A Genetic Algorithm for Dodecaphonic Compositions

Roberto De Prisco, Gianluca Zaccagnino, and Rocco Zaccagnino

Laboratorio di Musimatica
Dipartimento di Informatica
Università di Salerno
84084 Fisciano (SA) - Italy
http://music.dia.unisa.it

Abstract. In this paper we propose an automatic music composition system for dodecaphonic music based on a genetic algorithm.

Dodecaphonic music, introduced by A. Schoenberg, departs from the concept of tonality by considering all 12 notes equally important. Dodecaphonic compositions are constructed starting from a 12-note series, which is a sequence of all the 12 notes; the compositional process uses the starting 12-note series as a seed and builds the music creating new fragments of music obtained by transforming the seed series.

The algorithm proposed in this paper automates the compositional process taking a seed series as input and automatically creating a dodecaphonic composition. We have implemented the algorithm and we have run several tests to study its behaviour.

1 Introduction

In this paper we are interested in the design and implementation of an *algorithmic music composition system* for *dodecaphonic music*. The system must be capable of composing music without *any* human intervention.

Algorithmic music composition fascinates both computer scientists and musicians. The literature has plenty of works in this area, starting from the ILLIAC[1] suite by Hiller [7,8], and arriving to more recent efforts, like, for example, the ones by Cope [3] and Miranda [13].

Several music composition problems have been considered in the literature: 4-voice harmonizations, jazz solos, music styles reproduction, and others. To the best of our knowledge, no one has yet considered the possibility of using genetic algorithms for producing dodecaphonic compositions.

The dodecaphonic (or 12-note) technique has been introduced by Arnold Schoenberg in the 1920s: this technique departs from the concept of tonality, in which the notes of a given tonality are more important than notes outside the given tonality, and considers all the 12 notes equally important. Dodecaphonic music is built starting from a *seed series*: a particular sequence of the 12 different

[1] ILLIAC is the name of a computer built in the 50s.

C. Di Chio et al. (Eds.): EvoApplications 2011, Part II, LNCS 6625, pp. 244–253, 2011.

notes. An example of a 12-note series is: [C,D,D♯,A♯,C♯,F,B,G♯,E,A,G,F♯]. The series is not a theme but a source from which the composition is constructed. It can be transposed to begin on any of the 12 pitches, and it may appear in various derived forms.

Schoenberg dodecaphonic technique has been later generalized, by his disciples Berg and Webern and by others, to the serialism technique. In this paper however we consider only dodecaphonic music where the seed series is made up of all the 12 notes and the possible transformations are limited to certain types of series manipulation.

During the last few decades, algorithms for music composition have been proposed for several music problems. Various tools or techniques have been used: random numbers, formal grammars, cellular automata, fractals, neural networks, evolutionary algorithms, genetic music (DNA and protein-based music). The book by Miranda [12] contains a good survey of the most commonly used techniques.

In this paper we are interested in the use of genetic algorithms. Genetic algorithms are search heuristics that allow to tackle very large search spaces. A genetic algorithm looks for a "good" solution to the input problem by emulating the evolution of a population whose individuals are possible solutions to the given problem. Once we have provided an evaluation function for the compositions, composing music can be seen as a combinatorial problem in a tremendously large search space. This makes the genetic approach very effective for our problem. As we will explain later in the paper, in our algorithm each individual of the evolving population is a complete dodecaphonic composition created from the seed series.

We have implemented the algorithm using Java and we have run several tests. In the final section of the paper we report the results of the tests that show the behavior of the algorithm.

Related work. Several papers have proposed the use of genetic algorithms for music composition (e.g. [9,10,14,15]); to the best of our knowledge none of the papers that propose genetic algorithms has considered dodecaphonic music except for a recent paper by Maeda and Kajihara [11] that proposes a genetic algorithm for the choice of a seed series. The fitness function used exploits the concept of consonance and dissonance to evaluate the candidate solutions. The cited algorithm builds only the seed series of a dodecaphonic composition. Our genetic algorithm takes a seed series as input and produces a complete dodecaphonic composition.

Surfing the web, it is possible to find software that produces dodecaphonic compositions. However no details or documentation about the algorithms used is provided.

2 Music Background

In this section we briefly recall the needed musical background to understand the remainder of the paper.

The music system we are accustomed to is the 12-tone equal temperament which divides all pitches into octaves and within each octave defines 12 notes, which are `C,C♯,D,D♯,E,F,F♯,G,G♯,A,A♯,B`. The notion of "tonality" is the heart of the equal temperament. Roughly speaking, a tonality is a set of notes, which form a scale, upon which a composition is built. The 12 notes of the equal tempered system are "equally spaced" and this makes transposition between different tonalities very easy. A tonality is built starting from one of the 12 possible notes; hence we can have 12 different starting points (tonalities). For each tonality there are several modes, of which the *major* and *minor* modes are the most used ones.

A tonal composition has a main tonality, for example, `D` major, upon which the composition is built. Notes of that tonality, `D,E,F♯,G,A,B,C♯,D` in the example, are more important than other notes, and also specific degrees of the scale have special roles (e.g., tonic, dominant, sub-dominant).

Dodecaphonic music departs from the notion of tonality. We do not have anymore scales and tonalities but all 12 notes in the octave are equally important. There are no special degree functions for the notes in the scale; in fact there are no scales anymore. Instead of the scale the structural skeleton of the composition consist of a sequence (that is, a particular order) of the 12 notes called *seed series*. An example of 12-note series is provided in Figure 1. The figure emphasizes the intervals between the notes; such intervals are indicated with a number (the size) and a letter (the type). For example "2M" indicates a major second, while "2m" indicates a minor second; "A" stands for augmented and "P" for perfect. Notice that intervals are always computed as ascending; for example the interval between the fourth (A♯, or 10) and the fifth note (C♯, or 1) of the series is a minor third because we consider the first C♯ above the A♯.

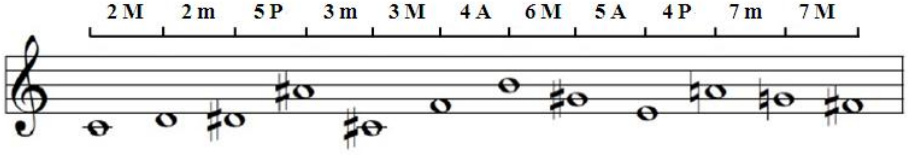

Fig. 1. Example of a seed series

A 12-note series is usually represented with an abbreviated form in which each of the twelve notes correspond to an integer in the range $[0, 11]$, where 0 denotes `C`, 1 denotes `C♯` , and so on up to 11 that denotes `B`. For example the series shown in Figure 1 corresponds to the sequence $[0, 2, 3, 10, 1, 5, 11, 8, 4, 9, 7, 6]$.

The seed series is then manipulated in order to obtain different fragments of music that are used to build up the dodecaphonic composition. Four standard manipulations can be used to obtain a set four of derived series, which are

1. `O`: Original, this is the seed series;
2. `R`: Retrograde, obtained by reversing the order of the intervals of the seed series;

3. `I`: Inversion, in which every interval in the seed series is inverted;
4. `RI`: Retrograde of Inversion, which is the retrograde of `I`.

Some theorists consider an extended set of derived series which includes beside `O, R, I, RI` also the following:

1. `D4`: Fourth, based on correspondences with the circle of fourths;
2. `D5`: Fifth, based on correspondences with the circle of fifths;
3. `R4`: Retrograde of `D4`;
4. `R5`: Retrograde of `D5`.

For more information about the derived series `D4` and `D5` see [6].

The derived series are not melodies, but only a reference schema for the composition. Rhythmic choices play a crucial role in creating the actual music from the derived series. Later we will explain how we choose rhythmic patterns. A *series transformation* will consists in the choice of a derived series and the application of a rhythmic pattern. Roughly speaking, the compositional process can be seen as the application of several series transformations.

The transformations can be used to obtain fragments of music for several voices to make up polyphonic compositions. Dodecaphonic music theory establishes some rules about the relative movements of the voices. We use such rules to evaluate the quality of the compositions.

3 The Algorithm

3.1 Rhythmic Patterns and Seed Series Transformations

The composition is made up of *transformations* of the seed series. A transformation of the seed series is obtained by first choosing a derived form of the seed series and then applying to such a derived form a rhythmic pattern. The derived forms that we consider are the ones explained in the previous section.

A rhythmic pattern P is a list $(n_1^{d_1,r_1}, \ldots, n_m^{d_m,r_m})$, with $m \geq 12$, where for each $n_k^{d_k,r_k}$ we have:

- $n_k \in \{0, 1, \ldots, 11, -1\}$, indicates the note ($-1$ indicates a rest note);
- $d_k \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$, indicates the duration of note n_k;
- $r_k \in \{1, 2, \ldots, r_{max}\}$, indicates the number of repetitions of note n_k (r_{max} specifies the maximum number of repetitions that we allow).

Notice we have $m \geq 12$ because we can insert rests in the transformed series. For example, let F be the series $[0, 2, 3, 10, 1, 5, 11, 8, 4, 9, 7, 6]$ and P be the rhythmic pattern

$$(0^{\frac{1}{4},1}, 2^{\frac{1}{4},1}, 3^{\frac{1}{8},1}, 10^{\frac{1}{8},1}, -1^{\frac{1}{4},1}, 1^{\frac{1}{2},2}, 5^{\frac{1}{4},1}, 11^{\frac{1}{4},1}, 8^{\frac{1}{2},1}, 4^{\frac{1}{4},2}, -1^{\frac{1}{2},1}, 9^{\frac{1}{4},2}, 7^{\frac{1}{2},1}, 6^{1,1}),$$

where $m = 14$ because there are 2 rests. Figure 2 shows the result of a transformation of F obtained choosing the `R` (retrograde) derived form and applying the rhythmic pattern P to it. Remember that the intervals are always ascending intervals.

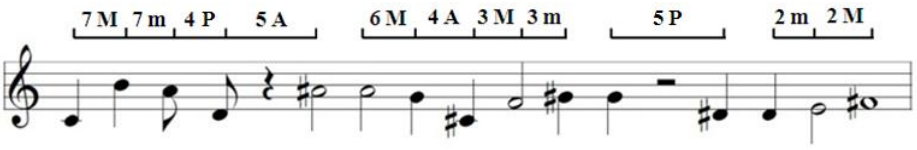

Fig. 2. Example of transformation

3.2 Chromosomes and Genes Representation

We represent each chromosome as an array C of dimension $K \times V$, where $K > 1$ denotes the number of melodic lines (voices) and V is an apriori fixed number that determines the length of the composition. We assume that all melodic lines are composed by the same number V of series transformations.

Formally, a chromosome C is an array $C = G_1, ..., G_V$, where each $G_i = T_i^1, ..., T_i^K$ is a gene and each T_i^j is a transformation of the seed series. Figure 3 shows a graphical representation of the structure of a chromosome. Notice that the blocks that represent genes are not aligned because the transformations T_i^j change the length of the seed series.

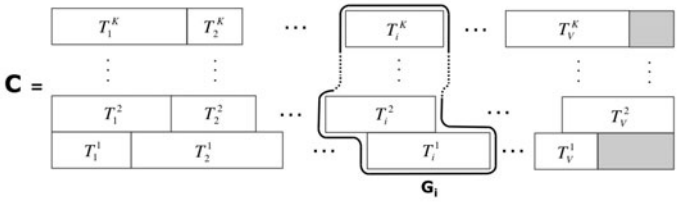

Fig. 3. Chromosomes and genes structure

3.3 Genotype-Phenotype Mapping

Since the seed series, and thus its transformations, does not specify the exact pitch of the notes (the same note can be played in several octaves, depending on the voice extension), we have that the chromosomes do not correspond immediately to solutions of the problem and thus our search space is different from the problem space. This implies that given a chromosome, in order to obtain a solution, we need to apply a genotype-phenotype mapping. In the following we define such a mapping.

To each voice we associate an *extension*, that is, a range of pitch values that the voice can reproduce. We use the MIDI values of the notes to specify the exact pitch. For example the extension $[48, 72]$ indicates the extension from $\texttt{C}_3$ through $\texttt{C}_5$. Recall that each note s in a series is denoted with an integer in the interval $[0, 11]$. We need to map such an integer s to a pitch (MIDI) value.

For each of the K voices, we fix a range $Ext_j = [\ell_j, u_j]$, $j = 1, 2, ..., K$, where ℓ_j is the lowest pitch and u_j is the highest pitch for voice j. Given a note $s \in [0, 11]$ played by voice j, the genotype-phenotype mapping has to choose an actual note by selecting a pitch in Ext_j.

The possible pitch values for a note s are the values in the set $Ext_j(s) = \{s' | s' \in Ext_j, s' \mod 12 = s\}$. For example, given $s = 0$, that is note C, and the extension $Ext = [57, 74]$, the possible MIDI values for s are $Ext(0) = \{60, 72\}$.

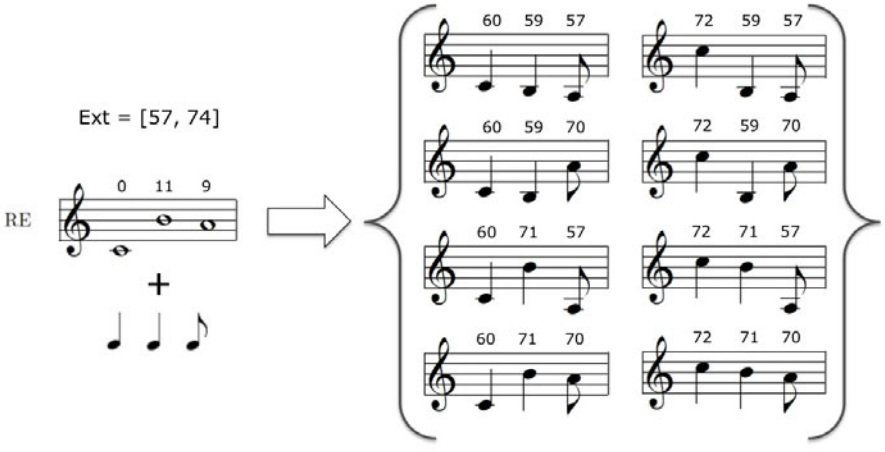

Fig. 4. An example of genotype-phenotype mapping

Figure 4 shows an example of genotype-phenotype mapping. To keep the example small we used a 3-note series instead of a complete 12-note series. In the example the chromosome consists of a single voice playing the 3-note series $\{0, 11, 9\}$, the voice extension is $Ext = [57, 74]$ and the rhythmic pattern is $(0^{\frac{1}{4},1}, 11^{\frac{1}{4},1}, 9^{\frac{1}{8},1})$. All the possible actual compositions that we can obtain in this simple example are 8 because for each of the 3 notes in the series we have exactly two possible actual pitches that we can choose. Figure 4 shows all these 8 possible compositions. Notice that with 12-note series and reasonable parameters K and V the space of all possible compositions is enormous.

The genotype-phenotype mapping has to choose one of the possible actual compositions to transform the chromosome in a solution of the problem. It does so by selecting uniformly at random one of the possible actual compositions. Notice that choosing an actual composition uniformly at random is equivalent to choosing uniformly at random an actual note for each abstract note.

We denote with *gpm* the genotype-phenotype mapping. Hence if we take a chromosome C and apply the genotype-phenotype mapping we obtain an actual music composition $X = gpm(C)$ where X is selected at random among all the possible compositions that correspond to C.

3.4 Initial Population

We start from a random initial population of N individuals. Each individual, which is made up of $K \times V$ seed series transformations, is constructed by selecting

such transformations at random. The random choices are made as follows. For the derived form we choose at random one of the 8 forms explained in the music background section. In order to choose a random rhythmic pattern P for a derived series $[s_1, \ldots, s_{12}]$, we do the following

Let $i = 1$; do
1. Choose either
   (a) note s_i with an apriori fixed probability $p_{nr} > \frac{1}{2}$ or
   (b) -1 (rest) with probability $1 - p_{nr}$
   and let n_k be the result of the choice.
2. Assign to d_k a random value chosen in the set $\{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$.
3. Assign to r_k a random value chosen in the set $\{1, 2, \ldots, r_{max}\}$.
4. Add $n_k^{d_k, r_k}$ to P.
5. If $n_k \geq 0$ then increment i (that is, if we have selected s_i and not a rest, proceed to s_{i+1}).
6. If $i = 13$ we are done. Otherwise repeat from 1.

3.5 Fitness Measure

In order to evaluate the current population and make a selection, we use rules taken from the dodecaphonic music theory. We refer the reader to textbooks such as [2,6] for an explanation of such rules. The specific set of rules that we have taken into account are detailed in Table 1. The rules refer to the movement of two voices of the composition. For example, the "parallel octave" rule says that two voices that proceed in parallel octaves are considered a critical error. As specified in the table, we have assigned a penalty of 100 to such an error. The "cross" rule says that two voices that cross each other are also considered an error, but not so critical. The penalty that we use in this case is lower.

Table 1. Errors and penalties for the evaluation function

Description	Penalty	Level
unison	30	normal
octave	30	normal
cross	30	normal
parallel fourth	40	normal

Description	Penalty	Level
parallel fifth	40	normal
parallel seventh	40	normal
parallel unison	100	critical
parallel octave	100	critical

We have taken the liberty of interpreting and adapting what is reported in dodecaphonic theory textbooks in order to obtain the rules reported in Table 1. For example voice crossing is normally allowed; however we do assign it a small penalty because we would like that voices do not cross too often. We have also classified these errors in two categories, normal and critical. The critical errors will be used for the mutation operator, as we will explain in later.

Let C be the chromosome and let $X = gpm(C)$ be the solution that will be used for the evaluation. Notice that this means that the actual fitness value that we use for C is the one that we obtain for X. In other words the fitness value is relative to the actual solution chosen by the *gpm* function.

Let d be the smallest note duration used in X. We will consider such a note duration as the "beat" of the composition. Notice that this "beat" is used only for the evaluation function. Depending on the value of d we can look at the entire composition as a sequence of a certain number z of beats of duration d. For each of these beats we consider all possible pairs of voices and for each pair we check whether an error occurs or not. If an error occurs then we give the corresponding penalty to X. The fitness value is obtained by adding all the penalties. The objective is to minimize the fitness value.

While evaluating the population, when we find a critical error we *mark* the corresponding gene. This information will be used in the mutation operator. Notice that if the error occurs across two genes we mark both genes.

3.6 Evolution Operators

In order to let the population evolve we apply a *mutation* operator and a *crossover* operator.

- **Mutation operator.** This operator creates new chromosomes starting from a chromosome of the current population. For each chromosome $C = G_1, ..., G_V$ in the current population, and for each gene $G_i = T_i^1, ..., T_i^K$, we check whether G_i has a critical error (that is if it has been marked in the evaluation phase) and if so we generate a new chromosome replacing G_i with a new gene $G'_i = {T'}_i^1, ..., {T'}_i^K$ where ${T'}_i^j$ is obtained by applying a transformation to T_i^j. The transformation is chosen at random. We remark that the transformation might include the "identity" in the sense that we might not apply a derivation and/or we might not change the rhythmic pattern.
- **Crossover.** Given two chromosomes C^1 and C^2, the operator selects an index $i \in \{1, \ldots, V-1\}$ randomly, and generates the chromosome $C^3 = G_1^1, \ldots, G_i^1, G_{i+1}^2, \ldots, G_V^2$.

3.7 Selection and Stopping Criteria

At this point as a candidate new population we have the initial population of N chromosomes, at most new N chromosomes obtained with the mutation operator and exactly $N(N-1)/2$ new chromosomes obtained with the crossover operator. Among these we choose the N chromosomes that have the best fitness evaluation. We stop the evolutionary process after a fixed number of generations (ranging up to 200 in the tests).

4 Test Results

We have implemented the genetic algorithm in Java; we used the JFugue library for the music manipulation subroutines. We have chosen as test cases several seed series taken from [6]. We have run several tests varying the parameters on many input series. The results obtained in all cases are quite similar.

Figure 5 shows the data obtained for a specific test using $K = 4$, $V = 30$, $p_{nr} = 0.8$. The size of the population is $N = 100$. Figure 5 shows the fitness value and the number of errors of the best solution as a function of the number of generations.

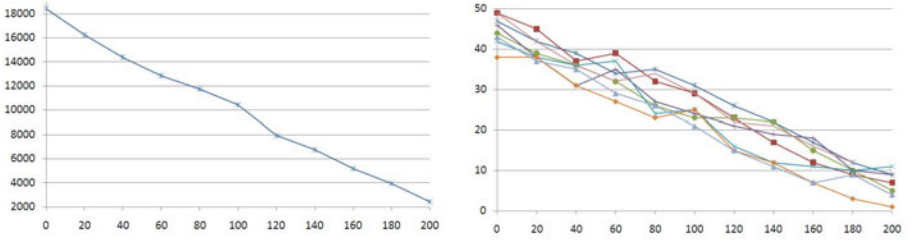

Fig. 5. Fitness value and errors as function of the number of generations

Since in this test we have used $V = 30$ the length of the composition doesn't allow us to show the best solution. However, just as an example, Figure 6 shows the actual composition given as output for a similar test in which we used $V = 2$ so that the total length of the composition is very small.

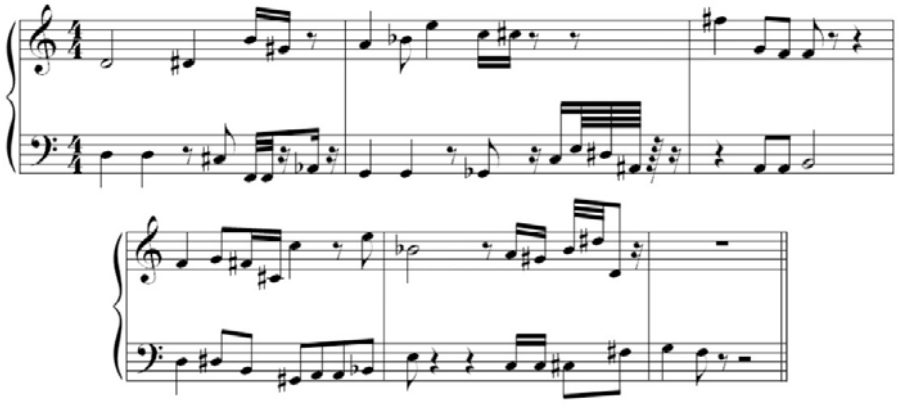

Fig. 6. Dodecaphonic music given as output for an example with $V = 2$

5 Conclusions

In this paper we have provided an automatic music composition system for dodecaphonic music using a genetic algorithm. The output of the system is promising. In this first version the system considers only the basic rules for the composition of dodecaphonic music. Future work might include: (*a*) study of the impact of the errors and penalties used; (*b*) use of a more complex fitness function taking into

account not only harmony rules but also rhythmic and melodic considerations; (*c*) study of the impact of the random choices (for example the probabilities seed for the selection of the rhythmic pattern); (*d*) development of a more complex version of the system in order to include advanced features, like, for example, symmetry rules spanning the entire music composition.

References

1. Biles, J.A.: GenJam: A genetic algorithm for generating jazz solos. In: Proceedings of the International Computer Music Conference, pp. 131–137 (1994)
2. Brindle, R.S.: Serial Composition. Oxford University Press, London (1966)
3. Cope, D.: Web page, `http://artsites.ucsc.edu/faculty/cope/`
4. De Prisco, R., Zaccagnino, R.: An Evolutionary Music Composer Algorithm for Bass Harmonization. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., Machado, P. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 567–572. Springer, Heidelberg (2009)
5. De Prisco, R., Zaccagnino, G., Zaccagnino, R.: EvoBassComposer: A multi-objective genetic algorithm for 4-voice compositions. In: Proceedings of the 12th ACM Conference on Genetic and Evolutionary Computation (GECCO), pp. 817–818. ACM Press, New York (2010)
6. Eimert, H.: Lehrbuch der Zwölftontechnik. Breitkopf & Härtel, Wiesbaden (1950)
7. Hiller, L.: Computer music. Scientific American 201(6), 109–120 (1959)
8. Hiller, L., Isaacson, L.M.: Experimental music. McGraw-Hill, New York (1959)
9. Horner, A., Goldberg, D.E.: Genetic algorithms and computer assisted music composition. Technical report, University of Illinois (1991)
10. Horner, A., Ayers, L.: Harmonization of musical progressions with genetic algorithms. In: Proceedings of the International Computer Music Conference, pp. 483–484 (1995)
11. Maeda, Y., Kajihara, Y.: Automatic generation method of twelve tone row for musical composition using genetic algorithm. In: Proceedings of the IEEE 18th International conference on Fuzzy Systems FUZZY 2009, pp. 963–968 (2009)
12. Miranda, E.R.: Composing Music with Computers. Focal Press (2001)
13. Miranda, E.R.: Web page, `http://neuromusic.soc.plymouth.ac.uk/`
14. McIntyre, R.A.: Bach in a box: The evolution of four-part baroque harmony using a genetic algorithm. In: Proceedings of the 1st IEEE Conference on Evolutionary Computation, pp. 852–857 (1994)
15. Wiggins, G., Papadopoulos, G., Phon-Amnuaisuk, S., Tuson, A.: Evolutionary methods for musical composition. International Journal of Computing Anticipatory Systems 4 (1999)