COMPUTER-GENERATED MUSIC USING GRAMMATICAL EVOLUTION

Abdel Latif Abu Dalhoum (1), Manuel Alfonseca (2), Manuel Cebrián (2), Rafael Sánchez-Alfonso (3) and Alfonso Ortega (2)
(1) University of Jordan, Computer Science Department, a.latif@ju.edu.jo
(2) Escuela Politécnica Superior, Universidad Autónoma de Madrid SPAIN, {Manuel.Alfonseca, Manuel.Cebrian, Alfonso.Ortega}@uam.es
(3)
Universidad Autónoma de Madrid & Neoris, Rafael.Sanchez@neoris.com

KEYWORDS

Evolutionary Computation, Grammatical Evolution, Genetic Algorithms, Computer Generated Music, Coding and Information Theory, Clustering

ABSTRACT[1]

This paper proposes a new musical notation with Lindenmayer grammars, and describes the use of grammar evolution for the automatic generation of music expressed in this notation, with the normalized compression distance as the fitness function. The computer music generated tries to reproduce the style of a selected pre-existent piece of music. In spite of the simplicity of the algorithm, the procedure obtains interesting results.

INTRODUCTION

The automatic generation of musical compositions is a long standing, multi disciplinary area of interest and research in computer science, with over thirty years history at its back [1-7]. Our group is interested in the simulation of complex systems by means of formal models, their equivalence and their design, not only by hand, but also by means of automatic processes, such as evolutionary computation, genetic programming and grammar evolution [8-9].

This paper extends Grammatical Evolution to the generation of music similar to a given target or guide piece, in such a way as to capture some of its stylistic properties. Two different generating procedures and fitness functions have been tested and compared.

This paper is organized thus: the second section provides a short introduction to musical concepts, with an enumeration of different ways of representing music, one of which is first presented here. The third section provides an introduction to grammatical evolution, and explains how it has been implemented to generate music. The fourth explains the normalized compression distance, which has been used to compute the distance of the results of the genetic algorithm from the target musical pieces. In the fifth section we describe our experiments. Finally, the last section presents our conclusions and possibilities for future work.

MUSICAL REPRESENTATION

The three fundamental elements in music are melody, rhythm and harmony. In the experiments performed in this paper, we shall restrict ourselves to melody, leaving the management of rhythm and harmony as future objectives. In this way, we can forget about different instruments (parts and voices) and focus on monophonic music: a single performer executing, at most, a single note on a piano at a given point in time. Melody consists of a series of musical sounds (notes) or silences (rests) with different lengths and stresses, arranged in succession in a particular rhythmic pattern, to form a recognizable unit.

In the English notation for Western music, the names of the notes belong to the set {A, B, C, D, E, F, G}. These letters represent musical pitches and correspond to the white keys on the piano. The black keys on the piano (the sharp or flat notes) are modifications of the white key notes. From left to right, the key that follows a white key is its sharp key, while the previous key is its flat key. A black key is indicated by adding a symbol to the white key name (as in A# or A+ to represent A sharp, or in B♭ or B- to represent B flat).

[1] *Acknowledgement*: This work has been partially sponsored by the Spanish Ministry of Science and Technology (MCYT), project number TIC2002-01948.

The distance from a note to its flat or sharp notes is called a "half step" and is the smallest unit of pitch used in the piano, where any two adjacent keys are separated by a half step, no matter their color. Two consecutive half steps are called a whole step. Instruments different from the piano may generate additional notes; in fact, flat and sharp notes may not coincide; also, in different musical traditions (such as Arab or Hindu music) additional notes exist. In this paper we shall restrict to the Western piano lay-up, thus simplifying the problem to just 88 different notes separated by half steps. By counting the number of half steps between two notes, we can obtain the interval, or difference in pitch between the two notes. Intervals between two successive notes are called melodic intervals, whereas those between two simultaneously sounding notes are harmonic intervals. Since this work focuses on monophonic music, only melodic intervals will be considered.

Notes and rests have a length or time duration. There are nine different standard lengths, each double than the next. We will consider only the first seven: whole, half, quarter, quaver, semi-quaver, quarter-quaver and half quarter-quaver. Intermediate durations are obtained by means of dots or periods. The complete specification of notes and silences includes their lengths.

A piece of music can be represented in several different, but equivalent ways:

- With the traditional Western bi-dimensional graphic notation on a pentagram.

- By means of certain coding systems which are used to keep and reproduce music in a computer or a recording system, with or without compression, such as wave sampling, MIDI, MP3, etc.

- Through a set of character strings: notes are represented by letters (A-G), silence by a P, sharp and flat alterations by + and – signs, and the lengths of notes by a number (0 representing a whole note, 1 a half note, and so on). Adding a period provides intermediate lengths. Additional codes define the tempo, the octave and the performance style (normal, legato or staccato). Polyphonic music is represented with sets of parallel strings.

- By means of sets of integer pairs. The first number in the pair represents the pitch of the note in the piano keyboard (1 to 88), with 0 representing a silence. The second corresponds to the length of the note, as a multiple of the minimum unit of time. Polyphonic music may be represented by means of parallel sets of integer pairs.

- In this paper we are offering a new way to represent music by means of a character string, similar to those used with turtle graphics [10]. the alphabet used consists of four symbols: $\Sigma = \{ F, f, +, - \}$. A given note with a given length is represented by a string of successive letters 'F' (as many as the length of the note is a multiple of the minimum length). The pitch of the note is represented by the state of the turtle, which is defined as the number of '+' signs minus the number of '-' signs previous to the first F in the current note. If this number is positive, the current note is located as many half steps above the initial state note; if it is negative, the same number of half steps below the initial note. The initial note (which corresponds to the initial state of the turtle) is assumed to be note C in octave 3 in the piano. Strings with the letter 'f' represent silences with the appropriate length (in this case the pitch information is ignored). For example: the set of two notes O3C3D3 would be translated as FF++FF if we consider semiquavers to be the minimum note length.

In our experiments, we represent melodies by the last three notation systems. We have built functions which translate music represented in one of those three formats into any of the others and into the MIDI format, for reproduction.

GRAMMATICAL EVOLUTION

Grammatical Evolution (GE) [11-12] is a grammar based, linear genome system, which has been applied in the area of automatic programming to automatically generate programs or expressions in a given language that solve a particular problem. Chomsky serial derivation grammars have been applied in most GE applications. We have also used Lindenmayer parallel derivation grammars [13]. The genotype is usually a string of 8 bit binary numbers, treated as integer values from 0 to 255. The phenotype is a running computer program generated by a genotype-phenotype mapping process. The mapping benefits from genetic code degeneracy, i.e. different integers in the genotype generate the same phenotype. The genotype-phenotype mapping in Grammatical Evolution is deterministic: each individual is always mapped to the same phenotype.

In Grammatical Evolution, standard genetic algorithms are applied to the different genotypes in a population, using crossover and mutation operators. For each domain, the proper fitness function must be designed to be used by the genetic algorithm to perform selection. This technique has been successfully applied to the automatic programming of problems in many different domains. We have used it to generate fractal curves with a given dimension [13].

Our genetic algorithm generates music from an initial population of 64 vectors of integers in the [0-255] interval. The genotypes of individuals in the population are translated into music by means of the following Lindenmayer grammar:

```
F ::= F                      (0)
    | FF                     (1)
    | F+                     (2)
    | F-                     (3)
    | +F                     (4)
    | -F                     (5)
    | F+F                    (6)
    | F-F                    (7)
    | +                      (8)
    | -                      (9)
    | FFFF                   (10)
    | FFFf                   (11)
f ::= f
+ ::= +
- ::= -
```

The translation is performed according to the following algorithm:

1. The axiom (first word) of the Lindenmayer grammar is the string 'F'. Rules in the previous grammar are numbered 0 to 11.
2. As many elements from the remainder of the genotype are taken (and removed) from the left of the genotype as the number of F in the current word. If there remain too few elements in the genotype, the required number is completed circularly.
3. The current word derives a new one in the following way: each F in the word is replaced by the right hand side of the rule whose number is equal to the remainder of the corresponding genotype integer obtained in the previous step modulo 12.
4. If the genotype is now empty, the algorithm stops and the last derived word is the output, a piece of music represented by the notation we are proposing, which has been described in the previous section.
5. If the derived word contains no 'F', the whole word is replaced by the axiom.
6. Go to step 2.

Example: Let the genotype be the string of integers `112 125 203 14 87 136 224`. The initial string 'F' contains one 'F'. The first element in the genotype is 112. Its remainder module 12 is 4. Rule 4 is applied, generating the string '+F'. This is the end of the first loop.

In the second loop, we start with genotype `125 203 14 87 136 224`. The current string '+F' contains one 'F'. The first element in the genotype is 125, whose remainder module 12 is 1. Rule 1 is applied, generating the string '+FF'. Here ends the second loop.

In the third loop, we start with genotype `203 14 87 136 224`. The current string '+FF' contains two 'F'. The first two elements in the genotype are 203 14, whose remainders module 12 are 11 and 2. Rules 11 and 2 are applied (one to each 'F'), generating the string '+FFFfF+'.

In the fourth loop, we start with genotype `87 136 224`. The current string '+FFFfF+' contains four 'F'. The genotype has only three elements, therefore the first one is added to the end circularly, giving 87 136 224 87, whose remainders module 12 are 3, 4, 8 and 3. Rules 3, 4, 8 and 3 (one to each 'F'), generating the string '+F-+F+fF-+'. Since the genotype is now empty, this string is the phenotype, which corresponds to notes 'C+C+PD', all with the same basic length.

We have used the following genetic algorithm to generate music:

1. One musical piece is selected as the target or guide for music generation.
2. The program generates a random population of 64 vectors of N integers, where N is the length of the guide piece. Each vector represents a genotype.
3. The fitness of the genotypes is computed as the distance to the guide set, measured by means of the normalized compression distance (see below).
4. The genotypes are ordered by their increasing distance to the guide set.
5. If the goal distance has been reached, the program stops.
6. 30 pairs of parent genotypes are built. Genotypes with better fitness have a higher probability of being used. Each pair of parents gives rise to two pairs of children, made of copies of the parents and modified by four genetic operations. The 60 genotypes with least fitness are removed. (The 4 genotypes with the best fitness remain in the next population). The 60 children are added to the population to make again 64, and their fitness is computed as in step 3.
7. Go to step 4.

The four genetic operations mentioned in the algorithm are:

- Recombination (applied to 100% generated genotypes). The genotypes of both parents are combined using different procedures to generate the genotypes of the progeny. Different recombination procedures have been tested in this set of experiments to find the best combination.

- Mutation (one mutation was applied to every generated genotype, although this rate may be modified in different experiments). It consists of replacing a random element of the vector by a random integer in the same interval.

- Fusion (applied to a certain percentage of the generated genotypes, which in our experiments was varied between 5 and 10). The genotype is replaced by a catenation of itself with a piece randomly broken from either itself or its brother's genotype.

- Elision (applied to a certain percentage of the generated genotypes, in our experiments between 2 and 5). One integer in the vector (in a random position) is eliminated.

The last two operations, together with some recombination procedures, allow longer or shorter genotypes to be obtained from the original N element vectors.

THE NORMALIZED COMPRESSION DISTANCE

The fitness function we have used for the genetic algorithm computes the distance between two pieces of music by expressing both as note-length pair sequences (the third notation in the previous discussion), converting the sequence of notes to a sequence of intervals (this is done by subtracting each note from the next) and computing the distance between both sequences of intervals by means of the normalized compression distance [14-16], an approximation of the universal distance between any two objects defined as a function of the Kolmogorov complexities of the objects, which can be computed by means of standard compressors. The expression we have used to compute the normalized compression distance is the following:

$$\hat{d}(x,y) = \frac{C(xy) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}}$$

Where `C(x)` is the length obtained by compressing object `x` with compressor `C`, and `xy` is the concatenation of `x` and `y`. This metric has been reported to outperform some of the finest algorithms for clustering music by genre [17]. This suggests that the normalized compression distance works well to extract features shared between two musical pieces. The compressor used to compute the distance between the musical objects (the two sequences of intervals) was the LZ77 algorithm [18-19].

We decided to apply the fitness function only to the relative pitches of the notes in the melody (this is done by working on intervals rather than absolute notes), ignoring the absolute pitches and the note lengths, because our own studies and other's [17] suggest that a given piece of music remains recognizable when the lengths of its notes are replaced by random lengths, while the opposite doesn't happen (the piece becomes completely unrecognizable if its notes are replaced by a random set, while maintaining their lengths).

EXPERIMENTS

In our experiments, we first used as the guide *Yankee Doodle*, represented by the following string:

```
M2T2O3L2C+4C+4D+4F4C+4F4D+4O2G+4O3C+4
C+4D+4F4C+3C4P4C+4C+4D+4F4F+4F4D+4C+4C4
O2G+4A+4O3C4C+3C+4P4O2A+4.O3C5.O2A+4G+
4A+4O3C4C+4P4O2G+4.A+5.G+4F+4F3G+4P4A+4.
O3C5.O2A+4G+4A+4O3C4C+4O2A+4G+4O3C+4C4
D+4C+3C+3
```

After applying the genetic algorithm, the succession of notes obtained was completed by adding length information in the following way: each note was assigned the length of the note in the same position in the guide piece (the guide piece was shortened or circularly extended, if needed, to make it the same length as the generated piece, which could be shorter or longer). The number of generations needed to reach a given distance to the guide depends on the guide length and the random seed used in each experiment, and follows an approximate Poisson curve [20]. In successive executions of the algorithm with different random seeds, we obtained different melodies at different distances from the guide. For instance, the distance to the guide of the following generated piece is 0.58 (figure 1 shows the beginning in typical musical notation):

T1O3M2C+4O3D4O3D3O3D4O3C+4P4O3E3.O3F+4
P4O3E4P4O3D4O3C+4O3C3O3C+3O2M1B3.P4O2
M2B4O3C+2O2A3O2G+3O2M1A2P4O2M2A4O2A+
2O2A3.O2G4O2F+4O2G3P3O2F4O2F+4O2F+4P3.O
2D+4O2F1P4O2F4O2F3O2E4P4O2F4P4O2E4P4O2F
4O2D4O2C4O2M1C+3.P4O2M2C2O2C+4O2D4O2D
+4O2D+3.O2M1E3.P4O2M2F4O2F3O2G2O2A4O2A
+3O2A4O2A+4O2B4P4P4O2B4O3C4

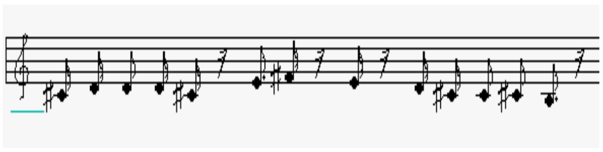

Figure 1: Beginning of music generated from *Yankee Doodle*.

In another experiment, we used the seventh prelude by Chopin as the guide piece, getting the following result at a distance of 0.67 from the guide (see also the beginning in figure 2):

T0O3M2C4O2M1A+3.P4O2M2A+4O2B4P4O2A+4O
2M1B3.P4O2M2A+4O2B4O3E4O3E4O3M1E2P4O3
M2E4O3F4P4O3F4O3F4O3F4O3M0D1P3O3M2D+4
O3E4O3C+4O3M1C+3.P4O3M2C+4O3D4P4O3C+4
O3M1D3.P4O2M2B3O3C4O3D+4P4O3D+4O3M0C+
3.P3O3M2C4O3C+4O3C4O3C4O3C+4P3O2A4O2A+
2O2A2O2A3O2A4O2M1A+3.P4O3M2D4O3F4O3D+
4O3M1D+3.P4O3M2D+3O3E4P4O3D+4

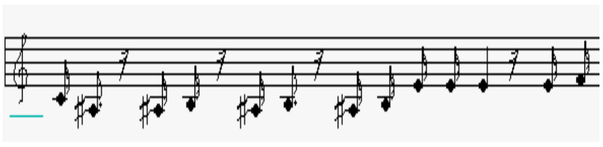

Figure 2: Beginning of music generated from Chopin's seventh prelude.

It can be noticed that our first generated music has a lighter sound to the ear than the second, which may be a consequence of the different guide pieces used in both cases.

In our third experiment we changed the fitness function to work directly at the level of the music represented by the turtle strings. The normalized compression distance was also used to compute the distance of the two character strings (the guide piece and each member of the population). In this case, both the pitches and the lengths of the notes were evolved at the same time. The following shows one result we obtained using Chopin's seventh prelude as the guide piece (see also the beginning in figure 3):

T1O2M2D+4O2F2.O2F+4O2G+3O2G+3P4O2G+4P4
O2A2O2A+4O2A4O2A+4O2A4O2A+4P4O3F4O3D+
3O3E4O3F3O3A+4O3A3O3G+4O3A3O3B3O3A+4O
3B4O3B4O3A+4O3G+4O3G+4O3F+3O3G4O3M1G3
.P4O3M2G2O3F+4P4O3G2.O3F+4O3G4O3F+4O3F4
O3F+4O3F+2O3G+4O3G4P4O3G+4O3A3O3G+4O3
A3O3A+4O3A4O3A+3O3A+4O3M1B3.P4O3M2B3P
4O3B4O3A+4O3A2O3G+4P4O3B4O3B4O3A+4O3G
+4O3G+4O3A3O3G+4P4O3A4O3A2O4C2P4O3M1B
3.P4P3O3M2B4O4C4O3B4O4C4P4P4O4C+4O4D4O
4E4O4M1D+2P4O4M2D+4P4P4O4F+2O4G4O4F4O
4E4O4D+3P4P4O4G+4O4G4O4E4O4F4O4F+4O4G4
O4G4O4F+4P4O4G4O4F+4O4G3.O4A3O4A4O4G+4
O4A3O4G+4O4A4P4O4M1F+3.P4P3O4M2G1.O4G+
3O4E3O4C+4O3A3O3M1G+3.P4O3M2F4O3F3O3E3
O3E3O3G4O3G4O3F+4O3M1G2.P4O3G1O3M2G+4
O3A3O3G+4P4O3A4O3G+4O3A4O3A3.O3G4P4O3
F4O3F+4O3M1F2P4O3M2F+1O3F+3O3F4P4O3F2O
3D4

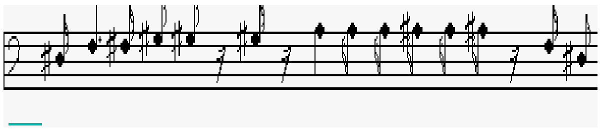

Figure 3: Beginning of another piece of music generated from Chopin's seventh prelude.

CONCLUSIONS AND FUTURE WORK

We have defined a new musical notation which can be used to automatically generate music by means of grammatical evolution and a genetic algorithm, minimizing its distance from a given guide piece. In the first set of experiments, just the pitches of the notes were evolved, in another set of experiments both pitches and lengths evolved at the same time. Some of the pieces thus generated seem to capture some of the characteristics of the guide piece (such as sounding lighter or classical-like).

In the future we intend to combine our results with those of other authors [18-19] to use as the target for the genetic algorithm more than one piece of music by the same author, thus trying to capture the style in a more general way. We also intend to test different fitness functions and the effect of different recombination procedures.

REFERENCES

[1] J. McCormack (1996). Grammar-based music composition. *Complex International,* Vol 3.

[2] J. Biles (1994). GenJam: A Genetic Algorithm for Generating Jazz Solos, *Proceedings of the 1994 International Computer Music Conference*, ICMA, pp. 131-137, San Francisco, 1994.

[3] E. Bilotta, P. Pantano, V. Talarico (2000). Synthetic Harmonies: an approach to musical semiosis by means of cellular automata, *Leonardo*, MIT Press, vol. 35:2, pp. 153-159, April 2002.

[4] D. Lidov, J. Gabura (1973). A melody writing algorithm using a formal language model, *Computer Studies in the Humanities* Vol. 4:3-4, pp. 138-148, 1973.

[5] P. Laine, M. Kuuskankare (1994). Genetic Algorithms in Musical Style oriented Generation, *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp 858-862, Orlando, Florida, vol. 2, 1994.

[6] D. Horowitz (1994). Generating Rhythms with Genetic Algorithms, *Proceedings of the ICMC 1994*, pp. 142-143, International Computer Music Association, Århus, 1994.

[7] B. Jacob (1995). Composing with Genetic Algorithms, *Proceedings of the 1995 International Computer Music Conference*, pp. 452-455, ICMC, Banff Canada, 1995.

[8] Alfonseca, M., Ortega, A., Suárez, A. (2003). Cellular automata and probabilistic L systems: An example in Ecology, in *Grammars and Automata for String Processing: from Mathematics and Computer Science to Biology, and Back*, ed. C. Martin-Vide & V. Mitrana, Taylor & Francis, pp. 111-120. ISBN: 0415298857.

[9] Ortega, A., Abu Dalhoum, A., Alfonseca, M. (2003). Grammatical evolution to design fractal curves with a given dimension, *IBM Journal of Research and Development*, Vol. 47:4, p. 483-493, Jul. 2003.

[10] Papert, S.: "Mindstorms: Children, Computers, and Powerful Ideas", Basic Books, New York, 1980.

[11] M. O'Neill and C. Ryan: *Grammatical Evolution*, IEEE Transactions on Evolutionary Computation, 5, Np.4, 349-358 (2001).

[12] M. O'Neill and C. Ryan: *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, Kluwer Academic Publishers, 2003, Book Series: Genetic Programming: Volume 4, ISBN 1-4020-7444-1.

[13] A.Ortega, A.Abu Dalhoum, M.Alfonseca: *Grammatical evolution to design fractal curves with a given dimension*, IBM Journal of Res. and Dev., Vol. 47:4, p. 483-493, Jul. 2003.

[14] M. Li, X. Chen, X. Li, B. Ma and P. Vitányi (2003). The similarity metric, *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms*, pp. 863-872.

[15] P. and M. Li (1993). An Introduction to Kolmogorov Complexity and its Applications, *Springer-Verlag*.

[16] R. Cilibrasi and P. Vitanyi (2005). Clustering by Compression, *IEEE Trans. Information Theory*, Vol.51 No.4, pp. 1523-1545.

[17] M. Li and R. Sleep (2004). Melody Classification using a Similarity Metric based on Kolmogorov Complexity, *Sound and Music Computing*.

[18] J. Ziv and A. Lempel (1997). A universal algorithm for sequential data compression, *IEEE Transactions on Information Theory*, Vol.23:3, pp. 337-343.

[19] S. R. Kosaraju and G. Manzini (1997). Some entropic bounds for Lempel-Ziv algorithms, *Data Compression Conference*, pp. 446.

[20] M. Alfonseca, M. Cebrián, A. Ortega. *A simple genetic algorithm for music generation by means of algorithmic information theory*. IEEE Congress on Evolutionary Computation (CEC'2007), Singapore, Sep.25-28, 2007. Published in the Proceedings, pp. 3025-3042, ISBN 1-4244-1340-0, IEEE Press.

SUMMARY OF AUTHOR BIOGRAPHICAL DATA

Abdel Latif Abu Dalhoum is a doctor in computer science and teaches at the university of Jordan, Manuel Alfonseca is a doctor in electronics engineering and computer scientist, and full professor at the Universidad Autónoma of Madrid, Manuel Cebrián is a doctor in computer science, Rafael Sánchez-Alfonso is a Ph. D. student and Alfonso Ortega is a professor at the Universidad Autónoma of Madrid.