Mercury® : A software based on fuzzy clustering for computer-assisted composition

Brian Martínez Rodríguez[1] and Vicente Liern Carrión[2]

[1] Universidad Politécnica de Valencia, Valencia, Spain
brian.martinez.rodriguez@gmail.com
[2] Dep. Matemáticas para la Economía y la Empresa,
Universidad de Valencia, Spain.
vicente.liern@uv.es

Abstract. We present MERCURY, a new software for computer-assisted composition based on algorithms of fuzzy clustering. This software is able to generate a big number of transitions between whatever two different melodies, harmonic progressions or rhythmical patterns. Mercury works with symbolic music notation, therefore the software is able to read music and to export the generated musical production into MusicXML format. This paper will focus on the theoretical aspects of the CFT-algorithm, implemented in the software for creating these complete transitions, overviewing the structure of the program as well the user interface and its music notation module. Finally, several computational examples will show the wide variety of compositive possibilities that Mercury brings.

Keywords: algorithmic · composition · computer · fuzzy · clustering.

1 Introduction

The first public performance of the *Illiac Suite* on 9th August of the year 1956 at the Illionis University [1], is considered to be the starting point of the computer-assisted composition. Along the last six decades, a big number of programs have been developed bringing new technical possibilities and widening the compositive abilities of the computers. The main disciplines that have been extensively explored can be summarized in [15]: using Markov models for generation of musical structures or sequences [6], creating generative grammars for the production of musical structures [16], algorithmic composition based on chaos theory and self-similarity [5], approach to composition and variation using genetic algorithms [3], use of cellular automata [13], neural networks and more recently techniques coming from artificial intelligence and machine learning [9].
The techniques of data clustering, either crisp or fuzzy, have been successfully applied to the field of computer-assisted musical analysis, used for mode recognition [18], musical patterns analysis [4] among others goals like style classification, etc. In the past edition of MCM 2017 Conference, we proposed the algorithm FOCM for comparing some ordered sequences of notes [11]. The software that is

presented now is able to generate variations and transitions from a given initial musical material to a given final one. For that purpose, we have constructed a new fuzzy clustering technique that transforms any initial sequence into a final sequence even though the number of elements is different. Besides, the software generates a transition accumulating all the intermediate states. The musical adaptation of this algorithm has been implemented in the software Mercury, that provides the users with three basic kinds of transitions, according to the three tradicional characteristics of music: melodic transition, harmonic transition and rhythmical transition. Once the user has decided which one will be used, Mercury reads the input data from *MusicXML* files and parses the information that at any case is required. Once user has set up the parameters, experimented with transitions and evaluated the results, the music material generated by the computer can be exported to a new MusicXML file, and subsequently loaded into any music notation software (e.g. *Musescore*, *Finale*, *Sibelius*, etc.) where the composer can continue with the art of music composition.

2 Theoretical background

According to the definition given in [11] for the FOCM algorithm, we can express the sequence of notes as:

Definition 1. *Let $\boldsymbol{x}$ be a musical note determined by q characteristics (pitch, intensity, duration, silence, etc.) expressed as a vector in $\mathbb{R}^q$ A melody of n notes is a sequence $\mathscr{M} = \{\boldsymbol{x}_i\}_{i=1}^n$, where each $\boldsymbol{x}_i \in \mathbb{R}^q$ is a musical note.*

As in this work we will use harmonic sequences, the treatment of the harmony will be similar to the sequence of notes. Moreover, as in this case we are only interested in the pitch, the notes can be expressed as real numbers, and consequently a chord of k notes will be an element of $\mathbb{R}^k$.

Definition 2. *Let $\boldsymbol{y} \in \mathbb{R}^k$ be a chord defined by a k-tuple of MIDI pitch values $\in [0, 127]$ expressed as a vector in $\mathbb{R}^k$. A k-harmony $\mathscr{H}$ is the sequence $\mathscr{H} = \{\boldsymbol{y}_i\}_{i=1}^n$ where each $\boldsymbol{y}_i \in \mathbb{R}^k$ is a chord of k MIDI pitches.*

To study the rhythm we will restrict the notation. A rhythmical element is defined by three characteristics: the duration coefficient $\delta \in]0, \infty[$, the silence $\in [0, 1]$ and the MIDI velocity $\in [0, 127]$. This duration coefficient was obtained in [11] as

$$\delta = \sum_{i=1}^{\tau} (\alpha_i \cdot \beta_i \cdot \gamma_i) = \sum_{i=1}^{\tau} \left[\frac{1}{2^{a_i}} \cdot \frac{2^{b_i+1} - 1}{2^{b_i}} \cdot \frac{d_i}{c_i} \right]. \tag{1}$$

Definition 3. *A rhythm pattern of n elements $\mathscr{R}$ is a sequence $\mathscr{R} = \{\boldsymbol{z}_i\}_{i=1}^n$ where each $\boldsymbol{z}_i \in \mathbb{R}^3$ is a rhythmical element.*

2.1 The main algorithm

Mercury is able to generate complete transitions between two different melodies, harmonies or rhythmical patterns by means of a recursive application of the

FOCM algorithm. The process works in this way: Once the two melodies (or two harmonic sequences or rhythmical patterns) have been represented as two sequences of elements in the metric space $\mathbb{R}^q$, the algorithm will initialize the set of centroids with the sequence that is desired to be changed, usually the shorter one having c, being $c < n$, denominated as *sequence B*. The final sequence with n elements, our goal, is denominated *sequence A* and is assigned to the points of the data set about which the partition will be calculated. Once FOCM has ended, a new centroid will be added to the set of centroids and FOCM will run again, repeating this process until the number of centroids (sequence B) is equal to the number of points of the data set (sequence A). At that point, the algorithm converges totaly and sequence B becomes equal to sequence A.

STEP 1. Choose any convenient neighbourhood function.
STEP 2. Choose any convenient distance function d in $\mathbb{R}^q$. Establish the fuzzyness parameter λ, $1 \leq \lambda < \infty$. Set $c_0 = c$.
STEP 3. Initialize the centroids: assign c_0 to the values of the initial sequence B. For each iteration l do:
STEP 4. If $c_l > n$ then stop; else if $c_l \leq n$ do:
STEP 5. If $c_l = n$ select discrete neighbourhood function (see Table 2.3), otherwise continue with the initial neighbourhood function.
STEP 6. Update $\widehat{U}^{(l)}$ and $\{\mathbf{v}_j^{(l)}\}$ using equations

$$u_{ij} = \frac{1}{\sum\limits_{k=1}^{c} \left[\frac{\|\mathbf{x}_i - \mathbf{v}_j\|}{\|\mathbf{x}_i - \mathbf{v}_k\|}\right]^{\frac{2}{\lambda-1}}}, \; \hat{u}_{ij} = \frac{u_{ij} \cdot f(i,j)}{\sum\limits_{k=1}^{c} u_{ik} \cdot f(i,k)}, \; \mathbf{v}_j = \frac{\sum\limits_{i=1}^{n} \hat{u}_{ij}^{\lambda} \cdot \mathbf{x}_i}{\sum\limits_{i=1}^{n} \hat{u}_{ij}^{\lambda}}. \quad (2)$$

STEP 7. Compare $\widehat{U}^{(l)}$ with $\widehat{U}^{(l+1)}$ using any convenient matrix norm, being $\epsilon \in (0,1)$ an arbitrary termination criterion. If $\| \widehat{U}^{(l+1)} - \widehat{U}^{(l)} \| \leq \epsilon$ then go to STEP 8. Otherwise set $l = l + 1$, calculate the centroids $\{\mathbf{v}_j^{(l)}\}$ with $\widehat{U}^{(l)}$ and $\hat{u}_{ij}$ given in (2) and go back to STEP 6.
STEP 8. Choose the two consecutive points h, $h+1$ with $1 \leq h \leq c_l - 1$ for which the distance between them is maximum. In other words, for the current iteration l select a point h such that

$$\max_{1 \leq h \leq c_l - 1} [d(\mathbf{v}_h^l, \mathbf{v}_{h+1}^l)]. \quad (3)$$

STEP 9. Set $l = l + 1$. Add to $\mathbf{V}^l$ a new centroid in the position $h + 1$ of the sequence of the centroids, with the attributes calculated as the average: $(\mathbf{v}_h^l + \mathbf{v}_{h+1}^l)/2$. Update $c_l = c_l + 1$. Go back to STEP 4.

2.2 Global Fuzzy Transition States

The set of all the intermediate states through which the set of centroids crosses constitutes a complete transition from the initial state $\mathbf{V}^0$ to the final state $\mathbf{V}^k$,

in which the number of centroids c_k is equal to the number of points n and at the same time any centroid i is equal to any element i, accomplishing this condition

$$\mathbf{v}_i^k = \mathbf{x}_i, \quad 1 \leq i \leq n. \tag{4}$$

We denote $\mathbf{v}_i^l$ as a musical element (note, chord or rhythm element) in the l-th iteration, $0 \leq l \leq k$, $1 \leq i \leq c_l$. A state in the l-th iteration is given by the sequence $\mathbf{V}^l = \{\mathbf{v}_1^l, \mathbf{v}_2^l, \ldots, \mathbf{v}_{c_l}^l\}$. We asume that $\mathbf{V}^0 = \{\mathbf{v}_1^0, \mathbf{v}_2^0, \ldots, \mathbf{v}_{c_0}^0\}$ is the initial state. The set of all the iterated states is given by a Global Fuzzy Transition.

Definition 4. *A Global Fuzzy Transition* $\widehat{\mathscr{T}}$ *from initial centroids* $\boldsymbol{V}^0$ *to the final centroids* $\boldsymbol{V}^k$ *is defined by the sequence*

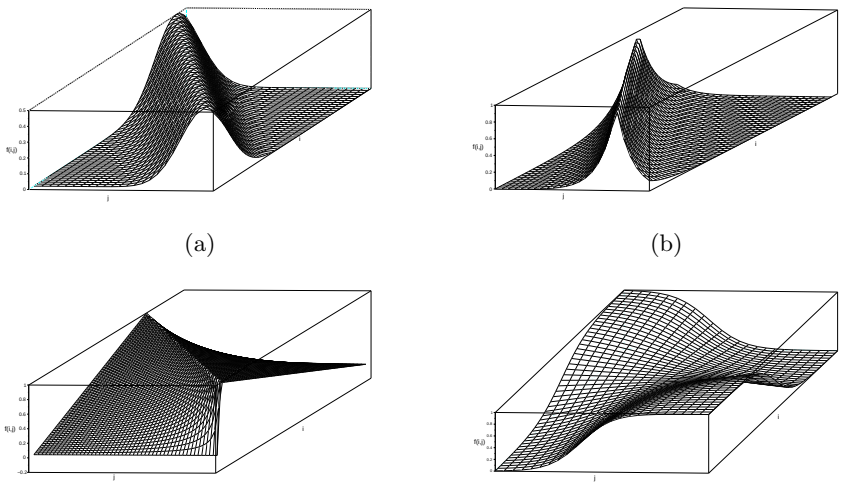
$$\widehat{\mathscr{T}} = \{\boldsymbol{V}^0, \boldsymbol{V}^1, \boldsymbol{V}^2, \ldots, \boldsymbol{V}^k\} = \{\boldsymbol{V}^l\}_{l=0}^k. \tag{5}$$

2.3 Neighbourhood and distance functions

Neighbourhood functions are the key point in algorithm FCM that allow to introduce the order of a sequence in the process of clustering. They set the weight that every point of the data set will be given when comparing it tho each point of the centroids set. In this way, if an appropriate function is defined, we can easily relate the first elements of the data set to the first elements of the centroids set with high weight values, and at the same time eliminate the relation of the first elements of the data set with the last elements of the centroids set just setting zero in this case to the weigh values. In Table 1 we show the more usual neighbourhood functions, where n, c are number of elements of the data set sequence and centroids sequence, respectively.

Table 1. Usual neighbourhood functions.

Name	Formula
1. Gaussian	$f(i,j) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left[\frac{1}{2\sigma^2}\left(i - \frac{(n-1)}{(c-1)}j\right)^2\right]}$
2. Triangular	$f(i,j) = \begin{cases} \frac{i}{\mu}, & \text{if} 0 \leq i \leq \mu \\ 1 - \frac{(i-\mu)}{(n-\mu-1)}, & \text{if } \mu < i \leq n-1 \end{cases}$ where $\mu = \frac{(n-1)}{(c-1)}j$
3. Exponential	$f(i,j) = e^{-\frac{1}{\tau}\left\lvert i - \frac{(n-1)}{(c-1)}j\right\rvert}$
4. Sigmoidal	$f(i,j) = \frac{1}{1+e^{[a\cdot b(i-c)]}}$ where $\begin{array}{c} a = -2j/(c-1)+1 \\ b = \tau/(n-1) \\ c = (n-1)/2 \end{array}$
5. Discrete	$f(i,j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$



(a) (b) (c) (d)

Fig. 1. Neighbourhood functions used in Mercury: (a) Gaussian, (b) Exponential, (c) Triangular, (d) Sigmoidal.

Table 2. Several distance functions implemented in Mercury.

Distance function	Expression
1. Euclidean	$d_{\text{euc}}(\mathbf{x},\mathbf{y}) = \left[\sum_{k=1}^{q}(x_k - y_k)^2\right]^{\frac{1}{2}}$
2. Average Euclidean	$d_{\text{euc}}(\mathbf{x},\mathbf{y}) = \left[\frac{1}{q}\sum_{k=1}^{q}(x_k - y_k)^2\right]^{\frac{1}{2}}$
3. Manhattan	$d_{\text{man}}(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{q}\lvert x_k - y_k\rvert$
4. Minkowski	$d_{\text{min}}(\mathbf{x},\mathbf{y}) = \left[\sum_{k=1}^{q}\lvert x_k - y_k\rvert^r\right]^{\frac{1}{r}}, \quad r \geq 1$
5. Chebyshev	$d_{\text{max}}(\mathbf{x},\mathbf{y}) = \max_{k=1}^{q}\lvert x_k - y_k\rvert$
6. Chord	$d_{\text{chord}}(\mathbf{x},\mathbf{y}) = \left[2 - 2\frac{\sum_{k=1}^{q} x_k y_k}{\lVert\mathbf{x}\rVert_2\lVert\mathbf{y}\rVert_2}\right]^{\frac{1}{2}}$
7. Geodesic	$d_{\text{geo}}(\mathbf{x},\mathbf{y}) = \arccos\left(1 - \frac{d_{chord}(\mathbf{x},\mathbf{y})}{2}\right)$
8. Canberra metric	$d_{\text{can}}(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{q}\frac{\lvert x_k - y_k\rvert}{\lvert x_k\rvert + \lvert y_k\rvert}$
9. Divergence coefficient	$d_{\text{div}}(\mathbf{x},\mathbf{y}) = \left[\frac{1}{q}\sum_{k=1}^{q}\left(\frac{x_k - y_k}{x_k + y_k}\right)^2\right]^{\frac{1}{2}}$
10. Discrete metric	$d_{\text{dis}}(\mathbf{x},\mathbf{y}) = \begin{cases} 1, & \text{if } \mathbf{x} = \mathbf{y} \\ 0, & \text{if } \mathbf{x} \neq \mathbf{y} \end{cases}$

Mercury allows the user to select one among several distance functions in order to proceed with the calculations. Due to their specific definition, each function returns very different computational results, providing a high degree of musical variety. The distance functions [7] that have been implemented are shown in Table 2.

2.4 The Quantization Process

It is necessary to establish a criterium for determining the equivalence between the real numbers generated by the CFT-Algorithm and the symbolic music notation. In the case of the pitch attribute, the notes are chosen enharmonically by means of selecting the closer integer MIDI pitch value. In an similar way the intensity, represented musically with the following symbols of *ppp*, *pp*, *p*, *mp*, *mf*, *f*, *ff*, and *fff*, will be approximated to the closest MIDI velocity value of the previous symbols (see [17]), according to the next table:

Table 3. MIDI velocity values for the dynamic symbols used in musical notation.

Dynamic	Velocity	Dynamic	Velocity
pppp	8	*mf*	64
ppp	20	*f*	80
pp	31	*ff*	96
p	42	*fff*	112
mp	53	*ffff*	127

In the case of the duration coefficient, a wide variety of possible symbolic rhythmical notations exist, whose duration coefficient is exactly the same. In order to approximate the numerical result obtained for the duration attribute to its closer symbolic notation, the user has to specify which of the possible combinations of durations (whole, black, quaver, semiquaver, etc.), number of tied notes, number of dots and kinds of tuplets (3:2, 5:4, 7:4, etc.) are possible. To achieve this, Mercury offers the *Quantization Settings Menu*. Figure 2 shows an example of an user-defined quantization:

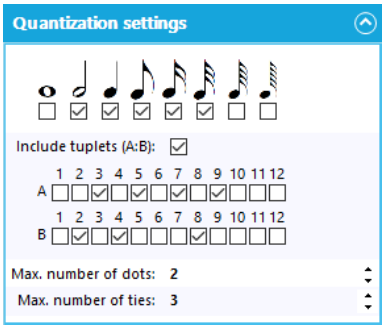


Fig. 2. Example of setting up for quantization.

Once the setting has been established, Mercury calculates the duration coefficient for all the possible allowed combinations, removing the repeated ones, sorting the final results by an increasing order and storing them in RAM memory just in case the quantization settings do not change in following calculations. When the CFT-algorithm has finished and the program needs to search for the most appropriate symbolic rhythmical notation, it will simply choose the closest one in the list to the numerical value of duration coefficient generated by CFT.

3 Structure of the program

Mercury is composed of four basic modules: the calculation kernel, the input/output module, the user interface module and the graphical music notation library. All these modules work in an independent way and communicate themselves by a set of domain classes capable to represent and share all the symbolic information that music needs. The calculation kernel is written in object-oriented programming language *C#* and implements FCM, FOCM and CFT algorithms. It also contains an architecture of classes to map the symbolic music notation objects that can represent melodies, harmonies, chords, etc. into matrices of real numbers with which the algorithm will work. In addition, each neighbourhood and distance functions are represented by classes that perform the required calculations within the process of the algorithm. Mercury receives as an input scores in format *musicMXL* that can be easily generated with standard music edition software. The program parses the information included in the tagged XML format and translates it into its musical object system. In this version, melody is restricted to monophonic lines written only in one staff, but there is no other restriction in terms of duration, rhythmical patterns, articulation, dynamics, silences, etc. Mercury is able to work and create transitions between two melodies of any kind and length. In the case of the harmonic transition, the only restriction is that initial and final harmonies must have the same number of voices, without any limit to the number of voices and length of any of the harmonic sequences. Finally, in the case of loading rhythmical structures, they should be written in only one voice and one staff; the program in that case will omit any information regarding the pitch.

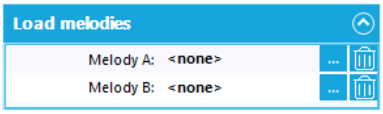


Fig. 3. Menu for loading melodies.

All the results generated by Mercury can be easily exported to a MusicXML file, so the user/composer has the possibility to use then as a musical material for the composition process. Furthermore, the program allows the user to listen, directly from the program by means of the MIDI playing, the melodies,

chords or rhythms generated. With this functionality the user can evaluate the musical interest of the results and choose those that fit better with the desired compositive criteria.

3.1 The User Interface

The user interface has been developed under VB.NET, using the free controls library *Syncfusion* for .NET programming, running at the moment under Windows operating system with .NET framework 4.5 technology. The main form allows the user to create, edit and save projects. Each project includes as many tabs as the user would need to experiment with the possibilities that Mercury offers. The music information can be easily edited or moved between tabs so the work of composer becomes flexible and powerful. In the *Play menu*, the user can specify the MIDI settings for playing the music in real time from the *Microsoft GS Wavetable Synth* or any other MIDI device connected to the computer like a MIDI digital piano.

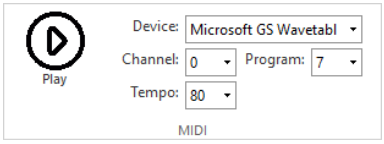


Fig. 4. Menu for MIDI settings: device, channel, program, tempo.

The CFT algorithm requires the user to establish several parameters that are direct and strongly related with the musical results obtained, and the *Fuzzy Settings Control* allows to control them. As is explained in [2, 11], the higher the *Fuzzy Coefficient* λ is, the fuzzier the process of clustering will be, so the individual elements of the intermediate steps will show more tendency to share their positions. The *Stop Criteria* is an arbitrary value that will stop the convergence process. High values will make stop the process before the goal has been accomplished, on the other hand, very low values will make the process slow and will increase the requirements of memory.

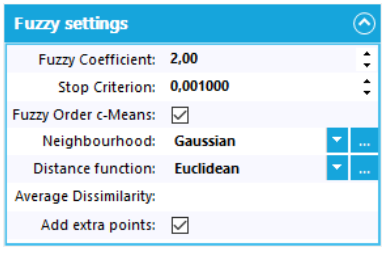


Fig. 5. Menu for setting the CFT parameters.

The menu brings also to the user the possibility to choose among several neighbourhood functions (three from the gaussian family, three from the exponential family, four triangular, one rectangular, one trapezoidal and one sigmoidal), and also the possibility to choose among several distance functions (Euclidean, Average Euclidean, Manhattan, Average Manhattan, Minkowsky, Chebyshev, Chord, Geodesic, Canberra metric, Divergence Index and Discrete metric). Each one of these neighbourhood or distance functions will provide very different intermediate states. The combination of all possible values for Fuzzy Coefficient, all possible neighbourhood functions and all possible distance functions brings to the user a huge amount of possibilities to explore in order to search for the desired musical material.

3.2 The graphic music notation library

The library used for displaying the music is based on *PSALMControlLibrary* (Polish System for Archivising Music Control Library), an open-source control developed for *.NET Framework* in 2010 by *Jacek Salamon*. It has been strongly modified to fit the requirements of our program. In Figure 6, three examples of a melody, a harmonic sequence and a rhythmical pattern are displayed.

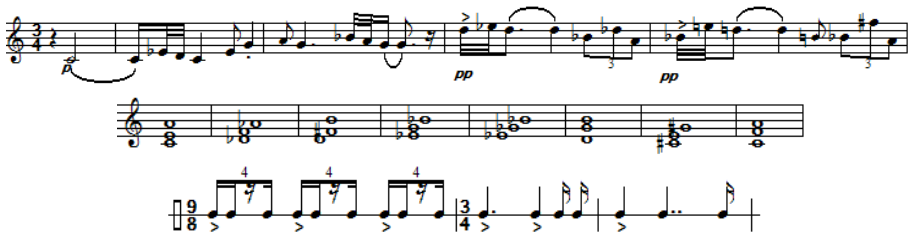


Fig. 6. A melody, a chord sequence and a rhythmical pattern displayed in the graphic music notation library of Mercury.

4 Computational examples

In this section we present three simple examples that illustrate the applicability of the method in three different scenarios: melody, harmony and rhythm. Each example is configured with different fuzzy coefficient λ, distance function and neighbourhood function, showing some of the intermediate musical states generated by the algorithm in each transition from element B to the objective element A. Each intermediate state is notated with its corresponding step number on the left. Notice that at the last state of each transition the element A is reached. The examples have been run on Mercury and finally exported in MusicXML format to a score edition software.

Example 1. Intermediate states of the complete fuzzy transition calculated with CFT between two birdsongs transcribed by composer *Olivier Messiaen* [14]. The fuzzy settings are $\lambda = 30$, exponential neighbourhood and chord metric.

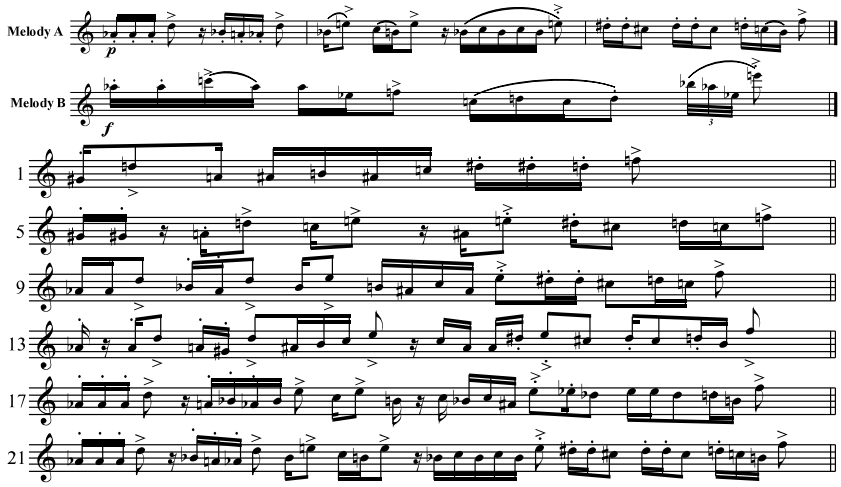


Fig. 7. Initial state and iterations number 1, 5, 9, 13, 17 and 21.

Example 2. Intermediate states of the complete fuzzy transition calculated with CFT between two harmonic sequences **A** and **B**. The fuzzy settings are $\lambda = 1.5$, gaussian neighbourhood and Manhattan distance.

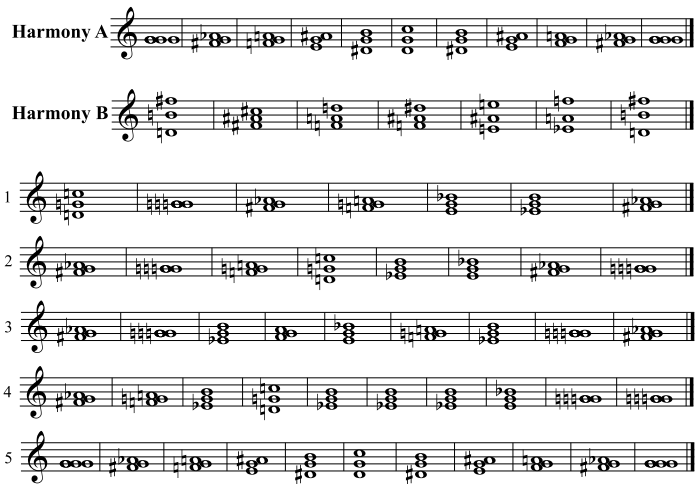


Fig. 8. Global harmonic transition from B to A.

Example 3. Intermediate states of the complete fuzzy transition calculated with CFT between harmonies *caccarî* and *simhavikrîdita*, belonging to 120 deçi-tâlas [8]. The fuzzy settings are $\lambda = 5$, exponential neighbourhood and chord metric.

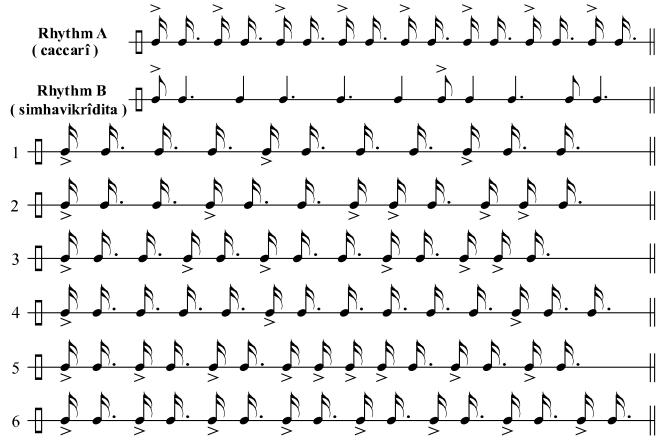


Fig. 9. Global rhythmical transition from *simhavikrîdita* to *caccarî*.

5 Conclusions

In this paper we have explored the possibilities of our software Mercury® in the field of the symbolic computer-assisted composition. It provides the composer with new and powerful tools to create transitions or variations of the musical material in terms of melody, harmony or rhythm, widening the number of possible creative options that may fit with the aesthetic requirements of his artistic criteria. The music generated with the software can be easily exported to the standard format MusicXML and afterwards used with any software or score edition. The limitations of this first version of the software include the restriction to monophonic melodic lines and the use of only one staff for the music material used as input data. Besides, the transitions between harmonies work only if all chords belonging to the initial and final ones have the same number of notes. Future versions of the software will improve this limitations and implement the CFT-algorithm to other characteristics of music: for example, is it possible to create transitions between the timbre of two sounds (as is shown in [12]) offering promising results for sound synthesis in the field of electroacoustics or spectral music. Is it possible, as well, to create transitions between two different tuning systems defined in [10]. The present version of Mercury works only under *Windows* operative system, following developments will include *iOS* and *Android* distributions. The software is currently available at `www.futurewebofmercury.com`.

References

1. Ariza, C.: Two pioneering projects from the early history of computer-aided algorithmic composition. Computer Music Journal **35**(3), 40–56 (2011)
2. Bezdek, J. C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press, New York (1981)
3. Biles, J.: GenJam: A genetic algorithm for generating jazz solos. In: Proceedings of International Computer Music Conference ICMC 1994, pp. 131–137. Michigan Publishing (1994).
4. Buteau, C.: Melodic Clustering within Topological Spaces of Schumann's Träumerei. In: Proceedings of International Computer Music Conference ICMC 2006, pp. 104–110. Tulane University, New Orleans (2006).
5. Dodge, C.: Profile: A Musical Fractal. Computer Music Journal **12**(3), 10–14 (1988)
6. Farbood, M., Schoner, B.: Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains. In: Proceedings of International Computer Music Conference ICMC 2001, pp. 1–4. Michigan Publishing (2001).
7. Gan, G., Ma, C., and Wu, J.: Data clustering: theory, algorithms, and applications. SIAM, Philadelphia (2007)
8. Johnson, R.: Messiaen. University of California Press, Paris (1989)
9. Krzyzaniak, M.: Interactive Learning of Timbral Rhythms for Percussion Robots. Computer Music Journal **42**(02), 35–51 (2018)
10. Liern, V.: Fuzzy tuning systems: the mathematics of musicians. Fuzzy sets and systems **150**(1), 35–52 (2005)
11. Martínez, 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, pp. 279–290. Springer, Heidelberg (2016).
12. Martínez, B., Liern, V.: Comparación y transiciones espectrales mediante el algoritmo fuzzy c-means. In: Tecniacústica 2017: 48° Congreso Español de Acústica; Encuentro Ibérico de Acústica; European Symposium on Underwater Acoustics Applications; European Symposium on Sustainable Building Acoustics:A Coruña 3-6 Octubre 2017, pp. 1169–1175. Sociedad Española de Acústica (2017).
13. Miranda, E.: Cellular automata music: An interdisciplinary project. Journal of New Music Research **22**(1), 3–21 (1993)
14. Messiaen, O.: The Technique of my Musical Language. Alphonse Leduc, Paris (1956)
15. Nierhaus, G.: Algorithmic composition: paradigms of automated music generation. Springer Science and Business Media, New York (2009)
16. Roads, C.: Composing Grammars. In: Proceedings of International Computer Music Conference ICMC 1977, pp. 54–132. University of California, San Diego (1977).
17. Selfridge-Field, E.: Beyond MIDI: the handbook of musical codes, MIT Press, Cambridge, Massachussets (1997)
18. Tompkins, D.: A Cluster Analysis for Mode Identification in Early Music Genres. In: Agustín-Aquino O., Lluis-Puebla E., Montiel M. (eds) Mathematics and Computation in Music. MCM 2017, pp. 313–323.Springer, Heidelberg (2016).