GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters

Brad Johanson
Computer Systems Laboratory
Gates Building 2A-216
Stanford University
Stanford, California 94305
bjohanso@stanford.edu

Riccardo Poli
School of Computer Science
The University of Birmingham
Birmingham B15 2TT
R.Poli@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~rmp/eebic/WSC2/gp-music/gp_music.html

ABSTRACT

In this paper we present the GP-Music System, an interactive system which allows users to evolve short musical sequences using interactive genetic programming. We also present an extension which uses a neural network to model a users preferences, then stands in for them during the evolutionary process. The use of this 'automated fitness rater' allows the system to operate both with and without user interaction.

1 Introduction

The GP-Music System is an interactive system which allows users to evolve short musical sequences using interactive genetic programming (GP). Extensions to the system allow it to work in a fully automated mode. The basic GP-Music system works by using GP with a small set of functions for creating musical sequences, and a user interface which allows the user to rate individual sequences.

In the past ten years some researchers have tried to apply genetic algorithms, sometimes with neural networks as automatic raters, to the task of composing and creating music. Only one approach based on GP has been presented. We describe previous work in this area in Section 2.

An important aspect of the GP-Music System is that it is focused on creating short melodic sequences. It does not attempt to evolve polyphony or the actual wave forms of the instruments. Only a set of notes and pauses is created by the system. This narrow focus allows a reasonable musical sequence to be generated by a user serving as the fitness function during runs that last about 10 minutes and require a relatively small number of evaluations. The interactive GP-Music System is described in Section 3.

One main problem with the system is that the user must listen to and rate each musical sequence in every generation during a run. For long tunes this may be a long and (at least initially) tedious process requiring hours to complete even with small populations and short runs. To alleviate this problem, we extended GP-Music creating automatic fitness raters which stand in for the user in rating sequences. The user rates a small number of sequences in a short run on the GP-Music System, and the automatic rater uses the resulting ratings to learn to rate sequences in a similar fashion. The automatic rater then stands in for the user in longer runs of the system. These automatic raters, or auto-raters, are based on a neural network trained using back propagation. Auto-raters are described in Section 4.

Experimental results with the GP-Music system with and without auto-raters are described in Section 5. We draw some conclusions in Section 6.

2 Related Work

Cope (1987) created an expert system which was attuned to his own style of composition and was able to use it to create perturbations of a theme for use throughout a larger composition. Todd (1989) worked on a neural network which was trained to extract important features in music. Using a feedback system, the network is able to continue a composition based on previous notes. Mozer (1994) used a neural network system called CONCERT to predict note transitions based on learned style types.

Spector and Alpern (1995) came up with a GP system which evolved responses to call phrases in jazz pieces. As a fitness function they used a neural network trained on real responses from known jazz pieces, and tried to get the system to generate good responses. Biles (1994) designed the GenJam system which uses Genetic Algorithms. It evolves measures, and phrases simultaneously in a real time fashion using an interactive GA. Another system using GAs is Neurogen by Gibson et. al. (1991). It used a three stage approach where rhythm is first created, and then added in with melody, and finally combined with other phrases to create a harmony. Short GA strings are evolved for each of these stages, and a neural net is trained on

existing musical pieces, then used to rate the strings for the GA process.

3 The GP-Music System

This section outlines the main features of the GP-Music System which allow users to interactively evolve music. The system uses a modified version of the *lil-gp* GP system by Zongker, et. al. (1996).

3.1 Music Sequences

For storing the melodies created by GP-Music we used the XM, or extended module file format. XM files store note sequences synthesized at play time.

One of the features of Genetic Programming is that it can achieve good results without the need to explicitly specify a lot of domain knowledge for a problem. We therefore decided not to constrain the search of GP Music using music theory, but instead used the basic note features available in the XM format to determine the general structure of what was being evolved.

Specifically, the XM file allows a basic pattern size of up to 255 note events. Each note event can contain either a note to be played, or a rest command. The notes themselves come from the standard scale and include: C, C Sharp, D, D Sharp, E, F, F Sharp, G, G Sharp, A, A Sharp and B. The notes fall over eight octaves. A fourth octave D Sharp is noted as D#4. A single channel of melodic piano is used for all sequences, and all notes are currently of constant duration.

3.2 Function and Terminal Sets Description

The terminal and function sets for the GP-Music System are the notes which are used in the created melodies, and a small collection of routines to modify note sequences. They are summarized in Table 1, and are subsequently explained in more detail.

Table 1- The Terminal and Function Sets in the GP-Music System

Function Set:	play_two, add_space, play_twice, shift_up, shift_down, mirror, play_and_mirror	
Terminal Set:	Notes:	C-4, C#4, D-4, D#4, E-4, F-4, F#4, G-4, G#4, A-5, A#5, B-5
	Pseudo-Chords:	C-Chord, D-Chord, E-Chord, F-Chord, G-Chord, A-Chord, B-Chord
	Other:	RST

The Terminal Set

The terminal set used consists of notes in the 4th and 5th octaves available in the XM file format. The range of notes used was limited to one or two octaves (depending on the run) to prevent pieces with large pitch ranges from being too prevalent. In addition, the RST terminal is used which indicates one beat without a note.

Finally, there is a set of seven pseudo-chord terminals. Each of these is a sequence of three notes that follow the same pitch separation as a chord (an arpeggio). They are denoted pseudo-chords since they are played sequentially instead of simultaneously. See [5] for more details.

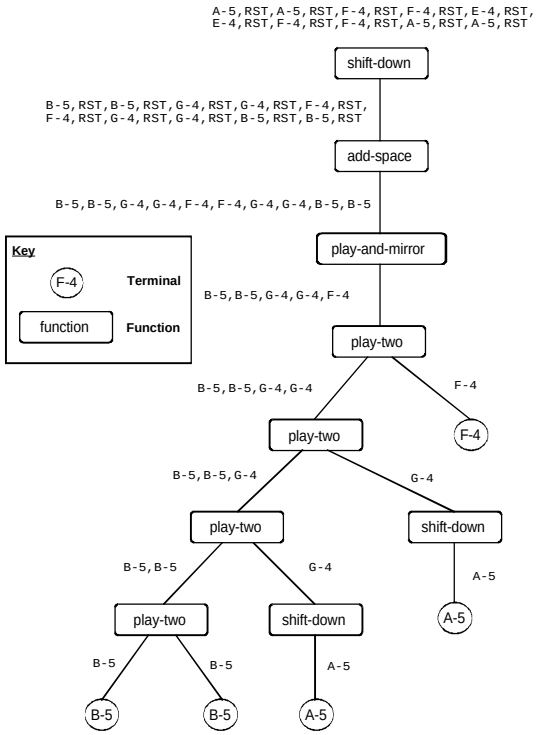

Figure 1- Example Music Program Tree Evaluation

The Function Set

The routines in the function set all operate on one or more note sequences which are passed to them. They perform some transformation on the note sequence (or sequences), and then return a new sequence. They will now be discussed individually.

- **play_two (2 arguments)-** This function takes two note sequences and concatenates them together. Along with the terminals, this function allows all note sequences which fall in the octave of the terminal set to be constructed.
- **add_space (1 argument)-** The note sequence which is passed to add_space has a rest inserted after each time slot in the original sequence. This has the effect of slowing down the tempo.
- **play_twice (1 argument)-** This routine plays the note string which is passed to it twice in succession.
- **shift_up (1 argument)-** Every note in the argument note sequence is shifted up to the next valid higher note. Notes which shift out of the eight octave range are clamped to the highest note value.

- **shift_down (1 argument)-** This function is identical to the shift_up routine except that the notes are shifted down.
- **mirror (1 argument)-** The argument sequence is reversed.
- **play_and_mirror (1 argument)-** The argument sequence is reversed and concatenated onto itself.

3.3 Interpreter

The item returned by the program tree in the GP-Music System is not a simple value but a note sequence. Each node in the tree propagates up a musical note string, which is then modified by the next higher node. In this way a complete sequence of notes is built up, and the final string is returned by the root node. Note also that there is no input to the program; the tree itself specifies a complete musical sequence. Figure 1 shows how a note sequence is built up.

It is worth noting that this representation provides more flexibility than the GA approaches used by others in previous research by allowing structure and variable length sequences.

3.4 Fitness Selection and User Interface

Since the suitability or quality of a musical piece is largely subjective, a human using the system is asked to rate the musical sequences that are created for each generation of the GP process. This is similar to Poli and Cagnoni's (1997) system for evolving pseudo-coloring algorithms.

The user rates the individual sequences using a simple 'list' style X-Windows interface. The principle of the list interface was to allow the user to change their mind about a sequence's rating after they have heard what the `competing' sequences sound like. The user rates each musical sequence on a 1-100 scale. The user interface is shown in Figure 2.

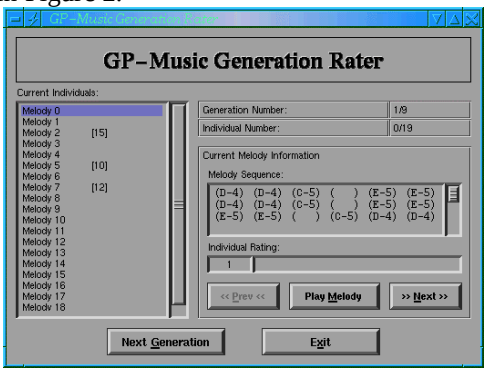

Figure 2- The GP-Music User Interface

3.5 Modifications to the Basic GP Algorithm

One of the problems with interactive GP applied to music is that ratings are subjective. The list interface helps the user to maintain a consistent rating scheme among sequences in a given generation, but not between generations. Since each generation was rated as a separate group, a user might always rate the best individual in a generation the same, despite the fact that overall the sequences were improving. One of the GP operators, however, is reproduction, where an individual is copied directly from the previous generation into the new one. If the copied sequence is presented to the user in a new generation it is quite likely to be rated differently from in the previous generation. To fix this problem, the system was changed so that an individual's rating is locked in from generation to generation. The list interface also allows the user to listen to the previously rated piece and see its rating before rating the new generation. This helps them to mentally recalibrate for the new set of sequences. Note that this locking in of fitness was added to assist the user in rating consistently, and is not actually necessary for the GP algorithm.

Another problem in early versions of the system was the creation of melodies that were so short or so long that they always garnered low ratings. Since the generation size is already small for user interactive genetic programming, having these unsuitable individuals in the population reduced the diversity and efficiency of the evolution process. The user can now choose a certain note sequence minimum and maximum length for a run, as well as a minimum and maximum number of rests. Individuals that don't meet the criteria are automatically destroyed, and a new individual is created until a satisfactory individual is found.

4 Automated Raters

Our automatic fitness raters are based on neural networks with shared weights trained with the back propagation algorithm. They give ratings on a 1-100 scale in a similar fashion to a human using the list interface.

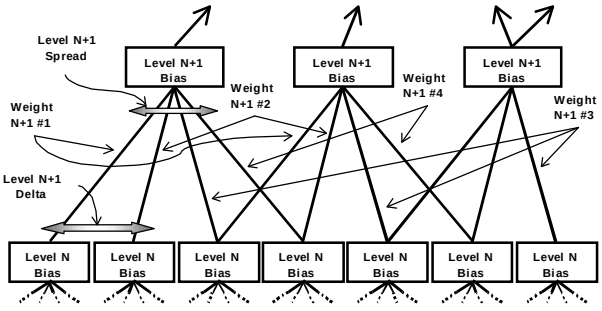

Figure 3- Basic Auto-Rater Network Layout

In normal back propagation networks, each connection into a node has its own weight which is modified by the back propagation training. In a network with shared weights, however, some of the connections use the same weight, and the weight will therefore be modified several times during the back propagation, once for each connection with which it is associated. The use of shared weights allows the rating of sequences of variable length,

which would be a very hard problem using standard neural network topologies.

The basic unit of topology for the network is shown in Figure 3. Level N is closest to the inputs (or possibly is the input layer), while Level N+1 is closest to the output node, or nodes (or possibly is the output layer). Each node in the upper level receives input from the lower level nodes. The value of the parameter 'Level Spread', in this case 4, determines how many nodes feed into one of the higher level nodes. The first node on a level receives input from the first 'Level Spread' nodes of the next lower level. The second node receives input from subsequent nodes, possibly receiving some of its inputs from lower level nodes also feeding into the first node.

The 'Level Delta' determines the amount of overlap between connections to adjacent nodes in the upper level. In the case of the diagram it is two, meaning that the first node receives inputs starting with the first lower level node, while the second node receives input starting with the third lower level node. Setting the 'Level Delta' to lower values increases the overlap, along with the ability for the higher level to correlate among nodes in the lower level, while increasing it causes each upper level node to act in a more autonomous fashion.

As mentioned earlier, each of the top level nodes has 'Level Spread' connections to lower nodes. The weights on these connections are all shared, so the weight on the first input to each upper level node is identical, and during back propagation the weights are modified according to the error coming back from each of the top level nodes. Note that the weights are used in a consistent sense with weight one always being used to connect the first lower level node to the upper level one (which also corresponds to connecting to a point earlier in the note sequence, as will be discussed below). The biases are also shared between all nodes on a given level, so in effect each node and its inputs are duplicate networks. The overall topology, shown in Figure 4, has five layers, including the input layer, and one output node.

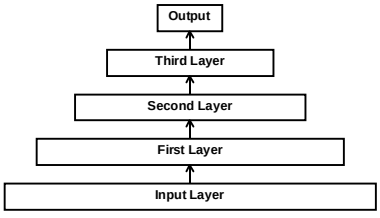

Figure 4- Global Topology for Auto Raters

Since the note sequences being rated are variable in length, a new network is built up for each individual that the network is required to evaluate or with which it is being trained. Consistency is maintained by storing the shared weights and biases and using them for each network that is built. The network is constructed by first creating one input node for each time slot in the sequence being evaluated. The value of the note at that point in the sequence is then loaded into the input node in the following manner:

- If the time slot contains a note, the value of the note divided by 72 is loaded into the node. The value of a note is taken to be 12 times the octave, plus the note value, where a C-0 is 1, a C#0 is 2, etc.
- If the time slot contains a rest (RST), a –1 is loaded into the node.

Enough first layer nodes are then constructed to exactly match the number needed given the first level spread and delta amounts. Once the nodes for the first level are created, the connections between the two levels are made as described above, using the shared weights for the first layer. The second and third layers are constructed in a similar fashion, and then each third layer node is connected to the output node of the network which outputs a value between 0 and 1. This is then multiplied by 100 to create the appropriate rating.

The construction of the network in this fashion allows it to telescope out to whatever the length of the input note sequence.

5 Experimental Results

This section documents the experimental work done with the GP-Music System and the auto raters. The basic parameters used for the experimental runs were similar to those suggested by Koza (1994). Individuals were selected for reproduction using 4 individual tournament selection. The genetic operators were crossover, reproduction, and mutation with probabilities 0.7, 0.15 and 0.15, respectively. Six generations were used with a generation size of 16. Initial tree depths were limited to between 1 and 4 levels, with a global maximum depth of 6 levels.

5.1 User Interactive Runs

Table 2- Base Line Trial Best of Run Individual

Gen.	Nodes	Depth	Seq. Len.	Fitness
3/5	28	6	26	40.00
Program Tree: (play-two (play-two (add-space (play-two F-4 B-5)) (add-space (play-two B-5 F#4))) (play-two (add-space (play-two (add-space F#4) (play-two B-5 (play-two F#4 D#4)))) (play-two (add-space (add-space E-4)) (add-space (play-two F#4 D#4)))))				
Web Site File Name: 2tune.au				

Base Line Trial

The first trial made was primarily to verify what the GP-Music System can generate using the minimal set of functions and terminals. The operators were 'play_two' and 'add-space.' The notes were restricted to one octave, and no pseudo-chords were allowed.

The sequences generated tended to not have much structure, and many of the individuals sounded poor. One best of run individual is shown in Table 2.

This particular individual sounds quite pleasant and was found in the 4th generation. This and the other sequences reported in the paper are available at http://www.cs.bham.ac.uk/~rmp/eebic/WSC2/gp-music/gp_music.html.

Complex Functions

The next step was to add in the more complicated functions in Table 1. The sequences generated during this trial were better overall than those of the Base Line trial. Adding the new functions seemed to smooth out the variation between the best and worst individuals. An example best of run individual is shown in Table 3.

All Functionality

The next trial involved the addition to the terminal set of the pseudo-chords and the limitation of the notes to the C-Major scale. In addition, the initial tree depths were set to 4 to 6 levels, and the maximum depth to 9 levels.

Table 3- Complex Function Set Best of Run Individual

Gen.	Nodes	Depth	Seq. Len.	Fitness
3/5	21	8	21	25.00

Program Tree:

```
(play-two (play-and-mirror B-5) (mirror (play-two (shift-up (play-two (mirror G-4) (play-and-mirror D-4))) (play-twice (play-two (play-twice F#4) (play-twice (play-two (shift-down A-5) (add-space F#4)))))))
```

Web Site File Name: 4tune.au

The effect of this trial was startling. Almost all of the generated individuals were pleasant to listen to. The only drawback is that some of the individuals sounded similar as they all tended to rely on the pseudo-chords. A typical best-of-run individual is shown in Table 4.

Table 4- All Functionality Best of Run Individual

Gen.	Nodes	Depth	Seq. Len.	Fitness
4/5	15	8	60	22.75

Program Tree:

```
(play-and-mirror (shift-down (play-two (play-and-mirror (shift-down (play-two (shift-up (add-space A-Chord)) (shift-up (add-space A-Chord))))) (shift-up (add-space A-Chord)))))
```

Web Site File Name: 9tune.au

This sequence uses the structuring of the complex functions and the pseudo-chords to its advantage, playing the 'A-Chord' backwards and forwards with an interesting stutter in the middle.

Training Run

An additional, longer run, was made to gather data for use in training the auto rater. The training data was gathered by running the GP-Music System over 10 generations with 20 individuals per generation. This led to the rating of a total of two hundred individuals. The maximum depth allowed was also increased to 12. In all other respects the computer was configured as for the 'All Functionality' trial. The human rater was Anne Pearce, a retired music teacher. The best individual generated by the run is shown in Table 5.

Table 5- Training Run Best Individual

Gen.	Nodes	Depth	Seq. Len.	Fitness
9/9	24	9	120	46.00

Program Tree:

```
(play-twice (play-two (add-space (shift-up (play-two (play-twice (play-and-mirror (shift-down (shift-up D-Chord)))) (play-two (play-twice G-Chord) (mirror F-Chord))))) (play-and-mirror (shift-down (shift-up (play-two (play-twice G-Chord) (mirror F-Chord)))))))
```

Web Site File Name: anne-list-best.au

This individual is quite nice, certainly longer, and perhaps superior to any of the others that we have generated. The tune sounds almost like some old sea shanty, although it ends quite abruptly. During this run the two hundred ratings and the associated sequences were captured for use to train the auto-rater.

5.2 Training the Auto Rater

The ratings generated during the human run were used to train the auto-rater network described in Section 4. The ratings were divided up into two sets of 100 individuals, one to serve as a training set and one to serve as a control set. For each training of the network, the individuals in the training set were repeatedly used to modify network weights and biases using back propagation. Statistics were kept during the training measuring the decimal error (the absolute value of the difference between the human and network rating on a 1-100 scale) for both the control and training sets. These measurements were made after each complete cycle through individuals in the training set.

In preliminary runs we found that the best parameters were: Level 1 Spread = 8, Level 1 Delta = 4, Level 2 Spread = 8, Level 2 Delta = 4, Level 3 Spread = 4 and Level 3 Delta = 2.

Statistics gathered during training show that the average error on the training set goes down to +/-5 after 1000 cycles, which is quite good when the rating is out of 100. The minimum average error on the control set reached a minimum of 7.16 at cycle 850, where training was stopped to avoid overfitting. This error seems quite good on a 1-100 scale. The cycle 850 weights were used for the auto-rater during later automated runs of the GP-Music system.

This analysis indicates that it is reasonable to train a network to rate sequences in the GP-Music System. Further details of our analysis are presented in [5].

5.3 Auto Rater Runs

The weights and biases for the auto-rater network trained for 850 cycles were used in several runs of the GP-Music System. The first run was made with the same parameters used during the human run that generated the training set data. The best individual (anne-200-best.au) was discovered in the seventh generation, indicating that evolutionary forces are coming into play. This individual actually sounds quite nice, although not as good as the one generated during the human generated run.

To evaluate how well the auto-rater works in larger runs, runs with 100 and 500 sequences per generation over 50 generations were made. The resulting best individuals were 'anne-5000-best.au' and 'anne-25000-best.au.'

Unfortunately, the 100 individual per generation sequence doesn't sound nearly as good as the one generated during the smaller run. It alternates between low and high note sequences at the beginning, and then diverts into a different style at the end.

The 500 individual per generation sequence is better than the previous one. It stays consistent during the length of the sequence, not changing pitch sequence or style. The sequence is quite strange, though, with only three different notes. Nonetheless, it is not unpleasant to listen to.

The three trials show that the auto-rater on its own is able to evolve interesting, and pleasant sequences in the GP-Music System, but not in a consistent fashion.

6 Conclusions

Our work so far with the GP-Music System has shown that it is possible to evolve reasonable short melodies using interactive genetic programming. The improvement in quality between runs using only simple concatenation, and those using more complex structuring functions shows that GP has advantages over genetic algorithms for this type of task. Nonetheless, there is a user bottleneck problem whether GA or GP is used. A user can only rate a small number of sequences in a sitting, limiting the number of individuals and generations that can be used. We addressed this problem with auto raters, which learn to rate sequences in a similar fashion to the user, allowing longer runs to be made. These proved somewhat successful, but the auto rater runs were not able to generate nice sequences with the reliability of human rated runs. Still, our research suggests that computers will be able to take a much more active role in computerized music composition in the future.

Acknowledgements

We wish to thank the members of the EEBIC Group at the University of Birmingham for their support. Special thanks are also deserved by Anne Pearce and Philip Underwood, who both participated in runs of the GP-Music System

Bibliography

1. Biles, J. A., "GenJam: A Genetic Algorithm for Generating Jazz Solos, " *Proceedings of the 1994 International Computer Music Conference*, ICMA, San Francisco, 1994.
2. Cope, D., "An Expert System for Computer-assisted Composition," *Computer Music Journal*, Vol. 11, No. 4, pp. 30-46, 1987.
3. Gibson, P. M., Byrne, J. A., "Neurogen, Musical Composition Using Genetic Algorithms and Cooperating Neural Networks," *IEE Conference Publication*, No. 349, pp. 309-313, 1991.
4. Johanson, B.E., "Automated Fitness Raters for the GP-Music System," University of Birmingham, Masters Degree Final Project, 1997.
5. Johanson, B.E., Poli, R., "GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters," *Technical Report CSRP-98-13*, School of Computer Science, The University of Birmingham, 1998.
6. Johanson, B. E., "The GP-Music System: Interactive Genetic Programming for Music Composition," University of Birmingham, Second-Semester Mini-Project Report, 1997.
7. Koza, J. R., *Genetic Programming 2: Automatic Discovery of Reusable Programs*, The MIT Press, Cambridge, MA., 1994.
8. Mozer, M.C., et al., "Neural network music composition by prediction: exploring the benefits of psychoacoustic constraints and multi-scale processing," *Connection Science*,. Vol.6, No.2-3, pp. 247-80, 1994.
9. Poli, R., Cagnoni, S., "Genetic Programming with User-Driven Selection: Experiments on the Evolution of Algorithms for Image Enhancement," *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Morgan Kaufmann, 1997.
10. Spector, L., Alpern, A., "Induction and Recapitulation of Deep Musical Structures," *Proceedings of the IJCAI-95 Workshop on Music and AI.*
11. Todd, P. M., "A Connectionist Approach to Algorithmic Composition," *Computer Music Journal*, Vol. 13, No. 4, pp. 27-43, 1989.
12. Zongker, D., Punch, B., Rand, B., *lil-gp 1.01*, Michigan State University, 1996.