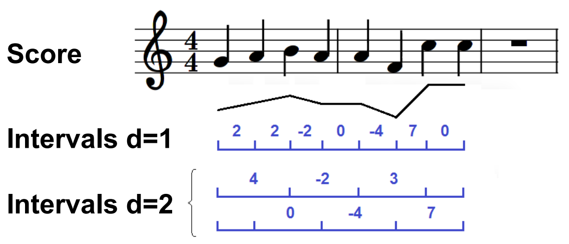


Figure 5: Melodic intervals on an example voice

After applying the seven described functions, the result is calculated by an addition, as shown in Figure 6. The first summand of this operation is the result of all the musical fitness criteria. The second summand is a lineal combination of the following terms: i) the average of the comparisons between the generated score and all the reference entry sounds and ii) the comparison to the most similar score found on the database. The coefficients in the linear aggregation were empirically adjusted in preliminary experiments, resulting in 0.3 for the average of all distances and 0.7 for the closest distance.

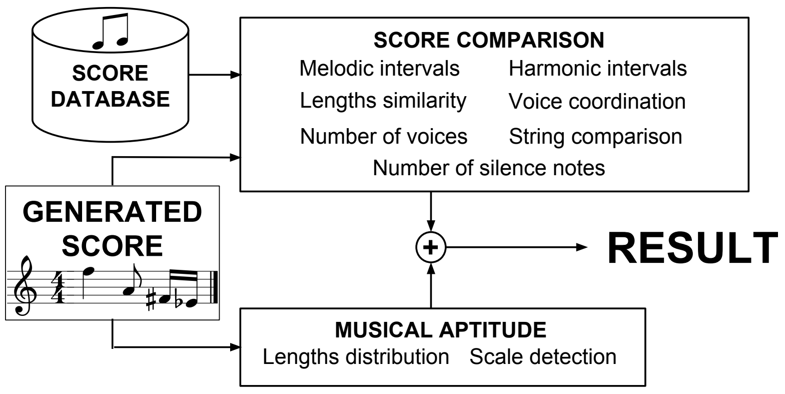


Figure 6: Diagram of the computation of the fitness function

The fitness value of an individual is the inverse of the RESULT value (better fitness values are associated to better compositions). The proposed approach for fitness calculation is a novel contribution to the field of automatic harmonic music composition using EAs.

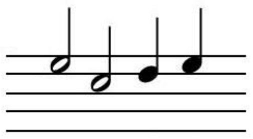
5.2.2 Penalization according to musical criteria

Five musical criteria are applied to penalize individuals that encode musically incorrect solutions.

Three criteria penalize solutions with trivial errors: *i*) scores where voices have different lengths; *ii*) scores where the last note is shorter than a quarter note, which implies an abrupt finish on the sound; and *iii*) scores where the total duration differs from those on the reference database in more than 20%.

Applying a scale detection criterion is very important to avoid dissonant sounds thus the fourth musical criterion applies the Krumhansl-Schmuckler (K-S) algorithm [18]. K-S weights each note on the score and compares those weights to an ideal distribution (using the Euclidean distance) for each key. The generated scores with shorter distances are selected as preferable over scores with larger distances. All major and minor tonalities are used for the comparison, but any other scale could be applied.

The fifth musical criterion is based on the idea that similar note lengths should exist within a voice. To check this property, voices where more than 50% of its notes differ in more than one position in the list of relative duration values are penalized. For example, a short voice with all whole and half notes (such as score 1 in Figure 7) is not penalized according to this criterion, while a voice composed by wholes, half, quarter and sixteenth notes (such as score 2 in Figure 7). As a result, score 1 has a better fitness value than score 2.


a: Score 1

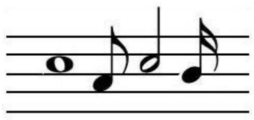

b: Score 2

Figure 7: Note lengths on different voices; score B is penalized due to having different note lengths.

5.3 Evolutionary operators

The details of the evolutionary operators used in the implementation are described next.

Selection. The stochastic universal sampling operator [2] was used, which provides an appropriate selection pressure for the search.

Recombination. The single point list crossover operator was implemented. Since the length of each parent is not necessarily the same, the crossover point is randomly selected between 1 and the length of the shortest parent.

Mutation. Four mutation operators were implemented. The first one modifies a randomly chosen note on a score, according to a fixed mutation probability assigned to each note component (note, octave, and length). This operator is used to introduce genetic diversity into the population. The second operator adds or deletes a new note on the mutated individual, in order to obtain similar lengths to the reference scores. The third one splits or joins a randomly selected note in the score. For example, the score $[A6i\ B4q\ B4q\ D6w]$ would result in $[A6i\ B4h\ D6w]$ (when joining the 2^{nd} and 3^{rd} notes) or $[A6i\ B4q\ B4q\ D6h\ D6h]$ (when splitting the 4^{th} note). This operator aims to modify the rhythm without changing the tones. The fourth operator permutes the order of the individual voices, which are important to the melody criterion within the fitness function.

Migration. The migration operator periodically exchanges one randomly selected individual between neighboring demes, considering a unidirectional ring topology. Migration is performed after a given number of generations (this period is called *epoch* in the implementation).

5.4 Evolution strategy and stopping criterion

A generational gap of populations was implemented, using an elitism rate of 5%. In each generation, the population is composed of the best fitted scores of the previous generation, all the offspring individuals resulted in the recombination, and the mutated scores.

A stagnation stopping criterion is applied. If the best solution found does not change during 10 epochs, the EA finishes its execution. This decision allows obtaining good compositions, having musical characteristics taken from a subset of the reference database.

6 Experimental analysis

This section describes the evaluation of the proposed parallel EA for harmonic music composition.

6.1 Experimental settings

Problem instances. The evolutionary optimization is performed using three score databases containing 5, 10, and 25 musical pieces composed by J. S. Bach. In addition, four databases are used to generate sample sounds in the validation stage, containing 10 scores composed by four selected authors: W. A. Mozart, L. van Beethoven, and P. I. Chaikovsky (classical music) and The Beatles (modern music). To generate the problem instances, a MIDI-to-codification translation module was implemented.

Experimental platform The proposed EA was implemented in the Java language using the Watchmaker framework [7]. The experimental analysis was performed on a Core i7-3770 processor with 16 GB RAM and Fedora 19 operating system.

Performance metrics. The similarity between the generated scores and the entry database is not easy to evaluate: for musical pieces, the human ear should be the ideal referee. The proposed metric takes into account two components: *i*) the distance of the resulting scores to the entries and *ii*) the distances between each entry score to the rest of the database. This metric evaluates both generated individuals distances and human-composed music distances, using the comparison criteria implemented on the fitness function. The validation stage is also based on providing links to generated music, leaving part of the performance evaluation to human (readers) criteria.

6.2 Parameter configuration

The parameter setting studied the best values for four important parameters in the proposed parallel EA: population size per deme (pS), generations per epoch (*epoch*), migrants per epoch (*mig*), and mutation probability (p_M). The candidate values were: $pS \in [80; 100; 120]$, *epoch* $\in [40; 50]$, *mig* $\in [5; 10]$, and $p_M \in [0.001; 0.005]$. The number of demes was set to 8, the elite population to 5, and the recombination probability to 0.75, after preliminary configuration experiments. Twenty independent executions of the proposed EA were performed for a problem instance with 10 Bach scores, different from those used in the experimental evaluation to avoid bias. The stopping criterion was set to 300 generations.

Table 1 reports the mean of the musical similarity results (*1/fitness*) and the execution time of the parallel EA for the different combinations of parameter values. The best results (with statistical significance) were computed using the configuration: *pS* = 100, *epoch* = 50, *mig* = 5, and p_M = 0.001.

pS	*epoch*	*mig.*	p_M = 0.001		p_M = 0.005	
			1/fitness	*time (s)*	*1/fitness*	*time (s)*
80	40	5	26.66	221.4	27.88	233.2
100	40	5	28.41	196.4	27.96	261.1
120	40	5	27.01	382.6	27.81	325.6
80	50	5	27.55	330.6	28.58	373.3
100	**50**	**5**	**25.87**	429.1	26.91	452.4
120	50	5	26.91	509.4	26.66	408.1
80	40	10	29.95	208.5	31.58	219.4
100	40	10	28.59	252.2	31.34	219.4
120	40	10	29.12	355.6	28.80	364.8
80	50	10	27.40	276.4	28.81	393.6
100	50	10	26.66	333.2	26.52	414.4
120	50	10	27.54	409.2	26.89	529.2

Table 1: Parameter configuration results

6.3 Experimental results and discussion

In order to evaluate the efficacy of the proposed EA, 40 independent executions were performed for each of the three experimental instances considered (J. S. Bach databases).

Results in Table 2 show that better fitness values are obtained in larger instances. The growth in fitness values is also observed in Figure 8, which shows the musicality distribution for the three experimental database instances. The histogram for the database with 5 scores has several good values and some outliers; the small instance size does not allow computing many highly fitted scores. On the contrary, the histogram for the database with 25 scores shows a more symmetric distribution and a larger number of highly fitted scores. These results confirm that larger instances are useful for generating better compositions, as they include more musical characteristics.

# scores	*1/fitness* ($\mu \pm \sigma$)	*time (s)* ($\mu \pm \sigma$)
5	30.09±2.57	454.8±113.4
10	27.31±2.62	413.6±108.9
25	27.05±2.88	378.9±95.5

Table 2: Results of the experimental analysis

	$\mu \pm \sigma$	*max*	*min*
generated score	37.37±37.33	156.26	3.03
Bach scores	47.38±13.07	74.18	29.86

Table 3: Distances for a generated score, and the scores in the database.

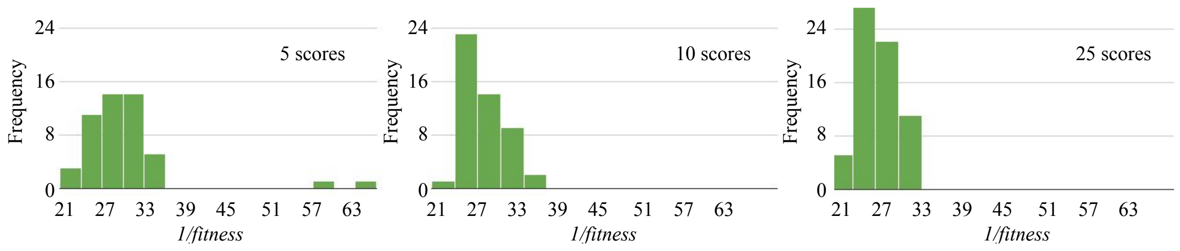


Figure 8: Histograms of the musicality result distributions for the sample instances

Table 3 reports the mean, standard deviation, and maximum/minimum distances between generated scores and each one of the 25 entry scores from the largest J. S. Bach database. These values show the similarities between an example output score and the compositions within the entry database. In addition, the same metrics are presented for the distances between each possible pair of Bach scores in the database. These results show that, according to the distance functions presented in Section 5.2.1, the generated score is more similar to the scores in the database than the own Bach compositions themselves.

Taking into account the resulting mean value, a high standard deviation is observed when comparing the generated individual to the database. On the contrary, the deviation for the Bach scores comparison is lower and a higher mean is found. This results imply the existence of individuals with large similarities to certain scores and, on the other hand, differences to others. This phenomenon is also found for the maximum/minimum values for the two comparisons, and is a specific goal of the proposed fitness function (capturing similarities but not converging to any particular score in the database). The minimum value for the individual comparison, combined with the low mean, shows the success of the process used for solving the underlying optimization problem.

In order to validate the results, the proposed EA is also applied to the other four score databases. The resulting values for the validation instances are presented in Table 4. The parallel EA achieves mean results and execution times values that are similar to those computed for the Bach instances. These results allow concluding that musical pieces composed by other musicians can be also used as reference for generating automatic compositions. Not only classical musical pieces are suitable to be used as entries: The Beatles songs generated empirically good-sounding results, as well as the other score databases.

	Mozart	*Beethoven*	*Chaikovsky*	*The Beatles*
1/fitness ($\mu\pm\sigma$)	31.49±2.57	28.54±1.97	32.05±2.67	30.11±2.47
time (s) ($\mu\pm\sigma$)	414.2±101.1	393.5±99.3	418.31±109.7	390.67±96.4

Table 4: Results obtained for the validation sample instances

All the generated musical pieces, along with the best individuals of the Bach 5, 10 and 25 scores database can be found at `http://www.fing.edu.uy/inco/grupos/cecal/hpc/TUME`.

7 Conclusions and future work

This article presents a parallel evolutionary approach for harmonic music composition that generates new musical pieces based on features from scores stored in a reference database.

The main contribution of the evolutionary approach is related to the proposed fitness function, which uses score comparison to evaluate the musical aptitude of generated compositions. Seven functions were implemented for evaluating similarities and performing comparisons between two input scores.

The proposed parallel EA was implemented in Java using the Watchmaker framework for evolutionary computation. Specific evolutionary operators were proposed to explore the space of possible compositions.ee Due to the large dimension of the search space and the hard-to-compute fitness function, a distributed subpopulations model was implemented.

The results obtained in the experimental analysis demonstrate that better fitness values (corresponding to better musical attributes) and shorter execution times are obtained as the size of the reference database growths. These results can be explained by the fact that each input score introduces new musical characteristics that can be incorporated in the generated score. Accurate results are reported for three different J. S. Bach databases. Moreover, similar results are shown for other four problem instances, considering musical pieces by other composers.

The main lines of future work are related to the implementation of other penalization functions based on music theory and score comparison, and executing the proposed EA in distributed memory systems to efficiently deal with larger databases in order to generate better compositions in lower execution times. In addition, other metaheuristics for harmonic music composition should be studied.

Acknowledgments

This work was partly supported by ANII (Agencia Nacional de Investigación e Innovación), PEDECIBA (Programa de Desarrollo de Ciencias Básicas), and ICT4V (Information and Communication Technologies for Verticals, R. Bayá, grant POS_ICT4V_2016_1_02).

References

[1] E. Alba, G. Luque, and S. Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013.

[2] T. Back, D. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[3] A. Burton and T. Vladimirova. Generation of musical sequences with genetic techniques. *Computer Music Journal*, 23(4):59–73, 1999.

[4] E. Cambouropoulos. Markov chains as an aid to computer assisted composition. *Musical Praxis*, 1(1):41–52, 1994.

[5] D. Conklin and I. Witten. Multiple viewpoint systems for music prediction. *Journal of New Music Research*, 24(1):51–73, 1995.

[6] R. Dean. *The Oxford Handbook of Computer Music*. Oxford University Press, New York, USA, 2009.

[7] D. Dyer. The Watchmaker framework for evolutionary computation. [Online] http://watchmaker.uncommons.org/, accessed January 26, 2015.

[8] I. Fujinaga and J. Vantomme. Genetic algorithms as a method for granular synthesis regulation. In *Proceedings of the International Computer Music Conference*, pages 138–138, 1994.

[9] L. Hiller. Computer music. *Scientific American*, 201:109–121, 1959.

[10] B. Jacob. Algorithmic composition as a model of creativity. *Organised Sound*, 1(3):157–165, 1996.

[11] K. Jones. Compositional applications of stochastic processes. *Computer Music Journal*, 5(2):45–61, 1981.

[12] M. Lo and S. Lucas. Evolving musical sequences with n-gram based trainable fitness functions. In *IEEE Congress on Evolutionary Computation*, pages 601–608, 2006.

[13] M. Marques, V. Oliveira, S. Vieira, and A. Rosa. Music composition using genetic evolutionary algorithms. In *IEEE Congress on Evolutionary Computation*, 2000.

[14] K. McAlpine, E. Miranda, and S. Hoggar. Making music with algorithms: A case-study system. *Computer Music Journal*, 23(2):19–30, 1999.

[15] E. Miranda. *Composing Music with Computers*. Butterworth-Heinemann, Newton, MA, USA, 2001.

[16] S. Nesmachnow. An overview of metaheuristics: accurate and efficient methods for optimisation. *International Journal of Metaheuristics*, 3(4):320–347, 2014.

[17] S. Pissanetzky. *Sparse matrix technology*. Academic Press, 1984.

[18] D. Temperley. What's key for key? the Krumhansl-Schmuckler key-finding algorithm reconsidered. *Music Perception: An Interdisciplinary Journal*, 17(1):65–100, 1999.

[19] C. Wu, C. Liu, and C. Ting. A novel genetic algorithm considering measures and phrases for generating melody. In *IEEE Congress on Evolutionary Computation*, pages 2101–2107, 2014.