2019

Intelligent Technique for Automating the Conversion between Major and Minor Melodies

Nermin N. J. Siphocly
Ain Shams University, nermine.naguib@gmail.com

El-Sayed M. El-Horbaty
Ain Shams University, sayed.horbaty@yahoo.com

abd el-badea Mohamed salem prof
ain shams university, absalem@cis.asu.edu.eg

Follow this and additional works at: <https://digitalcommons.aaru.edu.jo/fcij>

Part of the [Computer Engineering Commons](#)

Recommended Citation

Siphocly, Nermin N. J.; El-Horbaty, El-Sayed M.; and salem, abd el-badea Mohamed prof (2019) "Intelligent Technique for Automating the Conversion between Major and Minor Melodies," *Future Computing and Informatics Journal*: Vol. 4: Iss. 2, Article 2.

DOI: <http://doi.org/10.54623/fue.fcij.4.2.2>

Available at: <https://digitalcommons.aaru.edu.jo/fcij/vol4/iss2/2>

This Article is brought to you for free and open access by Arab Journals Platform. It has been accepted for inclusion in Future Computing and Informatics Journal by an authorized editor. The journal is hosted on [Digital Commons](#), an Elsevier platform. For more information, please contact rakan@aarj.edu.jo, marah@aarj.edu.jo, u.murad@aarj.edu.jo.

Future Computing and Informatics Journal

Volume 4
Issue 2 (2019) 2019 Volume 4, Issue 2

Article 2

2019

Intelligent Technique for Automating the Conversion between Major and Minor Melodies

Nermin N. J. Siphocly
Ain Shams University, nermine.naguib@gmail.com

El-Sayed M. El-Horbaty
Ain Shams University, sayed.horbaty@yahoo.com

abd el-badea Mohamed salem prof
ain shams university, absalem@cis.asu.edu.eg

Follow this and additional works at: <https://digitalcommons.aaru.edu.jo/fcij>

Part of the [Computer Engineering Commons](#)

Recommended Citation

Siphocly, Nermin N. J.; El-Horbaty, El-Sayed M.; and salem, abd el-badea Mohamed prof (2019) "Intelligent Technique for Automating the Conversion between Major and Minor Melodies," *Future Computing and Informatics Journal*: Vol. 4 : Iss. 2 , Article 2.

Available at: <https://digitalcommons.aaru.edu.jo/fcij/vol4/iss2/2>

This Article is brought to you for free and open access by Arab Journals Platform. It has been accepted for inclusion in Future Computing and Informatics Journal by an authorized editor. The journal is hosted on [Digital Commons](#), an Elsevier platform. For more information, please contact rakan@aarj.edu.jo, marah@aarj.edu.jo, u.murad@aarj.edu.jo.

Intelligent Technique for Automating the Conversion between Major and Minor Melodies

Nermin Naguib J. Siphocly^{1,a}, El-Sayed M. El-Horbaty^{1,b},
Abdel-Badeeh M. Salem^{1,c}

Faculty of Computer & Information Sciences Ain Shams University, Cairo, Egypt.

^anermine.naguib@cis.asu.edu.eg, ^bshorbaty@cis.asu.edu.eg,

^cabsalem@cis.asu.edu.eg

ABSTRACT

Nowadays, computers are extremely beneficial to music composers. Computer music generation tools are developed for aiding composers in producing satisfying musical pieces. The automation of music composition tasks is a challenging research point, specially to the field of Artificial Intelligence. Converting melodies that are played on a major scale to minor (or vice versa) is interesting to both composers and music listeners. Newly converted melodies of famous songs, either from major to minor or the opposite, are becoming blockbusters on the social media. In this paper we propose an intelligent method for automating the conversion between major and minor melodies using Artificial Intelligence techniques. We run our experiments on melodies in the MIDI format which is a standard music format enabling the communication between computers and various musical devices. We also propose a smart method for musical scale detection for the input melodies. Scale detection is a critical step for correctly converting between major and minor melodies. Additionally, this step is also important as a pre-processing step in various other music retrieval or transformation applications.

Keywords: Machine Learning, Algorithmic Composition Artificial Intelligence, Computer Music

1. Introduction

Since the early days of computer invention, musicians were interested in utilizing its computational powers to generate music. Along the years, as the computers' capabilities enhanced, many musical

applications have been developed. Ramon de Mántaras (Mántaras, 2006), classified music generation applications into three main categories;

- Improvisation: which means that computers play along in harmony with human music

players, on the spot. Examples of music improvisation applications can be found in (Déguernel, Vincent, & Assayag, 2018), (Nika, Bouche, Bresson, Chemillier, & Assayag, 2015) and (Dubnov, & Assayag, 2012).

- Composition: which means that computers aid either partially or fully in composing new musical pieces. Examples of music composition applications are found in (Nadeem, Tagle, & Sitsabesan, 2019), (Yamada, Kitahara, Arie, & Ogata, 2018) and (Ponce de León, Iñesta, Calvo-Zaragoza, & Rizo, 2016).
- Expressiveness: which means that computers simulate the personal touch of human music players. Examples of expressive performance synthesis application are found in (Della Ventura, 2018) and (Giraldo, & Ramírez, 2016).

In our work we focus on music composition category. A composed musical piece has three main components; melody (the main sequence of notes forming it), accompaniment (the music accompanying the main melody and in harmony with it), and the rhythm (which is the beats of the musical piece that defines the speed and style). In this work we are concerned with the melody generation automation.

Teaching computers to compose melody that sounds correct or perfect, is a challenging problem, specially to Artificial Intelligence field. Achieving this learning requires feeding computers with musical rules on one hand and training them to extract musical rules from music datasets on the other hand. In this research we aim to utilize Artificial Intelligence algorithms in developing

intelligent software for music composition through converting between major and minor scales. What motivated us for this work is that converting melodies that are played on a major scale to minor (or vice versa) is interesting to both composers and music listeners. Moreover, they are becoming blockbusters on the social media; it has become very common to find videos on YouTube of major or minor versions of popular songs.

This paper suggests two methods for the automation of major/minor conversion of a given melody the first adopts rule-based and the second adopts case-based reasoning (CBR) algorithms. We highlight the main idea of the major/minor conversion application using each algorithm. We then give a detailed overview of the suggested algorithms. We shed the light on the strengths and weaknesses of each comparing between them empirically. Furthermore, we discuss our findings and experimental results; illustrating the implementation tools used.

This paper is organized as follows; Section 2 states the recent work related to our research. Section 3 gives a brief musical background for musical concepts related to our topic. Section 4 discusses the rule-based approach for major/minor conversion. Section 5 illustrates the case-based approach for major/minor conversion. Section 6 highlights the strengths and weaknesses of both methods. Section 7 presents our results. Section 8 illustrates the implementation. Finally, Section 9 utters the conclusion and future work.

2. Related Work

Recently, there have been several research attempts in the field of computer music composition. Rule-based systems have been adopted in the field of music composition because they are the best to formulate music theory rules. Applications of rule-based systems in music composition include Navarro et al. work (Navarro-Cáceres, Caetano, Bernardes, Castro, & Corchado, 2015); they encoded chord generation rules in their system. However, this was just a part of the complete system that relies on Artificial Immune System (AIS) for predicting chord sequences. The rules were encoded inside the penalty function of the AIS. On the other hand, Aguilera et al. (Aguilera, Galán, Madrid, Martínez, Padilla, & Rodríguez, 2010) developed a system for counterpoint accompaniments generation through coding counterpoint music rules with the help of probabilistic logic. Hadimlioglu et al. (Hadimlioglu, & King, 2018) developed Intelligent Transition Generator (ITG) that generates a smooth transition between two input compositions. The properties of each of the input compositions are analyzed in order to extract appropriate information about note durations, time signatures, musical style and chord similarities. The extracted information from the input compositions are then used to determine appropriate rules for creating the transition.

Since music generation systems' main purpose is to learn from human composers' experience; hence, it has been inevitable to apply CBR for achieving this purpose. Navarro et al. (Navarro-Cáceres, Rodríguez, Milla, Pérez-Lancho, & Corchado, 2017) combined case-

based with Markov models to generate melody by predicting the probability of a new note from the last previously generated note in that melody. Their system meant to imitate a specific musical style in melody generation. Thus, the case base stored melodies' past solutions which were retrieved according to criteria and choices entered by the user.

3. Brief Musical Background

On a full-size piano, there is a total of eighty-eight keys, however only twelve different notes exist that repeat from low to high tones. Each set of twelve consecutive notes forms what is called an octave. Musical notes are given alphabetical letters from 'A' to 'G' such that 'C' is the 'Do'. Figure 1 shows piano keys that form one octave. Black key notes are called flats or sharps. The distance between any two consecutive notes is called *half tone*. Music notes are grouped harmonically into scales that consist of seven notes each. There are two main types of scales:

- Major scales are the scales that sound positive – they are used when a composer desires to portray happiness. There are twelve major scales.
- Minor Scales (harmonic minor) are the scales that sound more dramatic or nostalgic. Each major scale has a corresponding minor scale; thus, there exists twelve harmonic minor scales.

Fig. 1 – Piano

Fig. 2 – (a) C Major Scale Notes, (b) C Minor Scale

Figure 2 shows an example of major and minor scales of the note ‘C’; such that the notes forming the C major scale are {C, D, E, F, G, A, B} and those forming the C minor scale are {C, D, E flat, F, G, A flat, B}.

4. Rule-based Major/Minor Conversion

A rule-based music system is that in which music theory is formulated into production rules.

4.1. Application of rule-based system in major/minor conversion

A rule-based music system for converting simple melodies from major to minor scale relies on the following rule: “flat” the third and sixth notes on the scale (decrease them by a half tone). The opposite conversion is done through “sharpening” the same notes (increasing them by a half note). Thus, it is of great importance to know the scale of the melody before applying the conversion. Detecting the scale of a given melody is important for a wide range of music retrieval or transformation applications as a preprocessing step.

The main steps of the rule-based conversion are:

1. Input the melody to convert along with a flag indicating whether it is in a major or minor scale
2. If in a major scale
 - a) Detect its specific scale out of twelve possible major scales
 - b) Apply flattening rules
3. Else (if in a minor scale)
 - a) Detect its specific scale out of twelve possible minor scales
 - b) Apply sharpening rules

For scale detection, we keep in memory all the twenty-four different scales. We start by extracting all the unique notes in each given melody, removing any redundancy. After extracting the unique notes forming the melody, we normalize them to be in the first octave. We match the extracted normalized notes with the notes of the twelve relevant scales in the memory giving a score for each scale representing the size of the intersection. This process is mathematically described as follows:

- Let $N = \{n_1, n_2, \dots, n_L\}$ be the set of unique notes of the input melody, where n_k is the k^{th} note in the set such that $1 \leq k \leq L$ and L is the number of unique notes in the melody.
- Let S be the set of all the 12 relevant scales where $S = \{s_1, s_2, \dots, s_{12}\}$ where s_i is one of the scales such that $1 \leq i \leq 12$.
- Let s_i be the set of notes forming the scale. $s_i = \{x_{i1}, x_{i2}, \dots, x_{i7}\}$ where x_{ij} is the j^{th} note in the i^{th} scale.
- We define the matching score between N and s_i as:

$$\text{Score}(N, s_i) = |N \cap s_i|$$
- We assume that the scale with the highest score is the melody’s scale.

For the conversion, we refer to the detected scale and identify its third and sixth notes. If the detected scale is major, we flatten all the occurrences of the identified notes in the input melody to convert it to minor. And the opposite applies to minor scales.

4.2. The algorithm

Algorithm 1 illustrates the rule-based algorithm for major/minor conversion which takes as an input the “*input_melody*” that needs to be converted.

Algorithm 1 – Rule-based major/minor conversion algorithm.**Inputs:**

- *input_melody*: list of all notes in the given melody
- *is_major*: Boolean flag indicating the conversion direction (major to minor if flag is true)

Output:

- *output_melody*: list of all output notes corresponding to the given melody after conversion

Algorithm:

major_scales = list of all music major scales

minor_scales = list of all music minor scales

//each scale is a list of notes belonging //to that scale

unique_notes = empty list

FOR {each note *n* in *input_melody*}

IF {*n* NOT in *unique_notes*}

 Append *n* to *unique_notes*

ENDIF

ENDFOR

Initialize the scales scores lists *major_scales_scores*, *minor_scales_scores* to zeros.

FOR {each unique note *un* in *unique_notes*}

IF {*is_major* is TRUE}

FOR {each major scale *major_s* in *major_scales*}

IF {*un* in *major_s*}

 Increment

major_scales_scores[*major_s*]

ENDIF

ENDFOR

max_index = index of the maximum element in

major_scales_scores

scale = *major_scales*[*max_index*]

ELSE

 //Do the same but for minor

 // scales

n1 = *scale*[2], *n2* = *scale*[5]

FOR {each note *n* in *input_melody*}

IF { *n* == *n1* OR *n* == *n2*}

IF {*is_major* is TRUE}

 Decrement *n*

ELSE

 Increment *n*

ENDIF

ENDIF

ENDFOR

output_melody = *input_melody*

the melody. We must also provide as an input a flag, “*is_major*”, indicating if the input melody is in a major or minor scale. The lists of notes belonging to all the major scales and those of minor ones are kept in memory in “*major_scales*” and “*minor_scales*”. The algorithm starts by extracting the “*unique_notes*” from the input melody ignoring any repetitions. Scale detection is performed such that, according to *is_major*, each scale in the “*major_scales*” or “*minor_scales*” gets a score relative to how many notes of the “*unique_notes*” it contains. This is done by looping on the notes in the “*unique_notes*” and incrementing the score of the scales containing it. The scale having the maximum score is considered the scale that the input melody follows. Accordingly, the algorithm fetches the third and sixth note from the detected scale and checks for all their occurrences in the input melody. Whenever these fetched notes are found in the melody, they are incremented by a half tone (in case of major conversion) or decremented by a half tone (in case of minor conversion). Hence, the “*output_melody*” is generated.

5. Case-based Major/Minor conversion

Our second proposed method for major/minor conversion of melodies relies on case-based reasoning. CBR is the use of previous experiences for solving current problems.

Fig. 3 – CBR Cycle

5.1. CBR overview

In CBR, successful solutions from past problems are retrieved and reused. A ‘case base’ in CBR system is where information about previous problems is kept. The description of a previously solved problem along with its solution is kept in what is called a ‘case’. CBR cycle has three main steps which are: case retrieval, adaptation, and storing. Figure 3 shows a diagram of the CBR cycle where, first, a new problem is input to the system. Consequently, cases that are similar to that given problem are retrieved from the case base. The best matching case (the closest to the input problem), along with its solution are adapted to meet the problem demands. Finally, the newly solved problem is stored in the case base to be used in the future.

5.2. Application of CBR in Major/Minor Conversion

In our application, cases are represented as pieces of melodies along with their corresponding converted melodies. In order to prepare the case base; we scan the melodies in the training dataset and their corresponding converted melodies, simultaneously, by a window of a fixed size (for example seven notes wide). For each set of window-sized notes we create a new case having the piece of melody in addition to its corresponding piece of converted melody. As a pre-processing step for cases retrieval, we divide the input melody into chunks of size equal to the window size used in the case base preparation.

5.2.1. Nearest neighbor approach

For the case retrieval we went through two different approaches, the first is the traditional nearest neighbour method that calculates the distance between each note in the input melody and its corresponding note in every

case in the case base, then choosing the case of the minimum distance. Adaptation of the case solution in the nearest neighbour approach reflect the relation between the input and the matching case’s melody. For example, if only the second note in the matching case melody is less than that in the input melody by a half tone, then the second note in the case solution (converted melody) will be increased by a half tone; and this will be the new problem’s solution. The nearest neighbour approach proved to be problematic because it matches with cases that follow different music scales other than that followed by the input melody. This problem happens due to the limited distances between all the twelve distinct notes.

5.2.2. Our proposed case retrieval approach

We retrieve cases according to three criteria:

- We retrieve the cases that include the largest number of notes from the input melody, thus, we match with cases that are more conforming to the same scale of the input melody.
- Of the retrieved cases, we choose the cases whose notes are likewise included in the input case.
- The last stage of case retrieval includes comparing the musical notes’ histograms of each of the retrieved cases from the previous step with that of the input case.

The adaptation phase in our proposed method is performed through “re-ordering” the case solution. For each input melody note, the corresponding converted melody is fetched from the case solution of the same note. Figure 4 shows a visual example for case adaptation in which the input melody is formed of the notes {‘A’, ‘C’, ‘C’, ‘B’, ‘A’, ‘D’, ‘D’} and the best matching retrieved case has the melody notes {‘C’, ‘A’, ‘C’, ‘B’, ‘B’, ‘D’,

'G'}). Each note in the input melody is matched with all the notes in the retrieved case melody. For example, the first note in the input melody 'A' is found to be the second note in the matching retrieved case whose corresponding converted melody is 'A_b' thus we set the first note of the converted melody solution to 'A_b' and so forth.

Fig. 4 – Case Adaptation

5.3. The Algorithm

Algorithm 2 shows our proposed algorithm for major/minor conversion. It is formed of four parts. The first part describes the general algorithm and the rest of the parts describe specific functions called in the main algorithm. The algorithm takes as an input the melody desired to be converted and outputs the

converted melody. The algorithm does not require the user to declare if the input melody is major or minor as it automatically converts major to minor and vice versa. The case base consists of cases of pairs of the melody pieces with their corresponding converted pieces, i.e. the algorithm is trained by both major and minor melody pieces besides their corresponding conversions. The input melody is split to melody pieces and converted one by one.

Algorithm 2 – Case-based major/minor conversion algorithm.

Input:

- *input_melody*: list of all notes in the given melody

Output:

- *output_melody*: list of all output notes corresponding to the given melody after conversion

Algorithm:

```

case_base = list of all case base pairs
//such that each pair holds
a //main_melody and a
corresponding //conv_melody

output_melody = empty list
of notes
test_cases =
split(input_melody)
FOR {each test_case in
test_cases}
    matching_cases1 =
match1(test_case, case_base)
    matching_cases2 =
match2(test_case,
matching_cases1)
    hist_nearest_neighbor =
histogram_nn(test_case,
matching_cases2)
    FOR {each note n in
test_case}
        Append
hist_nearest_neighbor.
conv_melody[n] to
output_melody
    ENDFOR
ENDFOR

```

For each test case the function “match1” is calls which takes as parameters the test case and the case base. “match1” returns the cases from the case base that contains most/all the notes of the test case. It gives a score to each case (pair) in the case base by counting the number of notes of its main melod .that exist in the test case. The matching cases returned from “match1” are sent as parameters to the function call “match2” in addition to the test case. “match2” returns the cases from the input cases whose notes are contained in the test case. “match2” works in the same manner as “match1” however counts the notes of the test case that are contained in each case of the input cases.

```

Function match1
Pass In:
- test_case: the input test case
  to be matched
- case_base: list of melody pairs
Returns:
- matches: list of maximum
  matching cases

//Initialize the pairs scores to
//zeros
init_to_zeros(pairs_scores)
FOR {each pair in case_base}
    pairs_scores[pair]=count
    test_case notes
    in pair.main_melody
ENDFOR
max_count = max(pair_scores)
matches = empty list
FOR {each pair in case_base}
    IF(pairs_scores[pair] =
max_count)
        Append pair to matches
    ENDIF
ENDFOR
RETURN matches
    
```

Function match2

Pass in:

- test_case: the input test case
to be matched
- input_cases: list of melody
pairs

Returns:

- matches: list of matching cases

```

//Initialize the pairs scores to //zeros
init_to_zeros(pairs_scores)
FOR {each pair in inputs_cases}
    pairs_scores[pair]=count
    pair.main_melody notes in test_case
ENDFOR
max_count = max(pair_scores)
matches = empty list
FOR {each pair in input_cases}
    IF(pairs_scores[pair] =
max_count)
        Append pair to matches
    ENDIF
ENDFOR
RETURN matches
    
```

Function histogram_nn**Pass In:**

- *test_case*: the input test case to be converted,
- *input_cases*: the matching cases from the previous matching step

Returns:

- *nearest_neighbor*: case whose histogram is the closest to that of the *test_case*

//Define and initialize test case //histogram
(dictionary of notes)

init_to_zeros(*tc_histogram*)

min_score = MAX_INT

FOR {each note *n* in *test_case*}

Increment *tc_histogram[n]*

ENDFOR

FOR {each case *c* in the

input_cases}

init_to_zeros(*ic_histogram*)

FOR {each note *n* in *c*}

Increment *ic_histogram[n]*

ENDFOR

dist =

calculate_distance(*tc_histogram*,

case "*ic_histogram*". Whenever a note occurs in the test case notes, its score is incremented in the histogram. Similarly, the notes in each input case takes a score in the "*ic_histogram*".

The distance between the two histograms is calculated as the sum of the absolute difference between each two corresponding notes scores. The case of the minimum distance is returned (the nearest neighbor case). Finally, the adaptation of the nearest neighbour case solution "*conv_melody*" is done by reordering it to match the order of occurrence in the test case. Each test case solution is concatenated to the final "*output_melody*".

6. Rule-based Versus Case-based Major/Minor Conversion

The rule-based method for major/minor melody conversion has a diminished error rate only if the scale is detected correctly. However, there are two problems with this method; the first is that in some melodies that have repeating musical phrases, the number of unique notes are not enough to successfully detect the scale (more than one scale get a maximum score – thus, confusion happens). Secondly, the rule-based method only applies for simple melodies of non-changing scale; however, if the scale changes within the same melody the rule-base method will fail to detect the melody scale in the first place.

On the contrary, the strength of CBR technique for converting between major and minor melodies lies in adapting the solution according to the input melody scale for each chunk of it, thus, the error rate is minimized. Moreover, in melodies where scale changes this technique can detect the scale change and generate the converted melody accordingly.

Accordingly, the matching cases returned from "match2" are sent as a parameter to the function call "histogram_nn" along with the test case. "histogram_nn" creates a histogram for the test case, "*tc_histogram*" which is a dictionary data structure for the twelve notes with a score corresponding to each note that is initially set to zero. Another histogram is created for the input cases that is initialized to zeros for each

Fig. 5 – (a) Rule-based vs. case-based major/minor conversion

Fig. 5 – (b) Rule-based vs. case-based Conversion Error Rate per Scale

7. Results

For experimenting our proposed methods, we created a dataset of simple melodies containing 116 midi files for various musical pieces in all the twenty-four scales. The songs were played on a Korg PA700 keyboard and recorded as midi through a midi cable. We kept thirty-three files of the dataset for testing.

We compare the rule-based and case-based conversion algorithms by calculating the number of wrong notes; i.e. the notes that were not converted correctly. Figure 5 (a) shows the comparison between the two algorithms with respect to number of wrong notes in each of the test files. Figure 5 (b) compares between the algorithms relative to the number of wrong notes detected in all the test files the belong to each musical scale. It is obvious that in both comparisons the case-based algorithm has better results, this is because when the rule-based algorithm fails to detect the correct scale, all/most of the notes are incorrectly converted. Since the case-based algorithm divides the input melody to chunks that are separately matched and converted, the error rate is much less than that in the rule-based algorithm.

8. Implementation

We fully implemented the suggested rule-based and case-based reasoning techniques from scratch using Python 3.7 (Anaconda Fig. 5 – (a) Rule-based vs. case-based Conversion Error Rate per Scale 2019.10 distribution) under Linux (Xubuntu 18.04 LTS). We used the standard MIDI format which stands for Musical Instrument Digital Interface. MIDI is a data communications protocol. MIDI data is not an audio signal. It contains instructions only, hence, leaving the creation of the actual audio signal to the receiving device. Instruction messages are transmitted in the form of binary numbers.

9. Conclusion and Future Work

In this paper we present two methods for automating the conversion between major and minor scaled melodies. The first method utilizes a rule-based algorithm, while the second adopts a case-based algorithm. The main contributions of our work are:

- We present intelligent techniques for composing music through major/minor conversion.
- We propose two techniques for handling the conversion; rule-based and case-based reasoning.
- We highlight the details of applying these techniques to the field of computer music.
- We provide two implementations for the case retrieval and adaptation phases of the CBR system.
- We propose smart methodologies for scale detection of the input melody, both in the rule-based and the case-based reasoning techniques.

The techniques proposed in this paper can be applied in wide range of computer music applications such as: music retrieval, accompaniment generation, etc. We can broaden our scope to include more types of scales such as: blues, diminished, chromatic, etc. We can include multi-track midi files in our future work. The rule-based implementation can be combined with scale detection module that can detect the change of scale within the same input melody.

Combining more techniques with both rule-based and CBR techniques is interesting to be experimented for achieving better results for more complicated melodies. Our work can be represented as a middle module of a bigger system that extract melodies from famous songs removing all its accompaniments, introduce them to our system to be converted to either major or minor, then the converted melodies would be introduced to another module that reconstructs back the harmonically suitable accompaniments.

REFERENCES

- R. L. de Mántaras, R. L. (2006). Making Music with AI: Some examples, IOS Press, pp. 90-100.
- Déguelnel, K., Vincent, E., and Assayag, G. (2018). Probabilistic factor oracles for multidimensional machine improvisation. *Computer Music Journal*, 42(02):52-66.
- Nika, J., Bouche, D., Bresson, J., Chemillier, M., and Assayag, G. (2015). Guided improvisation as dynamic calls to an offline model. In *Sound and Music Computing (SMC'15)*.
- Dubnov, S., and Assayag, G. (2012). Music design with audio oracle using information rate. In *Proceedings of the Conference on Musical Metacreation: Papers from the 2012 AIIDE Workshop*.
- Nadeem, M., Tagle, A., and Sitsabesan, S. (2019). Let's make some music. In *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1-4.
- Yamada, T., Kitahara, T., Arie, H., and Ogata, T., (2018). Four-part harmonization: Comparison of a Bayesian network and a recurrent neural network. In Mitsuko Aramaki, Matthew E. P. Davies, Richard Kronland-Martinet, and Sølvi Ystad, editors, *Music Technology with Swing*, pages 213-225, Cham. Springer International Publishing.
- León, P. J. P., Iñesta, J. M., Calvo-Zaragoza, J., and Rizo, D. (2016). Data-based melody generation through multi-objective evolutionary computation. *Journal of Mathematics and Music*, 10(2):173-192.
- Della Ventura., M. (2018). Computer system for designing musical expressiveness in an automatic music composition process. Jianyong Qiao, Xinchao Zhao, Linqiang Pan, Xingquan Zuo, Xingyi Zhang, Qingfu Zhang, and Shanguo Huang, editors, *In Bio-inspired Computing: Theories and Applications*, pages 434-443, Singapore. Springer Singapore.
- Giraldo, S., and Ramírez, R. (2016). A machine learning approach to ornamentation modeling and synthesis in jazz guitar. *Journal of Mathematics and Music*, 10(2):107-126.
- Navarro- Cáceres, M., Caetano, M., Bernardes, G., Castro, L. N., and Corchado, J. M. (2015). Automatic generation of chord progressions with an artificial immune system, C. Johnson, A. Carballal, J. Correia (Eds.), *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, Springer International Publishing, Cham, pp. 175-186.
- Aguilera, G., Galán, J. L., Madrid, R., Martínez, A. M., Padilla, Y., and Rodríguez, P. (2010). Automated generation of contrapuntal musical compositions using

- probabilistic logic in Derive. Math. Comput. Simul. 80, 6 , 1200–1211.
- Hadimlioglu, I. A., King, S. A. (2018). Automated musical transitions through rule-based synthesis using musical properties, Entertainment Computing, Volume 28, Pages 59-67, ISSN 1875-9521.
- Navarro-Cáceres, M., Rodríguez, S., Milla, D., Pérez-Lancho, B., Corchado, J. M. (2017). A user controlled system for the generation of melodies applying case based reasoning, in: D. W. Aha, J. Lieber (Eds.), Case-Based Reasoning Research and Development, Springer International Publishing, Cham, pp. 242-256.