A new fitness function for evolutionary music composition

Brian Martínez-Rodríguez[1][0000−0002−1109−0078]

International University of La Rioja (UNIR), Logroño, La Rioja, Spain
brian.martinez@unir.net
http://www.brianmartinez.es

Abstract. In this paper we propose a new fitness function for Evolutionary Computation purposes, based on a weighted by neighborhood average distance between two sequences of points within any metric space. We will apply this fitness function to the field of Computer-Assisted Composition focusing on the problem of thematic bridging, consisting in the evolutionary creation of a soft set of transitions between two given different melodies, the initial and the final one. Several self-adaptative strategies will be used to perform the search. A symbolic melody will be genotypically mapped into a sequence of genes, each of then containing the information of duration, frequency and time distance to following note. We will test the implementation of the fitness function by means of two experiments, showing some of the intermediate melodies generated in a successful run, and benchmarking every experiment with performance indicators for any of the three distinct evolutionary strategies implemented. The results prove this novel fitness function to be a quick and suitable way for individual evaluation in genetic algorithms.

Keywords: Evolutionary Computation · Genetic Algorithm · Computer-Assisted Composition · Fitness · Neighborhood · Thematic Bridging.

1 Introduction

The use of Evolutionary Computation in the field of computer-assisted composition has been widely addressed through a large variety of evolutionary techniques [1, 2]. In this paper we present a novel fitness function that can be employed on evolutionary algorithms that need to evaluate the dissimilarity between the genotype of individuals with different number of genes. We will focus on the specific problem of *Thematic Bridging* [5], conceived as obtaining automatically a set of smooth transitions from any given initial melody to any given goal melody. To achieve this, several evolution strategies [4] will be implemented and afterwards tested in two experiments, measuring the performance indicators for several settings. The evolutionary search, thanks to mutation and crossover operators, will progressively minimize the dissimilarity of every offspring, until an individual reaches the objective genotype of the goal melody. The best individuals of every offspring will be stored and will create the desired bridging, that could be written into a *musicXML* interchange format file [3].

2 Related work

The use of evolutionary algorithms in the field of Computer-Assisted Composition is said to have started at the beginning of the 90's. In the year 1991, Horner and Goldberg [5] implemented one of the first applications of the evolutionary algorithms to computer composition, describing an evolutive technique called *Thematic Bridging* that is able to produce melodic material as a result of iterative transitions between two small melodies. The composition is created by means of human selection and organization of the algorithmically created melodic material, in a form of an imitative five-voices canon. Three years later, Biles [6] presented *GenJam*, a noteworthy application of genetic computation that generates improvisations in Jazz style, keeping the hierarchical relations between different melodic ideas suggested by the harmonic chords progression that is playing. At the same time, the system retrieves feedback information in real-time from the human player. Other interesting evolutive designs were proposed by Hartman [9], Mcintyre [10], Horner [7] and Jacob [8] .

De la Puente et al. [11] introduced GEMUSIC, a tool that creates algorithmically melodic lines similar to human compositions, thanks to the implementation of Evolutionary Grammars. Weinberg et al. [12] described an interactive evolutive robotic system that collaborates with human players and improvises while playing on a xylophone. The system detects the musical material played by the human and evolves it using several fitness functions. Tzimeas et al. [13] developed the software *Jazz Sebastian Bach*, a system that evolves melodies originally composed by J. S. Bach and turns them into a jazzy style. The authors propose a fitness function called *Critical Damped Oscillator* that overcomes several algorithmic difficulties related to *Automatic Fitness Assessment* (AFA) or *Interactive Genetic Algorithm* (IGA). De León et al. [14] proposed the characterization of a melody as the result of a set of rules coming from a fuzzy genetic algorithm, aimed to distinguish if a given MIDI file contains a melody or not. The figure of the human-expert knowledge is replaced by a fuzzy genetic system. Sánchez et al. developed MELOMICS project [15], a sophisticated evolution system able to compose and orchestrate whole musical pieces.

In 2015 Hofmann presented an evolutionary algorithm [16] that uses a tree-based domain model of compositions, representing the musical pieces as a set of constraints that are able to change over time. The system uses multi-objective optimization functions. Scirea et al. presented in [17] the framework MetaCompose, for music composition that includes a chord sequence and accompaniment generator, and a melody generator that uses a novel evolutionary technique combining FI-2POP and multi-objective optimization. Loughran and O'Neill introduced in [18] a system that generates autonomous fitness functions for evolving short melodies using a population of agents and clustering methods. In 2019, Nam YW. and Kim YH. [19] automatized the production of good-quality jazzy melodies by means genetic algorithm, using a variable-length chromosome and geometric crossover. Trump proposed in [20] a evolutionary framework for improvisation in which the improvisation is created by successive sound cells containing a musical content transformed by a creative selection.

3 Theoretical background

3.1 Mathematical definition of a melody

As we presented in [21], a melody can be understood as a sequence of points, $\mathscr{M} = \{\mathbf{x}_i\}_{i=1}^{n}$, where each point $\mathbf{x}_i \in \mathbb{R}^q$ is a musical note. The most simple way to represent a note is using three musical characteristics: the duration, the frequency and the time distance that could exist until the next note (representing in this way the possible existence of a silence between this note and the next one). Thus, a musical note will be expressed by a point within a three-dimensional metric space $\mathbf{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$, where the feature x_1 expresses the time duration of the note, x_2 expresses the frequency and x_3 indicates time duration of an optional silence until the next note. In order to calculate the time features x_1 and x_3 of each note, we will use the *Relative Coefficient Duration* δ proposed in [21]. For the symbolic representation of the frequency in the feature x_2 we will use the *MIDI pitch number* associated to any musical note.

3.2 Neighbourhood functions

Neighbourhood functions introduced in [21], [22] are the key point of the fitness function that will be introduced in the following section. When making a comparison between two sequences, with the first one having a number of n elements and the second one having a number of m elements, the aim of the neighbourhood function will be to calculate the degree of relationship between any element i from the first sequence and any element j of the second one.

In this way, when comparing two sequences A and B with very different number of elements, if a correct function is defined, the first elements of sequence A will be strongly correlated with the first elements of the sequence B, but very weakly correlated with the final elements of B. In addition, the ending elements of sequence A will be weakly correlated with the first elements of the sequence B, but strongly correlated with the final elements of B. Equation 2 shows the expression of Gaussian Neighbourhood Functions used in this paper.

$$f(i,j) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left[\frac{1}{2\sigma^2}\left(i - \frac{(n-1)}{(m-1)}j\right)^2\right]}. \tag{1}$$

3.3 Fitness function

We propose a new fitness function for evolutionary music composition based on the definition of *Melodic Dissimilarity* proposed in [21]. Let $\mathscr{M}^A = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\} \in \mathbb{R}^q$ and $\mathscr{M}^B = \{\mathbf{y_1}, \ldots, \mathbf{y_m}\} \in \mathbb{R}^q$ be two different melodies constructed by a different number of notes. Let $d : \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}$ be any distance function on the metric space. Let $f(i,j)$ be any neighborhood function. The Neighborhood Average Dissimilarity $\mathscr{D}$ from melody $\mathscr{M}^A$ to melody $\mathscr{M}^B$ is defined as

$$\mathscr{D}(\mathscr{M}^A, \mathscr{M}^B) = \frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} f(i,j) \cdot d(\mathbf{x_i}, \mathbf{y_j}). \tag{2}$$

The proposed fitness function of each individual will be constructed by de absolute difference between the dissimilarity of each individual A with the goal melody B minus the dissimilarity of the goal melody with itself. Consequently, the expression for the Fitness Function is

$$F_{fitness}(\mathscr{M}^A) = |\mathscr{D}(\mathscr{M}^A, \mathscr{M}^B) - \mathscr{D}(\mathscr{M}^B, \mathscr{M}^B)| \tag{3}$$

Observe how expression (2) does not accomplish any of the requirements of a distance function. Consequently $\mathscr{D}(\mathscr{M}^B, \mathscr{M}^B) \neq 0$, for the most of the cases.

4 Genetic Algorithm

4.1 Genotype Representation

A melody will be represented into an individual. In the genotype, the sequence of all notes is codified into the sequence of genes. Each gene contains the minimum information of a note. For the following experiments, three different representations of an individual genotypes will be used, each of one containing the required information by any of the evolutionary strategies [6] tested: *Simple mutation*, *Uncorrelated mutation with One Step Size*, and *Uncorrelated mutation with n Step Sizes*.

Simple mutation In this representation, the genotype of every individual is a sorted array of float numbers in which the three features x_1, x_2 and x_3 of every note (*gene*) will be stored by order. The length of the array will be $3 \times n$, where n is the number of notes of the melody that is coded for each individual. The representation is as follows

$$(x_1^1, x_2^1, x_3^1, x_1^2, x_2^2, x_3^2, \ldots, x_1^n, x_2^n, x_3^n) \tag{4}$$

Uncorrelated mutation with one Step Size In this representation, the genotype of each individual is again a sorted array in which the three features of every note have been stored, besides three values σ belonging to features of time duration, pitch and time distance. The length of the genotype will be $3 \times n + 3$, being n the number of notes of the melody, and its structure will be the following:

$$(x_1^1, x_2^1, x_3^1, x_1^2, x_2^2, x_3^2, \ldots, x_1^n, x_2^n, x_3^n; \sigma_1, \sigma_2, \sigma_3) \tag{5}$$

Uncorrelated mutation with *n* Step Sizes In addition to the previous information of time duration, pitch and time distance belonging to each one of the n notes, this representation will include the sigma value σ corresponding to each one of this features. In this case, the length of the genotype will be $6 \times n$ and its structure:

$$(x_1^1, x_2^1, x_3^1, \ldots, x_1^n, x_2^n, x_3^n; \sigma_1^1, \sigma_2^1, \sigma_3^1, \ldots, \sigma_1^n, \sigma_2^n, \sigma_3^n) \tag{6}$$

4.2 Restrictions on the evolution strategy

Some constrains will be implemented into the evolution strategy aimed to reduce the space of research. The first restriction is introduced in the mutation of the MIDI pitch value. The mutated pitch value will be forced to be a integer number, due to the MIDI mapping of the musical notes is enclosed from 0 to 127, so the algorithm does not consider the possible existence of intervals smaller than a semitone.

The second constrain is implemented into the possible variation of the Relative Duration Coefficient δ. This feature will be mutated by means of adding or subtracting a multiple of a minimum duration figure. The arbitrarily chosen minimum duration is a demisemiquaver (thirty-second note), with coefficient $\delta_{min} = 0.03125$.

4.3 Mutation

The mutation in a specific gene will be done using *random resetting*, establishing an uniform mutation probability for every genes. When a gene is randomly chosen for being mutated, the feature represented by this gene into a float number will be modified adding or subtracting a certain amount, calculated according to the case that we consider.

Simple mutation In the case of simple mutation, the features of duration x_1, pitch x_2 and time distance x_3 of a selected note i will be modified using these expressions based on the equations exposed in [4]:

$$\begin{aligned} x'^{i}_{1} &= x^i_1 + \delta_{min} \cdot \lceil \sigma_1 \cdot N(0,1) \rceil \\ x'^{i}_{2} &= x^i_2 + \lceil \sigma_2 \cdot N(0,1) \rceil \\ x'^{i}_{3} &= x^i_3 + \delta_{min} \cdot \lceil \sigma_3 \cdot N(0,1) \rceil \end{aligned} \tag{7}$$

where $N(0,1)$ is a generator of Gaussian distributed random numbers, centered in the zero value (mean equal to zero), and with standard deviation equal to 1. In this case, the values of σ stay constant.

Uncorrelated mutation with one Step Size In this kind of mutation, the values σ_1, σ_2 y σ_3 used to calculate the changes in the features of duration, pitch and time distance, are assumed to change randomly for each individual. Therefore, the mutations on the standard deviations and features x_i will be done by means of the following expressions:

$$\begin{aligned} \sigma'_1 &= \sigma_1 \cdot e^{\tau \cdot N(0,1)} & x'^{i}_{1} &= x^i_1 + \delta_{min} \cdot \lceil \sigma_1 \cdot N(0,1) \rceil \\ \sigma'_2 &= \sigma_2 \cdot e^{\tau \cdot N(0,1)} & x'^{i}_{2} &= x^i_2 + \lceil \sigma_2 \cdot N(0,1) \rceil \\ \sigma'_3 &= \sigma_3 \cdot e^{\tau \cdot N(0,1)} & x'^{i}_{3} &= x^i_3 + \delta_{min} \cdot \lceil \sigma_3 \cdot N(0,1) \rceil \end{aligned} \tag{8}$$

We should check that, for every mutation of the deviation σ_j, the new value is not too small. To achieve this, we establish a threshold value ϵ_j below which σ can not still decrease, so:

$$\begin{aligned} \sigma'_1 < \epsilon_1 \Rightarrow \sigma'_1 = \epsilon_1 \\ \sigma'_2 < \epsilon_2 \Rightarrow \sigma'_2 = \epsilon_2 \\ \sigma'_3 < \epsilon_3 \Rightarrow \sigma'_3 = \epsilon_3 \end{aligned} \tag{9}$$

Uncorrelated mutation with n Step Sizes In this case, each one of the x_j^i features of an individual will mutate with a specific deviation σ_j^i. The mutations of the deviations and features of a chosen note i will be carried out with these expressions:

$$\begin{aligned} \sigma'^i_1 = \sigma^i_1 \cdot e^{\tau \cdot N(0,1)} \qquad x'^i_1 = x^i_1 + \delta_{min} \cdot \lceil \sigma^i_1 \cdot N(0,1) \rceil \\ \sigma'^i_2 = \sigma^i_2 \cdot e^{\tau \cdot N(0,1)} \qquad x'^i_2 = x^i_2 + \lceil \sigma^i_2 \cdot N(0,1) \rceil \\ \sigma'^i_3 = \sigma^i_3 \cdot e^{\tau \cdot N(0,1)} \qquad x'^i_3 = x^i_3 + \delta_{min} \cdot \lceil \sigma^i_3 \cdot N(0,1) \rceil \end{aligned} \tag{10}$$

Once again, is necessary to check that the mutated value of every deviation σ_j^i is not smaller than a given threshold ϵ_0.

Mutation operation concerning the number of notes of a melody Besides mutating the features of duration, frequency and time distance of a random note from the melody coded on the genotype of each individual, it is needed to establish a mutation operation to change the number of notes of the melody, since we want to achieve an evolutionary transition from one initial melody to a second one, both having conceivably a different number of notes.

Two arbitrary probabilities for insertion and suppression of a note will be implemented in order to insert a new note in a random position p inside the genotype, or remove the note located on the position p, respectively.

When inserting a brand new note in a random position of the genotype, there exists three different possibilities:

- *Inserting the note at the beginning of the melody* ($p = 0$): In this case, three new positions will be inserted at the very beginning of the genotypical array. The values of these three new positions will duplicate exactly the three features of the prior first note, so the new inserted note will duplicate exactly the ancient first one.
- *Inserting the note at the end of the melody* ($p = 3 \cdot n$): In this case, three new positions will be added at the end of the genotypical array. The values of this positions will duplicate the previous last note of the melody.
- *Inserting the note in an intermediate position within the melody* (p=k, $0 < k < 3 \cdot n$): In this case, a new note will be inserted between the notes places in the positions $k - 1$ y k. Each one of the three features of the new note will be calculated as an average value of the corresponding feature from the

two adjacent notes, taking into account the previously specified constraints of mutation changes in duration and pitch.

In the cases of the genotype related to the representation of Uncorrelated mutation with one Step Size and Uncorrelated mutation with n Step Sizes, it will also be necessary to include in the genotypical array one extra position in case of one Step size, or three extra positions in case os n Step Sizes, in order to include the new σ values corresponding to the new note.

4.4 Initialization of the population, parents selection and crossover

The population will be initialized creating a number μ of different individuals, whose genotypes have been initially cloned from the starting melody one, and afterwards subjected to a randomly mutation process.

A number of λ couples of parents will be randomly chosen to generate a new child from every couple. The recombination operation for the new genotype will be the uniform crossover, so each gene will be randomly inherited from any of the parents.

The selection process of survivors for next generation will be guided by method $\mu + \lambda$, that involves mixing together the population of parents and offspring [6], sorting by each individual's fitness and choosing the best μ individuals.

4.5 Performance indicators

For each execution there will be a maximum of 200 generations. Each experiment will be executed 1.000 times for any one of the pre-established setups. The algorithm will store the following performance indicators [6]:

- **SR** (Success Rate): Percentage of executions that finish successfully over the total number of executions.
- **MB** (Mean Best Fitness): Average of the best fitness value of the population when execution finishes, successfully or not.
- **MBFS** (Mean Best Fitness Success): Average of the better fitness value of the population taking into account only the successful executions.
- **AES** (Average number of Evaluations to a Solution): Average number of generations needed to reach a successful execution.
- **MST** (Mean Success Time): Mean time needed to find a successful execution.

5 Experiments

5.1 First experiment

Given initial melody A and goal melody B, use the evolutionary strategies with fitness function (3) to generate a melodic transition from A to B.

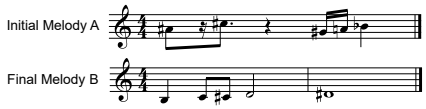

Fig. 1: Initial and final melodies of the first experiment.

Settings of the algorithm: $\mu = 20$, $\lambda = 200$, mutation prob.=0.15 and note insertion prob.= 0. Run 1.000 times. Results shown in Table 1 and Fig. 2:

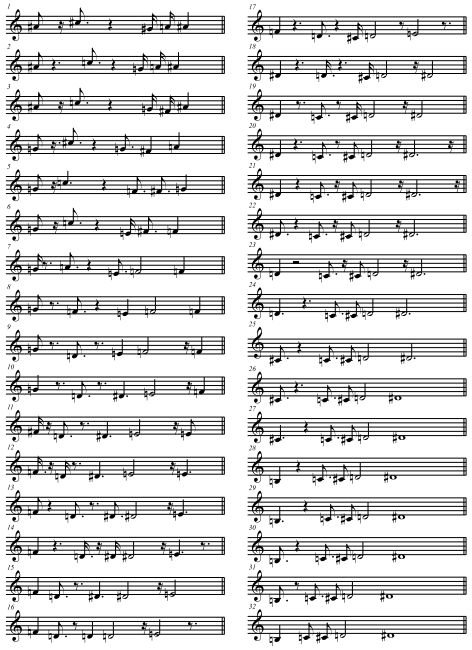

Fig. 2: Some intermediate melodies generated at one successful execution.

Table 1: Benchmarking indicators for experiment one.

	Simple	One Step	n Steps
SR	100.00%	100.00%	100.00%
MBFS	0.01904	0.01904	0.01903
MB	0.01904	0.01904	0.01903
AES	43.80	99.60	97.30
MST(ms)	167.86	373.03	378.77

5.2 Second experiment

Given initial melody A and goal melody B, use the evolutionary strategies with fitness function (3) to generate a melodic transition from A to B.

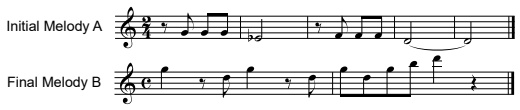

Fig. 3: Initial and final melodies of the second experiment.

Settings of the experiment: $\mu = 20$, $\lambda = 500$, mutation prob.$= 0.15$ and note insertion prob.$= 0.05$. Run 1.000 times. Results shown in Table 2 and Fig. 4.

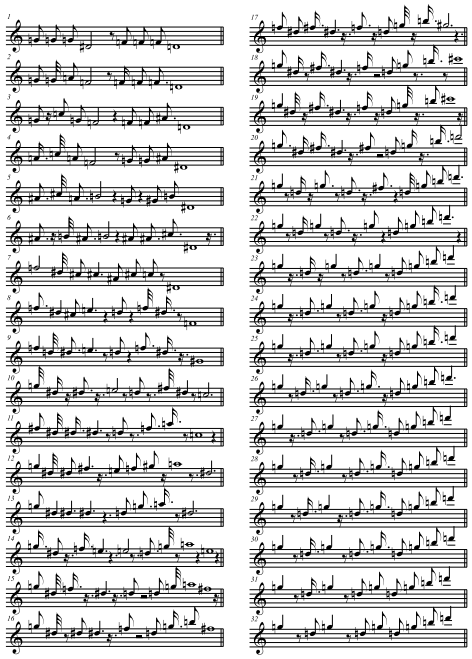

Fig. 4: Some intermediate melodies generated at one successful execution.

Table 2: Benchmarking indicators for experiment two.

	Simple	One Step	n Steps
SR	21.30%	34.60%	40.70%
MBFS	0.38764	0.38764	0.38764
MB	0.41251	0.40294	0.40050
AES	55.60	56.50	49.90
MST (ms)	1490.69	1490.53	1331.01

5.3 Results

In Fig. 5 it is possible to compare the performance curves for experiment one and two, for simple mutation, Uncorrelated mutation with one Step Size and Uncorrelated mutation with n Steps Size. The final benchmarks for the performance indicators of SR, AES and MST are summarized in Table 3.

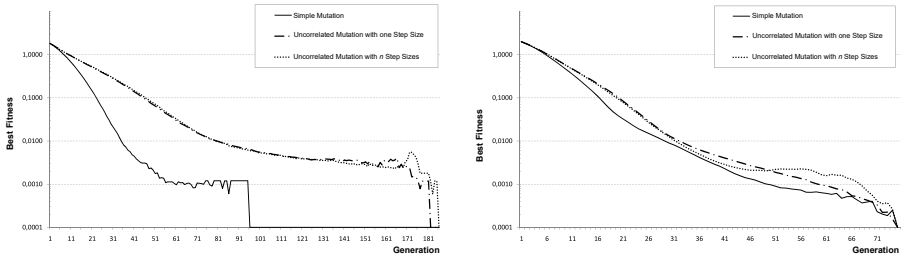

Fig. 5: Performance charts for the first and second experiment.

Table 3: Final performance indicators for the experiments.

	SR			**AES**			**MST**		
	Simp.	1 Step	n Steps	Simp.	1 Step	n Steps	Simp.	1 Step	n Steps
Exp. #1	100,00%	100,00%	100,00%	43.80	99.60	97.30	167.86	373.03	378.77
Exp. #2	21.30%	34.60%	40,70%	55.60	56.50	49.90	1490.69	1490.53	1331.01

6 Conclusions

The evolutionary algorithm implemented and tested in the two experiments has proved to find solutions to the problem of thematic bridging between two melodies, thanks to the minimization of the novel Fitness Function proposed in this paper (3) and based on the Neighborhood Average Dissimilarity (2). The searching process has been carried out using three different self-adaptative evolution strategies: Simple Mutation, Uncorrelated mutation with one Step Size and Uncorrelated mutation with n Steps Sizes. The parent selection has been randomly done, with no tournament techniques, and we have used discrete recombination in which each one of the parent alleles is randomly chosen with equal chance for either parents. To determine the survivor selection, the best μ individual from the union of parents and offspring it has been chosen ($\mu + \lambda$ selection). Besides the usual mutation operations on the genotype, it has been implemented the possibility to enlarge or lessen the number of genes of each child, since each gene encodes the basic music information of a note (duration

coefficient, pitch and silence time distance), and also our compositional problem of evolutionary thematic bridging requires possible changes in the number of notes of the offspring melodies. We have run 1000 executions for each set of experiments one and two, with a maximum number of 200 offsprings for each execution. The Success Rate of experiment one was 100%, due to the melodies not being very distant in terms of evolutionary search. The Success Rate of experiment two was 40,7% as the initial and final melodies where very distant. For the experiment one, the most efficient mutation was the Simple Mutation. Nevertheless, Uncorrelated mutation with n Steps Sizes has been the most efficient in experiment two. All the representations achieved low values of mean success time (MST) and low average number of evaluations to a solution (AES), exposing the potentiality of our Fitness Function (3) to be used in more sophisticated evolutionary techniques. Future work will involve widening the rhythmical restrictions of the evolutionary algorithm and incorporating the generation of harmonic chord sequences through a convenient modification of the Fitness Function.

References

1. Miranda, E. and Al Biles, J.: Evolutionary computer music. Springer-Verlag, London (2007).
2. Romero, J. J.: A handbook on evolutionary art and music. Springer-Verlag, Berlin Heidelberg,(2008).
3. Good, M., and Actor, G. : Using MusicXML for file interchange. In: Web Delivering of Music, International Conference, pp. 153-153. IEEE Computer Society, Leeds, UK (2003).
4. Eiben, A., Smith, J. : Introduction to evolutionary computing. Springer, Berlin, Heidelberg,(2015).
5. Horner, A. and Goldberg, D.: Genetic algorithms and computer-assisted music composition. In: Proceedings of the 1991 International Computer Music Conference, vol. 51, pp. 479–482. Montreal, Canada (1991).
6. Biles, J. A.: GenJam: A genetic algorithm for generating jazz solos. In: Proceedings of the 1994 International Computer Music Conference, pp. 131–137. International Computer Music Association, Copenhagen, Denmark (1994).
7. Horner, A. and Ayers, L.: Harmonization of Musical Progressions with Genetic Algorithms. In: Proceedings of the 1995 International Computer Music Conference, pp. 483–484. International Computer Music Association, Banff, Canada (1995).
8. Jacob, B.: Composing with genetic algorithms. In: Proceedings of the 1995 International Computer Music Conference, pp. 452–455. International Computer Music Association, Banff, Canada (1995).
9. Hartmann, P.: Selection of musical identities. In: Proceedings of the 1990 International Computer Music Conference, pp. 234–236. International Computer Music Association, Glasgow, Scotland (1990).
10. McIntyre, R.: Bach in a box: The evolution of four part baroque harmony using the genetic algorithm. In: Proceedings of the First IEEE Conference on Evolutionary Computation, pp. 852–857. IEEE, Orlando, Florida (1994).
11. de la Puente, A., Alfonso, R. and Moreno, M.: Automatic composition of music by means of grammatical evolution. In: Proceedings of the 2002 conference on

APL: array processing languages, pp. 148–155. Association for Computer Machinery, Madrid, Spain (2002).
12. Weinberg G., Godfrey M., Rae A., Rhoads J.: A Real-Time Genetic Algorithm in Human-Robot Musical Improvisation. In: Kronland-Martinet R., Ystad S., Jensen K. (eds) Computer Music Modeling and Retrieval. Sense of Sounds. CMMR 2007. Lecture Notes in Computer Science, vol 4969. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-85035-9_24
13. Tzimeas, D. and Mangina, E.: A GA Tool for Computer Assisted Music Composition. In: Proceedings of 2007 International Computer Music Conference. International Computer Music Association. Copenhagen, Denmark (2007). https://quod.lib.umich.edu/i/icmc/bbp2372.2007.019/1
14. de León, P., Rizo, D., Ramirez, R and Iñesta, J.: Melody characterization by a genetic fuzzy system. In: Proceedings of the 5th Sound and Music Computing Conference. Sound and Music Computing Association. Berlin, Germany (2008).
15. Sánchez, C., Moreno, F., Albarracín, D., Fernández, J. and Vico, F.: Melomics: A case-study of AI in Spain. AI Magazine **34**(3), 99–103 (2013)
16. Hofmann D.: A Genetic Programming Approach to Generating Musical Compositions. In: Johnson C., Carballal A., Correia J. (eds) Evolutionary and Biologically Inspired Music, Sound, Art and Design. EvoMUSART 2015. Lecture Notes in Computer Science, vol 9027. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16498-4_9
17. Scirea M., Togelius J., Eklund P., Risi S.: MetaCompose: A Compositional Evolutionary Music Composer. In: Johnson C., Ciesielski V., Correia J., Machado P. (eds) Evolutionary and Biologically Inspired Music, Sound, Art and Design. EvoMUSART 2016. Lecture Notes in Computer Science, vol 9596. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31008-4_14
18. Loughran R., O'Neill M.: Clustering Agents for the Evolution of Autonomous Musical Fitness. In: Correia J., Ciesielski V., Liapis A. (eds) Computational Intelligence in Music, Sound, Art and Design. EvoMUSART 2017. Lecture Notes in Computer Science, vol 10198. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55750-2_11
19. Nam YW., Kim YH.: Automatic Jazz Melody Composition Through a Learning-Based Genetic Algorithm. In: Ekárt A., Liapis A., Castro Pena M. (eds) Computational Intelligence in Music, Sound, Art and Design. EvoMUSART 2019. Lecture Notes in Computer Science, vol 11453. Springer, Cham (2019).https://doi.org/10.1007/978-3-030-16667-0_15
20. Trump S.: Sound Cells in Genetic Improvisation: An Evolutionary Model for Improvised Music. In: Romero J., Ekárt A., Martins T., Correia J. (eds) Artificial Intelligence in Music, Sound, Art and Design. EvoMUSART 2020. Lecture Notes in Computer Science, vol 12103. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43859-3_13
21. Martínez–Rodríguez B., Liern V.: A Fuzzy-Clustering Based Approach for Measuring Similarity Between Melodies. In: Agustín-Aquino O., Lluis-Puebla E., Montiel M. (eds) Mathematics and Computation in Music. MCM 2017. Lecture Notes in Computer Science, vol 10527. Springer, Cham (2017).https://doi.org/10.1007/978-3-319-71827-9_21
22. Martínez–Rodríguez B., Liern V.: Mercury: A Software Based on Fuzzy Clustering for Computer-Assisted Composition. In: Montiel M., Gomez-Martin F., Agustín-Aquino O. (eds) Mathematics and Computation in Music. MCM 2019. Lecture Notes in Computer Science, vol 11502. Springer, Cham (2019).https://doi.org/10.1007/978-3-030-21392-3_19