Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, February 15-17, 2006 (pp53-58)

Testing Genetic Algorithm Recombination Strategies and the Normalized Compression Distance for Computer-Generated Music

MANUEL ALFONSECA, MANUEL CEBRIÁN and ALFONSO ORTEGA
Escuela Politécnica Superior
Tomás y Valiente, 11
Universidad Autónoma de Madrid
28049 Madrid
SPAIN

Abstract: - This paper analyzes the use of the normalized compression distance as a fitness function for the automatic generation of music by means of genetic algorithms, and tests the effect on performance of several genetic recombination procedures. The minimization of the distance of the generated music to a set of musical guides or targets makes it possible to obtain computer-generated music that reminds the style of a certain human author. In spite of the simplicity of the algorithm, the procedure obtains interesting results. The paper includes some considerations on the use of procedures that improve the performance of heavy-tailed distribution processes.

Keywords: - Evolutionary Computation, Coding and Information Theory, Genetic Algorithms, Computer Generated Music, Classification, Clustering

Acknowledgement: - This work has been sponsored by the Spanish Ministry of Education and Science (MEC), project number TSI2005-08225-C07-06.

1. 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].

In a previous paper [8] we proposed the use of the well-known normalized compression distance [9-11] as a fitness function which may be used by genetic algorithms to automatically generate music in a given pre-defined style. The superiority of the relative pitch envelope over other musical parameters, such as the lengths of the notes, has been confirmed, bringing us to develop a simplified algorithm that nevertheless obtains interesting results.

In this paper we start on the results of the previous work and refine them, trying to increase the efficiency of the procedures described in the above mentioned paper.

This paper is organized in the following way: the second section, describes the genetic algorithm we have used for music generation. In the third section we present in detail our experiments, where we have compared the use of four different recombination procedures for the genetic algorithm. In the fourth section we explain why a procedure which has been used successfully in other kinds of experiments to improve performance is not indicated in this case. Finally, the last section presents our conclusions and possibilities for future work.

2. A genetic algorithm that generates music

Our genetic algorithm generates music coded as pairs of integers, the first element in the pair representing the pitch of a note and the second its length. This notation can then be transformed to a note string for reproduction. In this first set of experiments, the genetic algorithm is applied only to the relative pitches of the notes in the melody.

The proposed genetic algorithm scheme is described below.

0. A previous pre-process step:
   - Select one or more musical pieces as targets or guides for music generation.
     $$\Omega = \{\omega_i\}_1^N$$
     All the ω_i must be coded in the same way, as pairs of integers as described above.
   - Code the individuals in the population with the same coding system as the guides.
   - Use the following fitness function:
     $$f(x) = \frac{1}{\sum_i \hat{d}\,(x,\omega_i)}$$
     Where $\hat{d}(x,y)$ is the normalized compression distance which was defined in [8]. We expect that, by maximizing $f(x)$ (minimizing the sum of the distances), we will maximize the number of features shared by the evolving individuals with the guide set. For example, if Ω were the set of Mozart's symphonies, an individual with a high fitness should resemble a Mozart symphony.

1. The program generates a random population of 64 vectors of N pairs of integers. We are currently using for N the length of the first piece of music in the guide set. The first integer in each pair is in the [24,48] interval, the second in the [1,16] interval. Each vector represents a genotype.

2. The fitness of every genotype is computed as the distance to the guide set, measured by means of the normalized compression distance.

3. The 64 genotypes are ordered by their increasing distance to the guide set.

4. If the lowest distance is less or equal to the goal distance, the program stops and returns the notes in the corresponding genotype, paired with a function of the lengths of the guide piece(s) of music.

5. From the ordered list of 64 genotypes created in step 4, the 16 genotypes with least fitness/highest distance are removed (leaving 48), while the 16 genotypes with most fitness/lowest distance are selected. These 16 genotypes are paired randomly to make 8 pairs. Each pair generates another pair, a copy of their parents, modified according to four genetic operations. The new 16 genotypes are added to the remaining population of 48, to make again 64, and their fitness is computed as in step 2.

6. Go to step 3.

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.

3. Testing different recombination procedures

In this set of experiments, we tested the effect of changing the recombination procedure used by the

genetic algorithm. The following strategies were used:

- Strategy 1: given a pair of genotypes, (x_1, x_2 ... x_n) and (y_1, y_2 ... y_m), a random integer is generated in the interval [0, mín(n,m)]. Let it be *i*. The resulting recombined genotypes are: (x_1, x_2 ... x_{i-1}, y_i, y_{i+1} ... y_m) and (y_1, y_2 ... y_{i-1}, x_i, x_{i+1} ... x_n). This is the base case (the simplest recombination strategy).

- Strategy 2: given a pair of genotypes, (x_1, x_2 ... x_n) and (y_1, y_2 ... y_m), two random integers are generated in the interval [0, n] (let us call them i, j, i<j) and another two in the interval [0,m] (let us call them p, q). The resulting recombined genotypes are: (x_1, x_2 ... x_{i-1}, y_p, y_{p+1} ... y_{q-1}, x_j, x_{j+1} … x_n) and (y_1, y_2 ... y_{p-1}, x_i, x_{i+1} ... x_{j-1}, y_q, y_{q+1} ... y_m).

- Strategy 3: given a pair of genotypes, (x_1, x_2 ... x_n) and (y_1, y_2 ... y_m), four random ordered integers are generated in the interval [0, n] for each parent genotype. Each genotype is then cut into the five corresponding pieces, which are shuffled together (one of them is reversed). The genotypes of the progeny are obtained by concatenating five of the pieces in the shuffled set.

- Strategy 4: similar to the preceding one, but only three random ordered integers are used to divide the parent genotypes into four pieces, which are then joined, shuffled, and used (four at a time) to generate the genotypes of the progeny.

The one-point crossing-over strategy 1 has the property that the lengths of the parent genomes are invariant under recombination in the progeny. Since mutation also keeps the length of the genome, only fusion and elision change it. In fact, we did notice that fusion almost never leads to a fitter genome, while elision sometimes does, which means that the version of our genetic algorithm described in the previous section, which starts with a genome length copied from one of the target pieces of music, leads to genome lengths usually reduced by a little (not much) from their initial value. Strategies 2, 3 and 4, however, all lead to progeny genomes with lengths usually quite different from those of their parents (even when both parent genomes had the same length), which provides the population with a much larger genome length variety than strategy 1.

After performing several experiments we noticed that, at the beginning of the evolution, the second recombination strategy converges more quickly towards the target, but after a certain number of generations (usually between 150 and 200), the first and fourth strategies becomes better, while beyond about 500 generations after the beginning of the process the first strategy is clearly the best. Above 1000 generations, the first two strategies tend to converge, i.e. to obtain similar distances to the goal after the same number of generations.

This brought us to our fifth and sixth strategies, which are simple combinations of the four described above:

- In the first 150 to 200 generations, the algorithm uses the second strategy (the two point recombination procedure with four different crossing-over points between both parents). During all the remaining generations, the first strategy is used instead (i.e., the one point recombination procedure with a single crossing-over point for both parents).

- In the first 200 generations, the program uses the second strategy; between generations 200 and 500 it switches to the fourth strategy, and above 500 generations it uses the first strategy.

The results of the combined strategies are much better than those of any of the four strategies applied separately, as shown in table 1. It can be observed that the first mixed strategy reaches, in just 600 generations, target distances similar to those attained by the first two strategies in over 2500 generations. The improvement of the mixed strategies is therefore quite impressive. On the other way, the two mixed strategies attain comparable results.

Nr. of generations	Strategy 1	Strategy 2	Strategy 3	Strategy 4	First mixed strategy
1	0.930	0.930	0.930	0.930	0.930
100	0.782	0.766	0.807	0.791	0.766
200	0.734	0.710	0.756	0.744	0.697
300	0.714	0.692	0.740	0.712	0.676
400	0.702	0.692	0.722	0.704	0.659
500	0.690	0.689	0.722	0.704	0.648
600	0.681	0.683	0.716	0.704	0.643
1000	0.663	0.682			
1500	0.658	0.666			
2000	0.656	0.658			
2500	0.644	0.652			

Table 1. A comparison of the performance of five different recombination strategies.

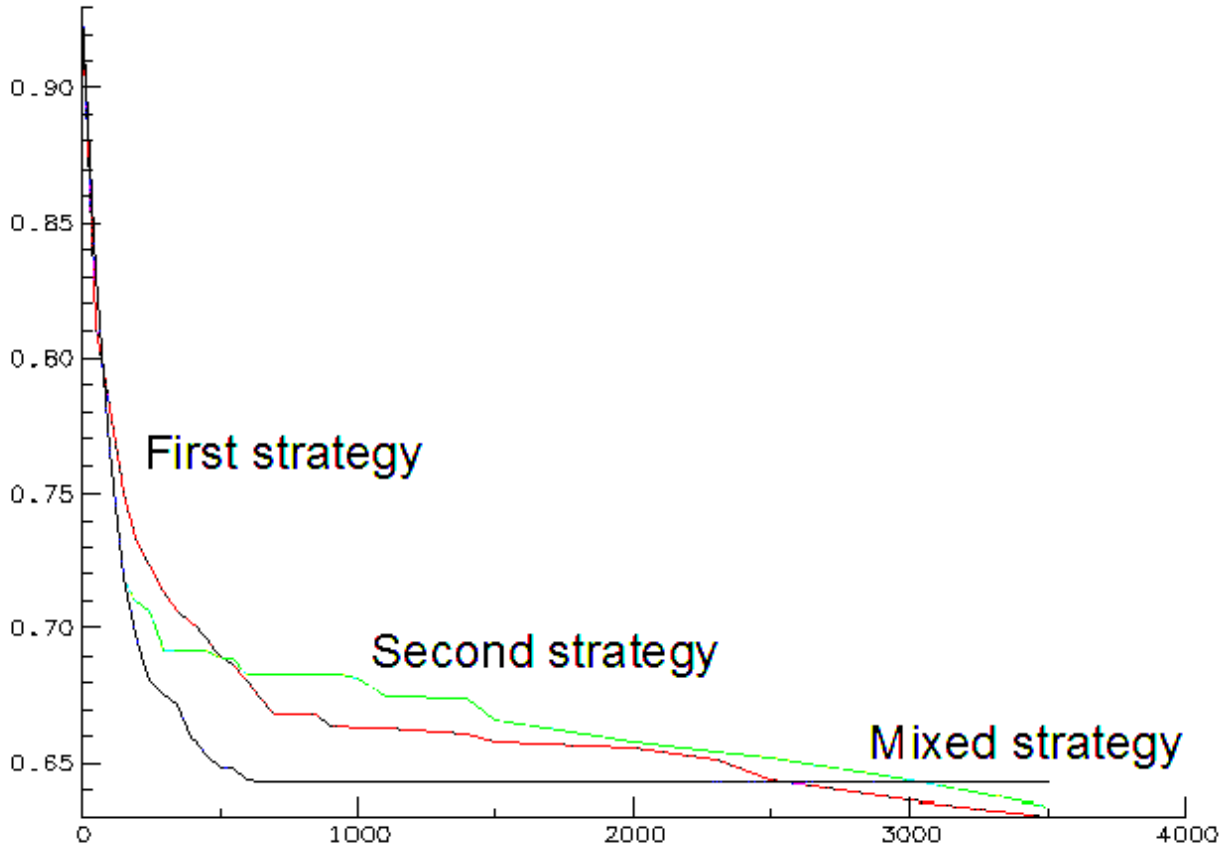

Figure 2. Comparison between three different recombination strategies.

Figure 2 shows a graphical representation of the results. Figure 3 shows the results of a different experiment with the same three strategies.

In our analysis of the reasons for this behaviour, we have come to the conclusion that, with the first strategy, the population reaches a smaller genetic variability, where favourable mutations have a greater probability of appearing. On the other hand, the second strategy generates a much greater genetic variability, both with respect to genome lengths and contents, where favourable mutations are much harder to come by. This means that, on the long range, the first strategy

should work better than the second, which on the other hand gets faster results during the first part of the process, by evolving simultaneously in many directions and testing widely different genomes at the same time. Thus, the mixed strategy makes the best use of both recombination procedures, which is the reason for its outstanding performance success.

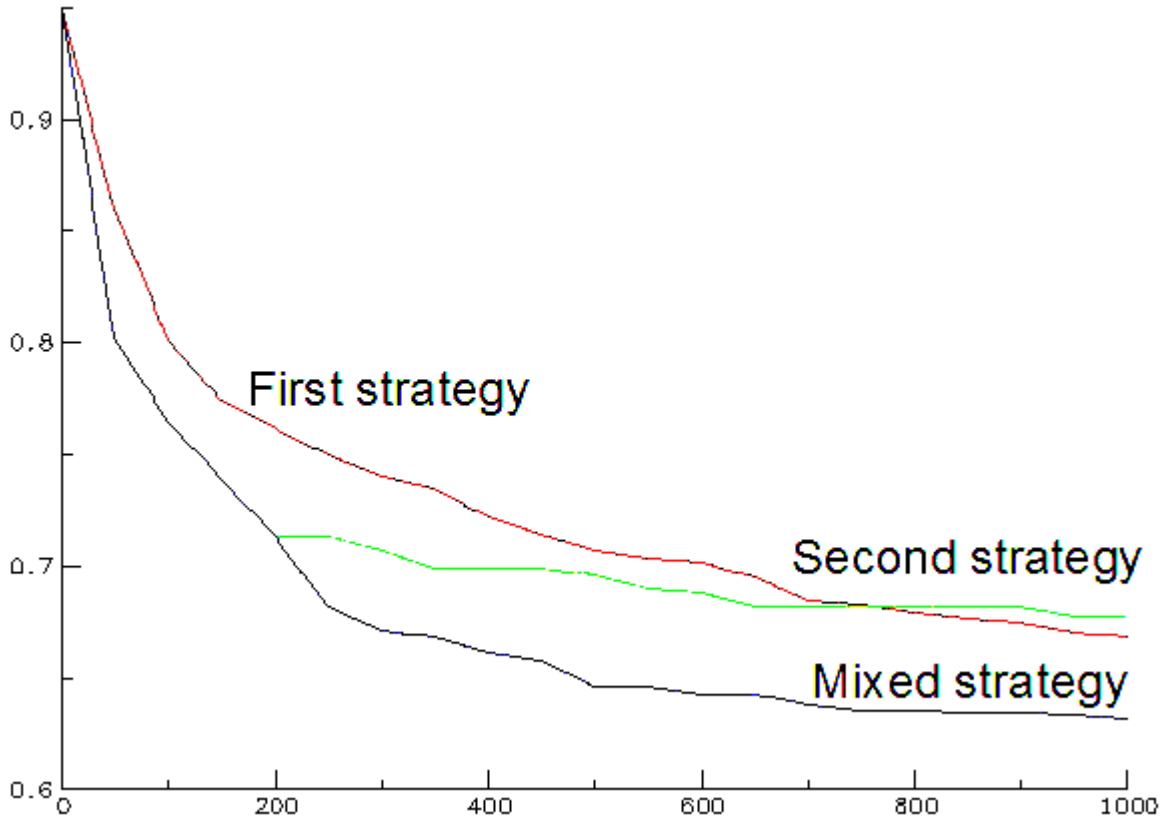

Figure 3. Performance comparison of another experiment with the same recombination strategies.

4. Heavy-tail distributions and automatic music generation

In a previous work on the automatic generation of fractal curves with a given fractal dimension [12], we proposed a procedure to make the genetic algorithm which we were using, in a grammar evolution context, increase its performance by about one order of magnitude. This procedure made use of the fact that the time needed to reach the goal in that case is not a normal distribution, but a heavy-tail one. Thus, a strategy based on stopping the algorithm and reinitializing it, when it has not reached an acceptable goal after a certain number of generations, gives rise to very good performance improvements.

With this procedure in view, we have analyzed the situation for the case of the automatic generation of music, but have come to the conclusion that, although it is possible that the distributions may still be heavy-tail, the performance improvement reached by applying the re-initialization procedure will be minimal, if any, because the minimum number of generations to reach an acceptable goal seems to be very large. This means that re-starting the algorithm does not provide us with a better chance of reaching an acceptable goal in a short time.

5. Conclusions and future work

We have found that the normalized compression distance is a promising tool to provide genetic algorithms for automatic music generation with a

measure of the distance to the desired target, which may be used as an appropriate fitness function. Some of the pieces of music generated by this program have a significant similarity to the style of well-known authors, in spite of the fact that our fitness function ignores the duration of the notes and takes into account only the relative pitch envelope. Our results have been much better than those we obtained previously with a different procedure and fitness function [13].

In the future we intend to combine our results with those of other authors [14-15], so as to use as the target for the genetic algorithm, not just one or two pieces of music by a given author, but a cluster of pieces by the same author, in this way trying to capture the style in a more general way. We also intend to modify the algorithm to use the information about note duration.

We shall also try to work with a more standard and richer system of music representation, such as MIDI.

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] M.Alfonseca, M.Cebrián, A.Ortega (2005). Evolving computer-generated music by means of the normalized compression distance, *WSEAS Transactions on Information Science and Applications*, Vol. 9:2, p.1367-1372, Sep. 2005.

[9] 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.

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

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

[12] M.Cebrián, A.Ortega, M.Alfonseca (2004). Acceleration of a procedure to generate fractal curves of a given dimension through the probabilistic analysis of execution time, in *Intelligent Engineering Systems Through Artificial Neural Networks*, Vol. 14, ed. C.H.Dagli, A.L.Buczak, D.L.Enke, M.J.Embrechts, O.Ersoy, pp. 265-270, ASME Press, New York, 2004.

[13] A.Ortega, R.Sánchez Alfonso, M.Alfonseca (2002). Automatic Composition of Music by means of Grammatical Evolution, APL Quote Quad (ACM SIGAPL), Vol. 32:4, p. 148-155, Jun. 2002.

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

[15] R. Cilibrasi and P. Vitanyi (2004). Algorithmic Clustering of Music, *Proc. Of the Fourth Intl. Conf. on Web Delivering of Music (WEDELMUSIC'04),* pp. 49-67, IEEE Computer Society, ISBN: 0.7695-2157-6.