Evolutionary Composition of Music with Learned Melody Evaluation

ROMAN KLINGER
Fraunhofer Institute for
Algorithms and Scientific Computation
Department of Bioinformatics
Schloss Birlinghoven, 53754 Sankt Augustin
GERMANY
roman.klinger@scai.fhg.de

GÜNTER RUDOLPH
University of Dortmund
Department of Computer Science
LS XI – Computational Intelligence
44221 Dortmund
GERMANY
guenter.rudolph@uni-dortmund.de

Abstract: We describe our approach for the automatic composition of monophone melodies on a user given chord sequence. For this purpose we use evolutionary algorithms for the music generation with automatically learned classifiers next to interactive evaluation as fitness functions.

Key–Words: computational intelligence, music, composition, classification, data mining

1 Introduction

The main motivation for developing a program for automatic composition is to attach importance on getting melodies that have something new: The program should simulate creativity so that it can be used for composition assistence. Similar to the work of Biles [2, 3] or Wiggins and Papadopoulos [10] we use an evolutionary algorithm. Biles uses an interactive method as evaluation function, Wiggins and Papadopoulos use weighted sums of numerical features values extracted from the melodies. The interactive approach has the disadvantage of requiring much time to evaluate melodies. Using weighted sums raises the question if that method maps the personal taste of music appropriately.

There have been some approaches to learn a fitness function, for example with neural networks, but without emphasizing creativity [5], so the generated melodies are not pleasing to the ear [4] or they are just not very interesting [8].

Our idea is to extract features [15, 16] on which a data mining algorithm can classify the melodies. That approach has the advantage of the possibility that the automatically generated classifier fits the user's taste and can classify the melodies fast.

Another important thing to point out is that the implementation is licensed under the Gnu Public Licence[1] so that everyone can try out the program and experiment with different parameters[2]. This is a special feature because most systems for automatic composition are closed source apparently.

The rest of the paper is organised as follows: At first we explain the evolutionary algorithm developed here. Then some of the variation operators are described and elucidated on some examples. The probably most important part of our work is the evaluation procedure for melodies which will be explained next. The last section is a conclusion and an outlook on future work.

2 Overview

An evolutionary algorithm [1] is an optimization scheme which works on a set on possible solutions, in our case on melodies. A visualisation is given in figure 1.

At first the set of solutions, also called population of individuals, is initialised. This is done by using statistical methods like Markov chains so that they are meaningful according to some musical laws. For that, the user specifies the chords on which the melody should be played. After that this set is altered by mutation and recombination of the individuals. Then the

[1] http://www.gnu.org/copyleft/gpl.html

[2] Download information can be found on http://www.romanklinger.de/musicomp/musicomp.html and on http://sourceforge.net/projects/musicomp

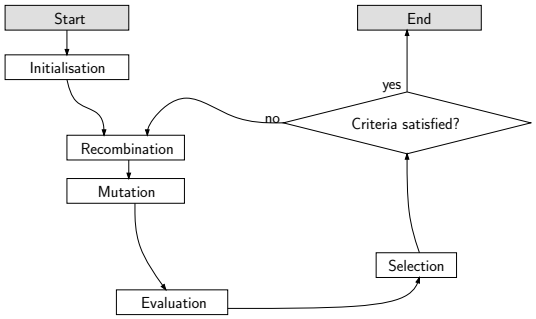

Fig. 1: Main components of an evolutionary algorithm.

original individuals and the altered ones are evaluated by some fitness function. The best ones, in our case the hopefully most pleasant melodies, form the successing population.

3 Mutation and Recombination

The operators *mutation* and *recombination* represent the methods to change the melodies so that they enhance their interestingness. Here we only give some examples because a description of the whole set of operators would exceed the size of the paper awfully. More details can be found in [9].

The mutation operators can be distinguished in those which change pitches (one-point-mutation, transposition, inversion), those which change rhythm (moving, merging, splitting of notes) and those which make some structural modification of the melody (rotating, sorting, mirroring of some range).

For the examples we assume figure 2 being the original melody. An example for the one-point-mutation could be the melody in figure 3 in which the second note is raised by one half step and the fifth tone is raised by one step. That method changes the pitch of every note with a low probability. The stepsize is determined by a bilateral geometric distribution [14].

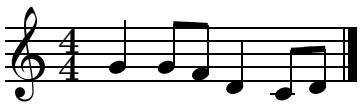

Fig. 2: Example of some melody prior to variation.

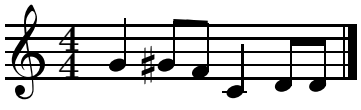

Fig. 3: One-point-mutation of melody in figure 2.

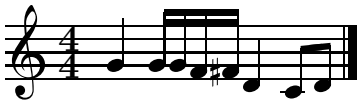

Fig. 4: Splitting some notes of melody in figure 2.

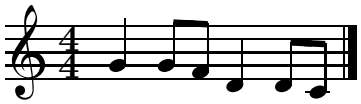

Fig. 5: Sorting of melody in figure 2.

An example for changing the rhythm is splitting every note with a low probability as we can see in figure 4. Here the second and the third note are split into two notes, each of half of the length of its original. The pitch of the second note is changed analogous to one-point mutation.

An example for a structural modification is sorting the notes downwards with respect to their pitches as we can see in figure 5. Here the whole melody is sorted. In our implementation only a randomly determined part of the melody is changed.

The recombination combines two parents to one or more new individuals. We experimented with intermediate methods which work by using the mean pitches of two notes on the same point in time of the two parents. Here the problem in using that kind of operator is that the melodies tend to a single tone repetition. So the better choice is using a one-point-crossover which builds two individuals by beginning with the first parent and ending with another and the other way round. The crossover point is determined randomly. An example is shown in figure 6.

4 Selection

The operator *selection* builds the successing population. We tried to use fitness proportional selection, but

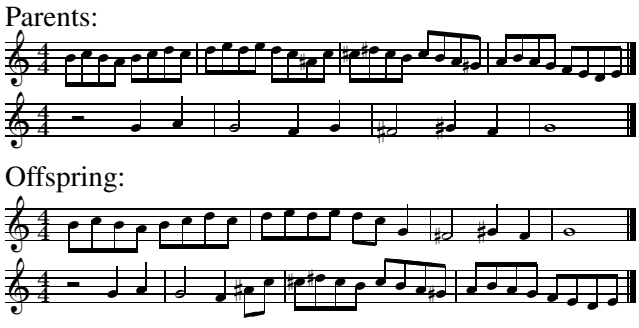

Fig. 6: Example for one-point-crossover.

this reduces the diversity of the individuals. It is nice not to have too similar individuals in the set because then it is more likely to have variations that could possibly fit the personal taste. Actually, it is crucial to maintain diversity in the population to provide sufficient potential for continuing evolution. Deterministic selection as used in evolution strategies works very fine, especially with niching methods to enhance the diversity.

We use two niching methods [1]. The first, *fitness sharing*, works by scaling down the fitness of similar individuals. The second, in our case much more successful, is called *crowding*. Here two parents are recombined to two children. The parents and children compete to each other in the pairing in which their similarity is higher. The function that gives the similarity between two individuals I_1 and I_2 with pitches m_i at points of time i is

$$sim(I_1, I_2) = 1 - \frac{\sum_{i=1}^{n} dist(m_i^1, m_i^2)}{h(I_1, I_2)}$$

with

$$dist(m_i^1, m_i^2) = \begin{cases} 0 & \text{if } m_i^1 \neq -2 \wedge m_i^2 \neq -2 \\ \min(|m_i^1 - m_i^2|, \Delta_{max}) & \text{otherwise} \end{cases}$$

and

$$h(I_1, I_2) = \sum_{i=1}^{n} a(x, y)$$

with

$$a(x, y) = \begin{cases} 0 & \text{for } x = y = -2 \\ \Delta_{max} & \text{otherwise} \end{cases}$$

For understanding the formula above it is important to know about our representation of melodies: A melody is a tuple $m \in \{-2, -1, 0, \ldots, 127\}^n$ where $i \in \{1, \ldots, n\}$ are points of time and the values $0, \ldots, 127$ represent the start of a tone with a note pitch according to the general midi specification[3]. The value -1 means "Holding the last event" and -2 starts a rest.

We set $\Delta_{max} = 4$, which means that the largest intervall between two pitches that is considered is 4. This also holds for an interval between a "-1" and the starting of a note with a given pitch in two individuals at the same point of time.

That function emphasizes the importance of the rhythm, so especially rhythmic features of the melodies are kept over the generations.

[3] http://www.midi.org/about-midi/gm/gminfo.shtml

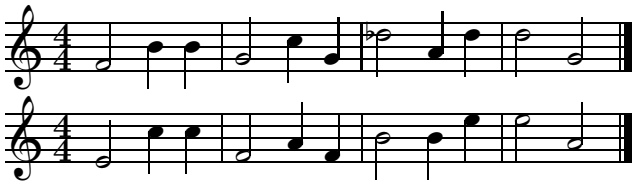

Fig. 7: Example for feature *Harmonicity*. The value of the first melody is 0, the one for the second is 1.

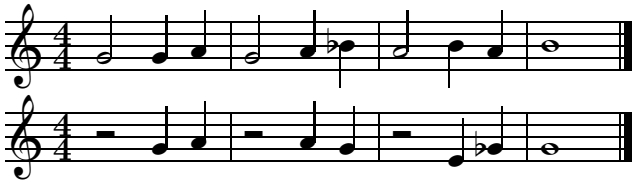

Fig. 8: Example for feature *Rests on Downbeats* with a value of 0 in the first melody and a value of 0.75 in the second one.

5 Evaluation

The evaluation function is a very important point in the generation of melodies with evolutionary algorithms. It "restricts" the creativity of the mutation and recombination operators. In addition to the interactive evaluation, which we also implemented in form of a slider the listener can move between 1 and 10 in steps of 0.01 after the melody was played that has to be evaluated, we implemented some methods based on feature extractions.

5.1 Feature Extraction

The feature extraction follows the work presented in [15, 16] with some additional methods. The features are subdivided into pitch features, tonale features, contour features, rhythmic features, pattern features, features for chord change and accentuation features. For explanation we give some examples which are all played on the chord sequence[4]: *Am, Dm, E, Am*

The feature *Harmonicity* gives the ratio between the number of notes with pitches of the current chord and the number of all notes. An example for two different melodies is given in figure 7.

The feature *Rests on Downbeats* determines the ratio between the number of downbeats and the number of downbeats on which there is a rest. In figure 8 are

[4] The chord Am is a set of the notes a, c, e. Dm is d, f, a. E is e, g#, b.

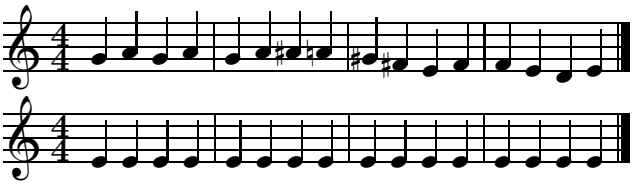

Fig. 9: Example for feature *Repeated Pitch* with a value of 0 in the first melody and a value of 1 in the second one.

two melodies with 4 downbeats: One with a value of 0 with no rests and one with 3 rests and a resulting value of 0.75.

The feature *Repeated Pitch* computes the ratio between the number of all intervalls with a size of 0 and the number of all intervals (= number of notes -1). In the example in figure 9 is one melody without pitch repetition (value 0) and one with all possible pitch repetitions (value 1).

5.2 Evaluation with Data Mining Methods

Typical problems of interactive evaluation are the long time required for listening to the melodies, the subjectivity and that this methods are not always reliable. So a nice idea is to use machine learning on the features mentioned before. We tried artificial neural networks and decision trees. For generating these we use a set of examples which were evaluated by a single person. It is composed of 45 well-known melodies with a majority of high evaluations, 136 automatically generated individuals (by saving all individuals of an evolution with interactive evaluation) and 24 outstanding unaesthetic individuals with very low fitness values. The melodies are given in MIDI-Format with an XML-File specifying the chords and the fitness.

5.2.1 Using Feed Forward Neural Networks

It is possible to select the features that should be used as input for the neural net. For every feature we use one input neuron and in every net one output neuron which gives the evaluation.

We experimented with neural networks with different structures and detected that when using all 42 implemented features it is reasonable to use a fully connected net with one hidden layer of 35 neurons. We decided to use resilient propagation [13, 7] for training which lasts only few minutes. Figure 10 illustrates the development of the total sum squared error (TSSE) for training the neural net. It is possible to reach a resubstitution error of 0.022.

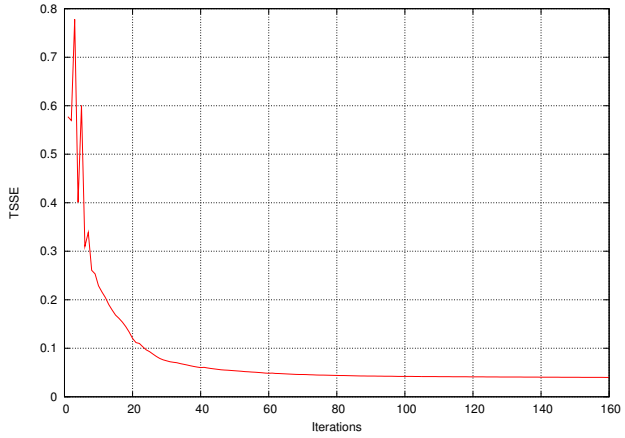

Fig. 10: Development of the total sum squared error (TSSE) of training the neural net.

5.2.2 Using Decision Trees

Neural networks are theoretically capable of approximating arbitrary functions, but the weights of the connections between the neurons are not intuitively interpretable. A very good approach for a better understandable classifier are decision trees that are built up in an inductive way [11, 12]. The algorithm we use is called C4.5 and is implemented in the Weka-Library [17] for Java. This algorithm deals with continuous attributes which correspond to our features but cannot handle regression. Because of that the fitness values have to be discretized. So the user specifies a number of classes in which the fitness values of the individuals in the example set should be reclassified.

An example for an automatically generated decision tree using only 2 fitness classes (0 and 10) so that the

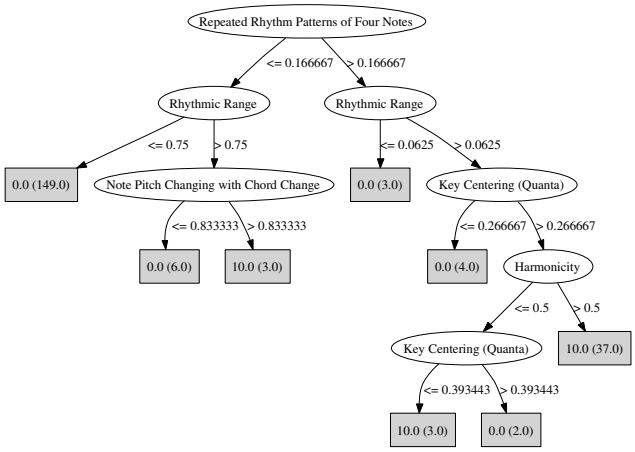

Fig. 11: Example for a decision tree for the classification of melodies.

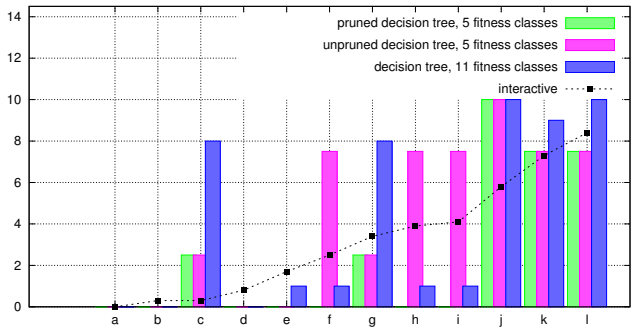

Fig. 12: Comparison of decision tree classifiers with interactive evaluation.

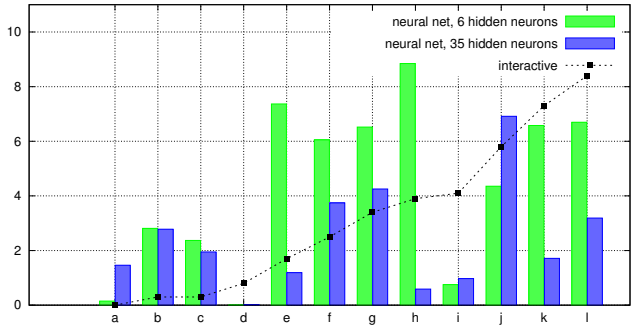

Fig. 13: Comparison of neural net classifiers with interactive evaluation.

tree is small enough to print it here is depicted in figure 11 (the numbers in brackets give the number of classified examples on the according leave).

Already this small example provides the facility for interpretation. If the feature *Repeated Rhythm Patterns of Four Notes* is very small and *Rhythmic Range* is also not very high the individual is classified as a bad melody. But if the *Rhythmic Range* is high and the *Note Pitch Changing with Chord Change* is also high it is classified as being a good one. Likely the following explanation holds: In the examples the chords are often changing with the bars. So if the rhythmic range is high there is a good possibility that the rhythm is confusing, but if there is always a note on the first beat in a bar that is not bad.

6 Conclusions

Since it is difficult to evaluate the quality of the automatic classification functions we used 10 different melodies and compared their automated classification with an interactive one. In figure 12 we see the comparison of some decision trees. The classification of the one with 11 fitness classes is identical using a pruned and an unpruned variant so there is no difference. The individuals a to l are sorted with respect to the interactive evaluation. The tree with 11 fitness classes makes some errors on the bad individuals and the unpruned one with 5 fitness classes is questionable in the middle fitness area. The pruned tree with 5 fitness classes leaves the best impression.

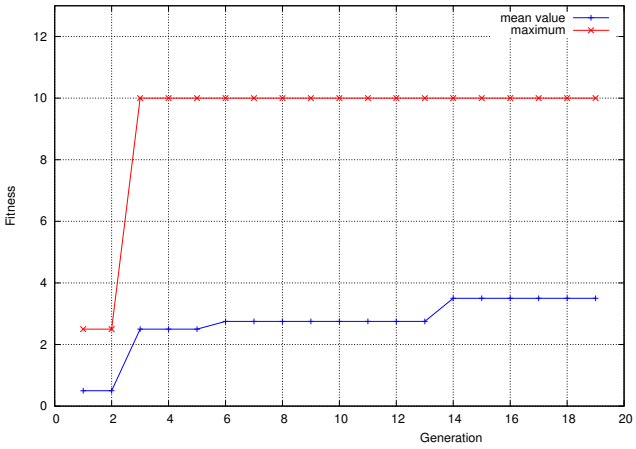

Fig. 14: Fitness development of an example evolution.

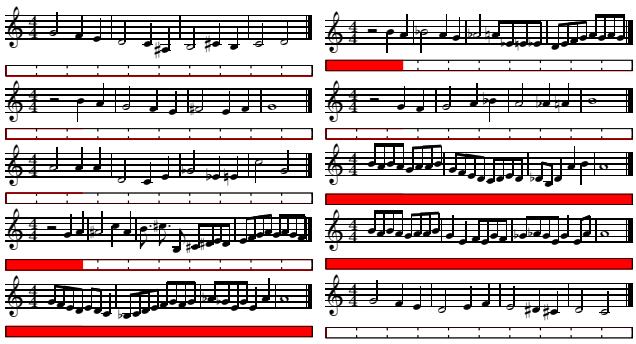

Fig. 15: Automatically generated melodies using a pruned decision tree with 5 fitness classes in 20 generations. The bars specify the automatically assigned fitness.

The comparison of the neural networks in figure 13 reveals that they are not very good in our configuration. Because of the very low TSSE it is likely that there is a problem with overfitting (see section 5.2.1).

An example for some automatically generated melodies is given in figure 15. The development of the fitness is presented in figure 14. Here for every generation the mean value of all individuals in the current population and the best fitness value in the current population is shown.

Based on this preliminary experimental study we conjecture that decision trees seem to be good for automatic classification on a comparative small number of example individuals. Neural networks are not con-

vincing and should be analysed on a larger training set. With our method it is possible to generate pleasent melodies in just a few generations of an evolutionary algorithm.

7 Future Work

One main task for future work is to generate a larger example set for analysing different methods for automatic classification in a more comprehensive manner. For that purpose it will be helpful to have many experiment participants for a not that subjective evaluation of the training melodies. Another point is the analysis of other similarity functions like the one mentioned in [6].

References:

[1] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, Bristol, UK, 1997.

[2] John A. Biles. Genjam: A genetic algorithm for generating jazz solos. In *Proceedings of the International Computer Music Conference (ICMC 1994)*, pages 131–137, San Francisco, USA, 1994. International Computer Music Association.

[3] John A. Biles. Genjam populi: Training an iga via audience-mediated performance. pages 347–348, San Francisco, USA, 1995.

[4] John A. Biles, Peter G. Anderson, and Laura W. Loggi. Neural network fitness functions for a musical iga. In *Proceedings of the Soft Computing Conference (SOCO 1996)*, pages B39–B44, Reading, UK, 1996. ICSC Academic Press.

[5] Anthony R. Burton. *A Hybrid Neuro Genetic Pattern Evolution System Applied to Musical Composition*. PhD thesis, University of Surrey, School of Electronic Engineering, Information Technology and Mathematics, Guildford, Surrey, England, 1998.

[6] Maarten Grachten, Josep-Lluís Arcos, and Ramon López de Mántaras. Melodic similarity: Looking for a good abstraction level. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, Barcelona, Spanien, 2004.

[7] Christian Igel and Michael Hüsken. Empirical evaluation of the improved Rprop learning algorithms. *Neurocomputing*, 50:105–123, 2003.

[8] Brad Johanson and Riccardo Poli. Gp-music: An interactive genetic programming system for music generation with automated fitness raters. Technical Report CSRP-98-13, Stanford University, University of Birmingham, 1998.

[9] Roman Klinger. Komposition von Musik mit Methoden der Computational Intelligence. Master's thesis, Department of Computer Science, University of Dortmund, Germany, June 2006.

[10] George Papadopoulos and Geraint Wiggins. AI methods for algorithmic composition: A survey, a critical view and future prospects. In *Symposium on AI and Scientific Creativity (AISB'99): Symposium on Musical Creativity*, pages 110–117, 1999.

[11] John R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[12] John R. Quinlan. Learning with continuous classes. In *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[13] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *Proceedings of the International Conference on Neural Networks*, San Francisco, USA, 1993.

[14] Günter Rudolph. An evolutionary algorithm for integer programming. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 139–148, Berlin, 1994. Springer.

[15] Michael Towsey, Andrew Brown, Susan Wright, and Joachim Diederich. Towards melodic extension using genetic algorithms. In A. R. Brown and R. Wilding, editors, *Proceedings of Interfaces: The Australian Computer Music Conference*, pages 85–91, 2000.

[16] Geraint Wiggins and George Papadopoulos. A genetic algorithm for the generation of jazz melodies. In *Proceedings of the Finnish Conference on Artificial Intelligence (STeP '98)*, Jyväskylä, Finnland, 1998.

[17] Ian H. Witten and Eibe Frank. *Data Mining*. Elsevier Inc., San Francisco, 2005.