Automated Fitness Raters for the GP-Music System

by
Brad Johanson

Abstract

In previous work, the basic GP-Music System was constructed which allowed human users to evolve short melodies using interactive Genetic Programming. For this project, the basic GP-Music System was improved, and automatic rating of melodies was implemented. The automatic rating is accomplished by automated fitness raters, or auto-raters, which are neural networks with shared weights. The auto-raters are trained using back propagation on a training set composed of sequences and their human assigned ratings. This data was generated during runs of the GP-Music System. Two types of auto-rater were created, one which assigns a 1-100 ranking, and another which indicates which of two sequences is better. The 'ranking' auto-rater was able get within 7 of the human rating, and generated some pleasant sounding melodies when substituting for a human in GP-Music runs. The 'comparative' auto-rater was never able to get more than 60% accuracy in determining which of two sequences was better and did not generate very satisfactory melodies during GP-Music runs.

Final Project
Supervisor: Dr. Riccardo Poli
University of Birmingham
15 September, 1997

Online Version:
http://www.cs.bham.ac.uk/~bej/gp-music/gp-music.html

This page left deliberately blank.

Table of Contents

List of Figures

List of Tables

1. Introduction

During the Spring of 1997, the GP-Music System was constructed as a mini-project at the University of Birmingham. It allowed users to evolve short musical sequences using interactive genetic programming. The project showed that it was possible to evolve musical sequences with genetic programming and achieve reasonable sounding melodies as a result.

One main problem with the system, however, was that it required users to listen and rate each musical sequence in every generation during a run. Even though the sequences were at most one minute long, rating 16 individuals over 5 generations (a bare minimum needed for successful evolution) could take up to one hour. For the evolution process to work better much larger populations over more generations were needed, but this was impractical with a human having to rate each sequence.

To alleviate this problem, the goal of this project was to create automatic fitness raters which could stand in for the user in rating sequences. The user would rate a small number of sequences in a short run on the GP-Music System, and the automatic rater would use the resulting ratings to learn to rate sequences like the user. The automatic rater could then stand in for the user in longer runs of the System.

To this end, two types of automatic raters, or auto-raters were constructed. Both are based on neural networks, and are trained using back propagation. One of the raters assigns to note sequences a rating on a 1-100 scale, while the other looks at two note sequences and indicates which is better. A variety of different parameters for the two networks were used to discover the best configuration for each auto-rater, and these best configurations were then trained and used to evolve sequences.

The remainder of the report is divided up into nine sections. They are as follows:

- Section 2 discusses related work by other individuals.
- Section 3 gives background information on some of the techniques and software systems used in this project.
- Section 4 gives an overview of the original work done on the GP-Music System.
- Section 5 provides a description of the code created for the project, and indicates what each source file does.
- Section 6 describes improvements made to the original GP-Music System to facilitate construction of the automatic fitness raters.
- Section 7 describes how the automatic fitness raters are put together.

- Section 8 gives the results of the research into how best to configure the automatic fitness raters, and gives examples of sequences evolved with the trained auto-raters.
- Section 9 has discussion of issues that arose during the project and gives ideas for future research.
- Section 10 is the conclusion.

2. Related Work

Since the first computers were created, people have been using them to create and compose music. In the past twenty years there have also been programming languages created for music composition, and in the past ten years some have tried to apply Genetic Algorithms, sometimes with neural networks as automatic raters, to the task of composing and creating music. This section outlines some of the work done in these areas.

2.1 General Attempts

Using Genetic Programming for music composition entails a programming language for the same. Several people have created robust languages for music composition. One example is Pla, developed by Bill Schottstaedt in the early 1980s [13]. The language is quite complex, and has many special features to deal with the low level sounds created by instruments, and it has other features for dealing with polyphony. Unfortunately it does not have many low level structuring features which would have been useful as a model for GP-Music. John Rahn's Lisp Kernel system [11] also has a basic language for composition, but is more concerned in integrating music synthesis, acquisition and composition across various hardware devices and software packages attached to a given computer.

Also of note as AI based attempts at music composition are Cope's EMI [5] and Todd's Connectionist Approach [15]. Cope 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 worked on a neural network which was trained on pieces in the hope that it would extract some ideas of important features in music. Using a feedback system, the network is able to continue a composition based on previous notes.

2.2 GA Related Attempts

Several attempts have also been made to apply Genetic Algorithms and Genetic Programming to music composition. Spector and Alpern [14] came up with a GP system which evolved responses to call phrases in Jazz pieces. As a fitness function they trained a neural network on real responses from known jazz pieces, and tried to get the system to generate good responses. Unfortunately it was not very successful since the neural network did not have an adequate amount of training on poor responses. Biles [3] designed the GenJam system which uses Genetic Algorithms. It evolves measures, and phrases (which are a series of measures) simultaneously in a real time fashion. As the user listens to a stream of phrases and measures, they type ‘b’ for bad or ‘g’ for good. The accumulation of ‘g’s and ‘b’s serves as the fitness for the measures. This is interactive GA, and the author reports that although some decent phrases eventually begin to emerge, the process is tedious. A final system using GAs is Neurogen [6]. 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. The authors report that initial results are promising, but do not present any derived music.

2.3 Other Interactive GP/GA Systems

Other systems have also used interactive GA or GP with a user serving as the fitness function. Such a system was used by Poli and Cagnoni [10] to evolve methods for coloring MRI scans of the brain in ways that best emphasized key features. Some attempts were made to provide a model of the user to reduce the burden of having to rate images, but they were not very successful.

3. General Background

In this section a background is given for various things that were used in the course of this project. Specifically, genetic programming and neural networks are discussed. The Lil-GP system and the XM music format which were used in the GP-Music application are also described.

3.1 A Brief Description of Genetic Programming

It is assumed that the reader has some knowledge of Genetic Algorithms and Genetic Programming, so only a brief description is provided here. In both techniques an initial random population of potential solutions to a problem (“chromosomes”) are created. In GA systems, the chromosome is a fixed length string, while in GP, introduced by Koza [9], the chromosome is instead a tree of functions and arguments. A fitness is assigned to each solution which reflects how well it solves the problem at hand. A new population of users is then created from the old one using the operators of crossover, reproduction and mutation, each of which favor the individuals which have the highest fitness. The three operators work in the following ways:

- *Crossover-* sections of the two chromosomes are swapped, creating two new individuals for the next generation.
- *Reproduction-* an individual is copied from the current generation to the next.
- *Mutation-* a section of the chromosome is randomly changed

This process is then repeated for the next generation, and continues until either a suitable solution is found, or until a pre-determined number of generations passes.

Using this technique, Genetic Algorithms and Genetic Programming can be used to search a problem space for valid solutions. Evolutionary forces come into play to create individuals with greater fitness as time goes by. One of the nice things about this method is that very little domain knowledge for the problem needs to be used. As long as a measure of the fitness of a solution can be determined, a very general definition of what a solution looks like is all that is needed. This is the principle the GP-Music System uses to create musical sequences without applying a lot of music theory.

3.2 Neural Networks Overview

As with Genetic Programming, it is assumed that the reader has some knowledge of neural networks, so only a brief overview is provided here. A neural network, or connectionist network, is an abstract network that is designed to work in a method similar to the brain. A network is composed of nodes that are designed to function like neurons in the brain. Each neuron integrates over incoming signals to determine an output signal. The network is usually constructed in several layers, with outputs from one layer of nodes serving as inputs to the next layer. The inputs to first layer nodes are signals derived from, or directly corresponding to some input pattern, and the output from the final layer node, or nodes, is the output corresponding to that pattern. Frequently the networks are used as classifier systems, in which case the input might be a word, and the output the appropriate phonemes used to pronounce the word.

The networks used for the auto-raters for this project are back propagation networks, which refers to the method used to train the network to give the appropriate output for a given input. In this type of network, each input to a node, or neuron, is multiplied by a weight, and the sum of the weighted inputs is added to a bias, and is then passed through a sigmoid function[1] to determine the output of the network. Training of the network begins by initializing all of the weights to and biases to random small values. Then, one by one, patterns from a training set are presented to the network. The output value of the network for the training pattern is then compared to the desired result, and the error is distributed back through the network in proportion to how much each part of the network is contributing to the error. That error amount is then used to adjust all of the weights and biases such that the error for that input is reduced. The training set is then presented repeatedly to the network, one pattern at a time, with each set of presentations referred to as a learning cycle. This continues until the error has been minimised, or another suitable stop criterion is met. This other criteria may be to verify the network error rate is also low on patterns not in the training set, in order to avoid adjusting the network to such a high degree that it will only recognize patterns in exactly the same form as in the training set (this is known as overfitting).

More information on back propagation neural networks at an introductory level can be found in [1].

3.3 The Lil-GP Programming System

The Lil-GP Programming system was used as the basis for the GP-Music System. It is a set of C libraries created by Douglas Zongker, Bill Punch, and Bill Rand at Michigan State University to perform Genetic Programming tasks [17]. One of its main features is that it evolves the program trees as trees of C-function pointers, which means that the trees can be executed rapidly to evaluate their fitness. This was not a big factor for the GP-Music System, since the bottleneck in this case was the time it takes a user to rate a sequence. The most important feature was the ease with which new GP problems could be implemented using lil-gp. Creating a new problem required only that C-code be written for the functions and terminals, and that a fitness evaluation function be implemented. This allowed the basic GP-Music System to be implemented in a straightforward manner, although subsequent changes to the lil-gp code were required for some of the features of GP-Music.

[1] Or similar continuously differentiable function.

3.4 The XM File Format

Rather than designing a new file format for storing the melodies created by GP-Music, an existing music file format was chosen—the XM, or extended module format. XM files store musical pieces rather than straight digital audio. In other words, note sequences are stored then synthesized at play time, rather than storing a recording of the music. The MIDI (Musical Instrument Digital Interface) format was also investigated, but requires extra hardware for playback on most workstations, as it only contains a series of notes and directions on which instrument to use in playing them. Typically the information from the MIDI file is sent to an a synthesizer attached to the computer. The XM format, on the other hand, stores digital samples of the instruments to be used in playing each of the notes, and can thus be played on any workstation with digital audio output capabilities. There is also software available to play back XM files in DOS, Windows, OS/2, Macintosh, and most UNIX workstations, so there was no need to spend time on writing a music synthesis program.

One of the nice things about Genetic Programming is that it can frequently achieve good results without the need to apply a lot of domain knowledge in the specification of the problem. Rather than researching a lot of music theory for the project, I used the features available in the XM format to determine the general structure of what was being evolved using the Genetic Programming. Specifically, the XM file allows a basic pattern size of up to 255 note events (only one pattern is used in the GP-Music System). 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, and the notation for this is to append the octave number, from 0 to 7, after the note. Thus, a fourth octave D Sharp is noted as D#4.[2]

4. The GP-Music System

This section describes the GP-Music System, and gives an overview of the work done prior to this project. Previous work was done during the Spring 1997 Semester at the University of Birmingham, and is discussed fully in [7].

---

[2] Note that the octaves are numbered from zero, so octave four is actually the fifth octave.

4.1 System Overview

4.1.1 Basic Functionality

The GP-Music System uses user interactive Genetic Programming to evolve short melodies. A small set of functions for creating musical sequences was constructed, and a small program was created to allow a user to rate individual sequences. By rating 16 individuals over a course of 6 generations some reasonable if not stellar short melodic pieces were created.

A key feature of the GP-Music System is its focus on using Genetic Programming instead of Genetic Algorithms. Since GAs use a byte string to represent a problem, in this case a musical sequence, they are forced to a fixed length, and cannot easily apply functions to add structure to a sequence. With Genetic Programming it is possible to add functions so that simple musical structures, such as phrase repetition, are available within the individuals being evolved. GP also has the advantage of allowing variable length sequences to be generated.

Another 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 during runs that last about 10 minutes and require a relatively small number of evaluations.

4.1.2 Music Sequence Details

The GP-Music System only evolves a simple melodic sequence. This contrasts with many of the other systems (see Section 2.2) which try to evolve either longer pieces, responses, or multi-part musical pieces. The basic structure of the sequences is taken from what the XM format allows in a note sequence (see Section 3.4). A single channel of melodic piano is used, with one pattern of 255 time slots. Each of the beats can be either a note or a rest (RST). The notes fall in the range of C-0, C#0, D-0, up to G#5, A-6, A#6, and B-6.[3]

4.1.3 Function and Terminal Sets Description

Any problem being addressed by Genetic Programming needs to define both terminal and function sets to be used in creating and evolving the solutions to the problem at hand

[3] Only a single octave was used in any of the GP-Music runs, however.

(although musical sequences cannot really be thought of as ‘solutions’). 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. The function and terminal sets are summarized in Table 1 below, and are subsequently explained in more detail.

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

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

The Terminal Set

The terminal set used consists of notes in the 4^{th} and 5^{th} octaves available in the XM file format. The range of notes used was limited to one octave in the original implementation to prevent pieces with large pitch ranges from being to prevalent. In addition, the RST terminal is used which indicates one beat without a note.

Pseudo-Chord	Corresponding Note Sequence
C-Chord	C-4,E-4,G-4
D-Chord	D-4,F#4,A-5
E-Chord	E-4,G#4,B-5
F-Chord	F-4,A-5,C-5
G-Chord	G-4,B-5,D-5
A-Chord	A-5,C#5,E-5
B-Chord	B-5,D#5,F#5

Table 2- Pseudo-Chord Note Sequences

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. They are denoted pseudo-chords since they are played sequentially instead of simultaneously due to the sequential nature of the melodies being evolved. The exact notes used in each of the pseudo chords are listed above in Table 2.

The Function Set

The routines in the function set all operate on one or more note sequences which are passed to the routine. They perform some transformation on the note string (or strings), and then return a new string. Since each of the routines in the function set performs a different operation, 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 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 lower note.[4]
- **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. In other words, the series is played and then played backwards.

The functions in the set were chosen for various reasons. 'Play-two' was included as a necessary function to allow sequences longer than one note. The other functions were added to allow structured note sequences to become more common, and were added as the GP-Music system evolved. The functions were designed to mimic common musical features, although, partly out of a desire to restrict the amount of domain specific knowledge applied to the composition process, very little actual music theory was used.

4.1.4 The Tree Structure

As in all Genetic Programming, the individuals being evolved are program trees composed of functions and terminals. In the GP-Music System, the functions and terminals used are those discussed above in Section 4.1.3. Unlike in many other GP examples, the item returned by the program tree is not a simple value. Instead, the program tree creates a note sequence. The programs created are typically printed in a LISP-like fashion, and look something like the following:

```
(shift-down (add-space (play-and-mirror (play-two (play-two (play-two (play-two B-5 B-5)
                                                                        (shift-down A-5))
                                                              (shift-down A-5))
                              F-4))))
```

Figure 1- Sample Music Program Tree

---

[4] If the C-Major Key is being used, the note is shifted to the next higher whole note.

When evaluated, the program shown above in Figure 1 generates a string of notes. 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 unlike most of the programs created in GP applications, there is no input to the program; the tree itself specifies a complete musical sequence. Figure 2 below shows how the note sequence is built up, using the program in Figure 1 as an example.

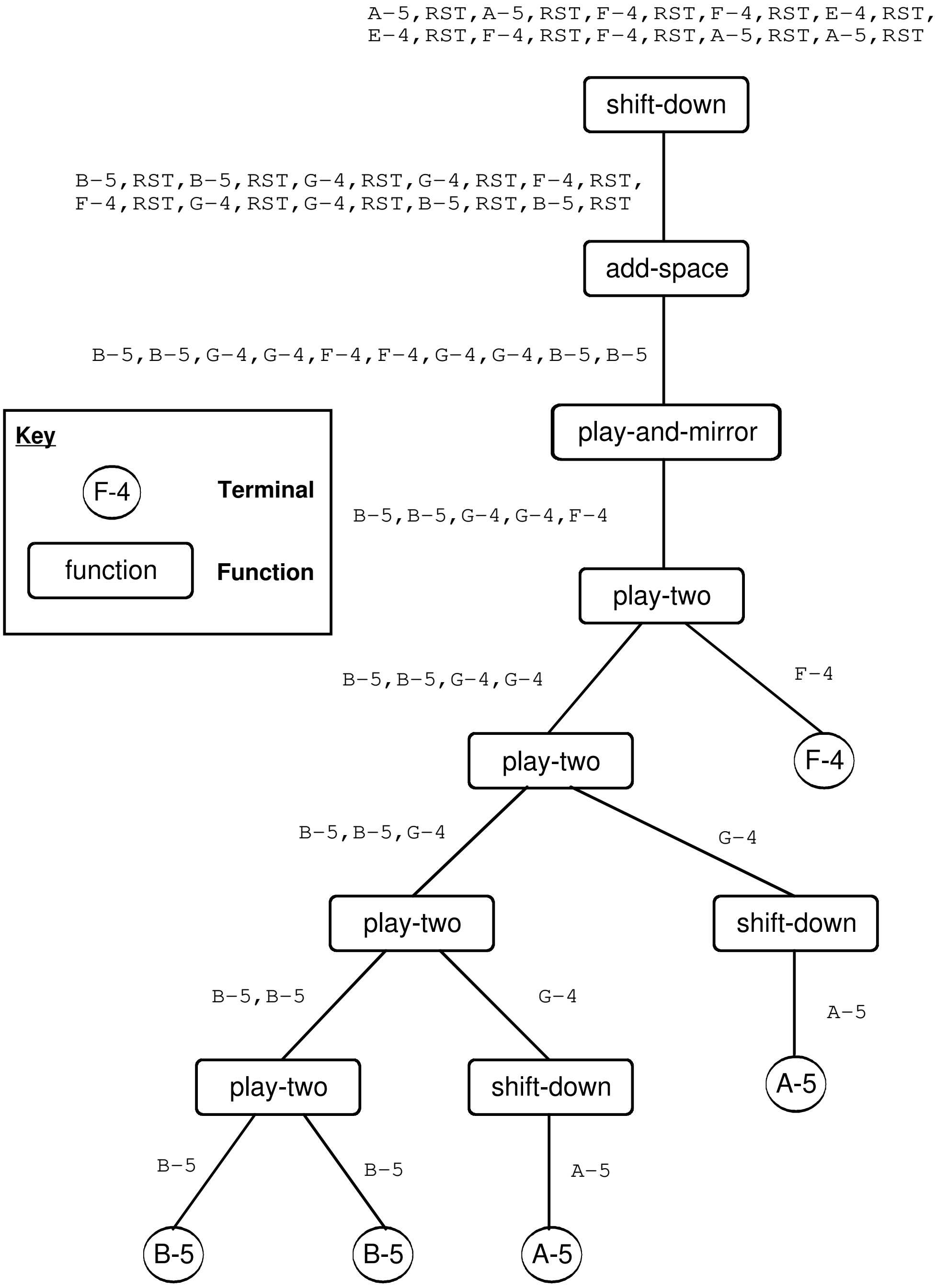


Figure 2- Example Music Program Tree Evaluation

The above figure shows the note strings being passed up from each node. Also, note that the 'shift-down' functions in the example shifts down to the next whole note. This is because the sequence in question was generated with the C-Major key feature turned on and only the seven whole notes are being used.[5]

Now that an example of how the program tree works has been presented, it is worth noting how this technique provides more flexibility than the GA approaches used by others in similar research (see Section 2.2). In the plain GA approach, fixed length byte strings are evolved, rather than a program tree. This fixes all sequences at a constant length, which restricts the domain of the generated note sequences. More importantly, however, the sequences are inherently unable to take advantage of any structure, since the only thing coded in the strings are the sequence of notes. Using Genetic Programming, however, the functions to create a musical sequence are stored. If the note terminals and the function 'play-two' are the only ones used, you get the same effect as using GA, since you are only evolving sequences, and no structure is imposed.[6] With GP, however, it is possible to go beyond this and add structure applying functions which are known to be pleasing to the ear. This allows the domain of the generated note sequences to be restricted somewhat so that the GP search is more efficient, which is especially important with the smaller populations mandated by the interactive Genetic Programming used in the GP-Music System. Overall this makes GP a more flexible approach to evolving musical note sequences.

4.1.5 Parameter Specification

General parameter specification was done using the facilities provided by the Lil-GP system [17]. Default values were chosen for each parameter, and these were then modified during some of the program runs to determine their effect on the resulting musical sequences. The default was to have 6 generations of 16 individuals. Individuals in new generations were created using the crossover, mutation and reproduction operators, with 70 percent of new individuals being generated using crossover, and 15 percent being created with each of mutation and reproduction. The default values used are fairly standard, and approximately mimic those suggested by Koza in his second book [8].

---

[5] This is discussed further in Section 4.2.

[6] Even in this instance, however, you have the benefit of variable length sequences when compared to using GA.

4.1.6 Fitness Selection and the GP-Music Interface

Since the suitability or quality of a musical piece is largely subjective, it is not possible to use a strict mathematical function to assign a fitness to individual note sequences which are generated. Instead 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's [10] system for evolving pseudo-color image algorithms, discussed in Section 2.3.

Currently the system lets the user rate each musical sequence on a 1-100 scale. A simple user interface in X-Windows was created to facilitate this rating. The interface is shown in Figure 3.

Figure 3- The GP-Music User Interface

The interface was designed to be very simple. After each generation is created, the user is asked to rate each of the individuals in the generation sequentially. The program automatically plays the given note sequence (which is converted to an XM file and played by an auxiliary player program), after which the slider can be used to assign a rating/fitness to the individual. Pressing the 'Re-Play' button plays the piece again. Once the piece has been rated, clicking on the 'Next Song' button plays the next sequence. Clicking 'Exit' allows the user to abort the current GP run.

4.2 Overview of Previous Results

During the previous work, a series of program runs were conducted, and the results of each run were used to tune the system to produce better output sequences. Six different trials were made, including a Base Line trial using default parameters, a trial using automatically defined functions, and a trial including the Pseudo-Chord terminals. Each trial, and observations made during the trial are summarized in the remainder of this section. Since it is difficult to appreciate the sequences generated during the trials by reading a series of notes, AU format sound clips of the sequences generated are also available on the web at:

http://www.cs.bham.ac.uk/~bej/gp-music/gp-music.html

Base Line Trial

The first trial made was primarily to verify that the system functioned correctly. It also provided sample sequences that are characteristic of what the GP-Music System can generate using the minimal set of functions and terminals. The parameters chosen were as simple as possible, as this trial was designed mainly to prove the viability of the GP-Music system. The main power of GP, the ability to use functions to provide structure to the sequences was not really tested in this trial. The only operator was 'play-two', which allowed a random sequence of notes to be built up, and the 'add-space' operator which allowed changes in tempo. The only advantage of GP over GA being used in this trial was the ability to have variable length sequences.

Note Sequence: F-4, RST, B-5, RST, B-5, RST, F#4, RST, F#4, RST, RST, RST, B-5, RST, F#4, RST, D#4, RST, E-4, RST, RST, RST, F#4, RST, D#4, RST
Web Site File Name: 2tune.au

Table 3- Base Line Trial Note Sequence

Initially, the one octave used was in the second and third octaves available, but this produced a melody that sounded like a bass guitar, so the octave for the notes was changed to span the fourth and fifth octaves. The generated note sequence was actually quite reasonable (see Table 3 above), although there were some problems that became apparent during the course of the run. Many of the sequences lacked structure, and there were many one or two note sequences which always earned the lowest rating.[7] Surprisingly, this sequence was one of the nicest generated during the original work on the GP-Music System, despite the fact that it was created using the simplest function set of all of the cases.

The Base Line trial demonstrated that the system worked properly, and that better pieces could be evolved over time. Further, some of the sequences were actually pleasant to listen to, suggesting that this technique for generating short melodies was reasonable.

Complex Functions Trial

The next step beyond the Base Line trial was to add more complicated functions. Although the pieces generated in the first trial were not bad, most of them lacked structure, since the ability of GP to use functions to add structure was not being used. This trial added in the rest of the functions discussed in Section 4.1.3. The sequences generated during this trial were somewhat better overall than those of the Base Line trial, although the best ones were not of quite as high a caliber. Overall, adding the new functions seemed to smooth out the variation between the best and worst individuals. Having the structure kept very bad individuals from appearing at the expense of less variety in the good individuals.

Increased Depth Trial

Both the Base Line trial and the previous one with the complex functions suffered from a large number of extremely short note sequences (one to two notes). Since all of these get rated very low, they serve little purpose in the evolution and are usually washed out of the generation in which they appear. To try and combat this, the allowed depth of the program trees was increased to 12, and the initial population trees were increased in depth to the range 6-9 (previously they had been 1-4 levels deep).

The individuals generated during this trial did not sound any better than those in the previous trials, and if anything were not as nice. The short individuals remained a problem, while long monotonous sequences became more common. This was especially onerous since all individuals in each generation had to be listened to, so the time for a user to complete a run was increased.

---

[7] This was a continuing problem during all of the trials.

The reason that short individuals still occurred in some frequency is that the depth of the program tree is not directly correlated to the number of notes played. For example, the tree:

```
(shift-up (shift-down (shift-up C-4)))
```

has a depth of four, but still only generates one note. On account of this problem, the overall quality of the individuals in these runs was decreased since in addition to short sequences automatically generating low ratings, many of the long individuals were now also receiving low ratings. With the population of each generation already small, this meant that it took longer for evolution to create decent pieces. Overall the effect of increasing the depth was not very successful.

Automatically Defined Functions Trial

In the initial concept for the GP-Music project, one of the exciting possibilities of using Genetic Programming seemed to be the use of Automatically Defined Functions, or ADFs. These were first presented in Koza's second book [8], which concentrated on their use. Essentially an ADF is a separate program tree which can be reused within the body of the main program tree. A given ADF can be either self-contained, or take parameters which are used in its own tree. An ADF may also be allowed to call other ADFs. The main use of the ADF is to allow individuals in a GP population to be able to discover and take advantage of structure. This frequently leads to smaller individuals, a shorter number of generations until a satisfactory individual is discovered, and more elegant and comprehensible solutions to problems.

Many melodies have common themes that repeat themselves during the course of the melody, and other musical pieces will have a common technique (such as the same sequence played more and more rapidly) used throughout them. Both of these seemed ideally suited to Automatically Defined Functions. If a function was generated with a pleasant tune, it could be used throughout the sequence. Similarly, an ADF might develop a certain stylistic capability which could be used throughout the sequence.

For the ADF trial, three ADFs were introduced. The first took no arguments, and was designed to generate a musical phrase for reuse in the sequence. The second took two inputs, and was hoped to be able to evolve a stylistic technique, and the third was allowed to call the other two ADFs. Unfortunately the trial was a disaster, and the sequences were extremely long and monotonous. The effect of adding the ADFs was not to improve the structure, but merely increase the length of sequences. It is possible that adding all three ADFs was partially to blame for this. ADFs still seem to hold promise, but probably need to be more restricted in their size, function and terminal sets based upon their desired function. Co-evolution of the ADFs along with the sequences would have probably been helpful as well, since the ADFs were frequently not used in the main sequences, in which case a very bad ADF could be carried to the next generation.

C-Major Key Trial, with Null String Elimination

After the disappointing results in the previous two trials, both ADFs and the increased-depth were removed (depth was dropped to a maximum of 9 and a start range of from 4-6). Instead two new techniques were introduced to try and increase the overall quality of the sequences generated.

The first was to limit the notes to the C-Major key, which includes only the whole notes and no sharps. Pieces composed in the C-Major key are known to sound pleasant and up beat from music theory. Although initially I had not intended to introduce domain knowledge, this seemed a fairly straightforward addition. Besides changing the notes terminals, this entailed modifying the 'shift-up' and 'shift-down' operators to only shift notes to whole-note values.

The second addition was to eliminate sequences with less than three notes from the initial population. This was another attempt to combat the extremely short sequences which were limiting the effectiveness of the evolutionary technique.

Overall the sequences generated during this trial were quite nice. As a whole the individuals were nicer than even those created during the initial addition of complex functions, primarily due to the use of the C-Major key which kept individuals from using dissonant note structures. The problem with short individuals was also almost completely eliminated, although some did appear in later generations due to mutation and crossover. An example individual is shown in Table 4.

Note Sequence: A-5, RST, A-5, RST, F-4, RST, F-4, RST, E-4, RST, E-4, RST, F-4, RST, F-4, RST, A-5, RST, A-5, RST	
Web Site File Name:	8tune.au

Table 4- C-Major Individual Example

This sequence was generated by an individual found in the fourth generation. It is a simple sequence using note pairs going down the scale and then back up again. While pleasant sounding, it is not really a complete melody in and of itself.

Pseudo-Chord Trial

The final trial involved the addition to the terminal set of the pseudo-chords (detailed in Section 4.1.3). In music theory, chords in the C-Major scale are formed by playing a note simultaneously with a note four half tones higher, and another seven half tones higher. The pseudo-chords introduced use the same notes in the chords except played sequentially. The motivation for this addition was to add some short sequences that were known to sound good into the program trees in the hope that better overall sounding individuals would be created.

The effect of this trial was startling. Almost all of the generated individuals were pleasant to listen to, a marked change from all of the previous trials. The only possible complaint is that many of the individuals sounded similar as they all tended to rely on the pseudo-chords. The sequences also did not develop very interesting motifs, as some in the Base Line trial with shifted notes or the complex function trial did. One best-of-run individual is shown in Table 5.

Note Sequence: G-4, RST, B-5, RST, D-5, RST, G-4, RST, B-5, RST, D-5, RST, RST, D-5, RST, B-5, RST, G-4, RST, D-5, RST, B-5, RST, G-4, A-5, RST, C-5, RST, E-5, RST, RST, E-5, RST, C-5, RST, A-5, G-4, RST, B-5, RST, D-5, RST, G-4, RST, B-5, RST, D-5, RST, RST, D-5, RST, B-5, RST, G-4, RST, D-5, RST, B-5, RST, G-4	
Web Site File Name:	9tune.au

Table 5- Pseudo Chord Individual Example

This sequence arose in the fifth generation and combined some of the sections from individuals in earlier generations.[8] It 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. It is interesting to note that this sequence only uses one pseudo-chord.

Analysis was also made of the Pseudo-Chord trial to see if the GP-Music System was actually functioning to generate better individuals over time. The trend across generations was that

[8] See 9tune-1.au, 9tune-2.au and 9tune-3.au on the web page from the 2nd through 4th generations.

the fitness of the best and average individuals always increased, although the worst individuals score decreased between some generations.[9] This proved that Genetic Programming was a viable technique to use in creating musical sequences. Note that this did not necessarily have to hold true, as it could have been that combining two pieces of music together would have always yielded two sequences both inferior to the parent.

4.3 Main Issues

Several issues arose during the previous work on the GP-Music System. The main ones were subjectivity, the tradeoff in constructing the function and terminal sets between a free structure, and a more rigid one, and the issue of convergence.

Subjectivity

The first point that the trials really emphasized was how musical sequences cannot be objectively classified. The one problem that was known at the outset of the project was that ranking of musical pieces would have to be relative to the group in which they exist. This was apparent in the trials, where the raw fitness values assigned were always relative to the current trial. A fitness value of 17 could mean something decent in one trial, or pretty bad in another. Not only was this a problem between trials, but it was also possible that individuals in each generation were rated relative only to their peers. Thus, an individual in a later generation with the same rating as one in an earlier generation might actually sound better. Of course, it also goes without saying that a musical sequence appealing to one person, also might not sound very nice to another.

Freedom vs. Direction Tradeoff

The second main issue is the tradeoff between giving complete freedom over the domain of musical sequences to the GP process, and imposing structure on it which may increase the quality, but restrict the movement over the fitness landscape. One of the nice features of Genetic Algorithms and Genetic Programming is that they can frequently find non-traditional solutions to problems since they do not require a lot of domain knowledge. When applying this maxim to musical sequences, one would expect that some unusual but nice sounding sequences might be found. In order to get this effect, however, you are best off with the Base

[9] Considering the problems encountered with extremely small sequences arising due to mutations, this was not to

Line trial which does not impose any structure on the pieces. Unfortunately this flexibility yields more sequences that are not very nice, and it takes longer to evolve nice pieces. This is especially burdensome when performing a user-interactive process. By adding more complex functions and pseudo-chords it was possible to greatly increase how pleasant each sequence was, which made the evolutionary process proceed more smoothly, but the pieces became more restricted to a common type of sound.

Convergence

The final issue is convergence, which is somewhat related to the domain restriction issue just mentioned. In many of the runs of the system there were only one or two individuals that sounded good, or even had nice sections in the initial population. Since these were rated so highly relative to the other sequences, they tended to dominate the other individuals and become more and more prevalent as the run proceeded. Frequently by the last generation half of the individuals were the same, and all of them were of mediocre quality. The converse also occurred, where all of the pieces were very bad in the beginning, so they all propagated to the next generation at approximately the same rate. Although convergence to better sequences would occur over the six generations of the run, it would be too slow to produce anything very nice.

4.4 Summary

The basic implementation of the GP-Music System showed that it is possible to evolve very reasonable sounding short melodies using interactive Genetic Programming. The use of structuring functions such as ‘play-twice’ and ‘play-and-mirror’ in many of the evolved individuals demonstrated that there is an advantage to the flexibility of program trees allowed by Genetic Programming as opposed to Genetic Algorithms. The “user bottleneck” problem of interactive GP was again an issue, as only a small number of musical sequences could be rated at a given time without losing concentration. Even with the limited populations and numbers of generations this required, however, decent sounding pieces were created.

---

surprising.

5. Applications Overview

This section gives an overview of the code developed for the GP-Music System and the associated auto-raters.[10] As mentioned earlier, the system is built on top of the 'lil-gp' system from Michigan State University [17]. The interfaces are based upon the 'xforms' library by Zhao and Overmars [16]. Table 6 lists all the files in the GP-Music System, while Table 7 lists the files for the auto-rating system:

[10] Some of the code was developed during the initial work on GP-Music.

File Size in Bytes	File Name	Description
16,937	app.c	Determines which functions and terminals are used, and defines the fitness function
1,114	app.h	Default header file
1,820	appdef.h	Configures compile time parameters
17,349	auto-raters.c	Interface to use trained auto-rater in GP-Music System
1,159	auto-raters.h	
2,409	cache.c	Caches ratings to insure inter-generation consistency
1,121	cache.h	
22,806	crossovr.c	Implements crossover (modified from lil-gp code)
24,569	exch.c	Modified Kernel Code for fitness evaluation
79,809	forms.h	Header file for xforms
1,024	forms/	Interface testing code
7,400	function.c	Implementation of functions and terminal sets
2,920	function.h	Header for functions and terminal sets
1,931	GNUmakefile	The Makefile
46,014	gp.c	Implements auto-elimination and man GP loop
2,226	gp-form.c	Implements xforms user interfaces
4,426	gp-form.fd	
552	gp-form.h	
562	gp-form_cb.c	
336	header.bin	XM file header used in creating music sequences for playback
729	input.file	Configures runtime parameters
28,406	instruments.bin	XM instruments
<DIR>	kernel	Directory with unmodified lil-gp kernel code
1,547	lilgp.h	Main lil-gp header file
28,869	main.c	Modified lil-gp main program file (added xforms initialization)
16,044	melody1.wav	Sound files used in tournament rater interface
16,044	melody2.wav	
180	play_pattern	Script to play back create and play back music sequences
13,018	populate.c	Creates initial population
1,658	print-ratings.c	Prints a rating file
1,513	protoapp.h	Lil-GP header file
2,048	rater/	Directory for Auto-Rater Code (see Table 7)
1,310	README	Describes how to compile and run the system
5,338	split-comparisons.c	Code to divide up comparisons and ratings for training the Auto-Raters
2,730	split-ratings.c	
19,480	xm_lib.c	Collection of functions to operate on XM file data
4,760	xm_lib.h	Header for xm_lib.c
4,980	xm-parse.c	Utility to parse XM files
5,504	xm-pattern-generate.c	Utility to allow a user to type in a note sequence and create a XM file
3,009	xm-rand-pattern.c	Utility to generate random note sequence XM files

Table 6- GP-Music System Files and Descriptions

File Size in Bytes	File Name	Description
785	GNUmakefile	The Makefile
29,730	l-auto-rater.c	The code to train the list auto-rater
15,713	l-stats.c	Code to parse the statistics output from the list auto-rater
11,433	neural.c	Basic neural network functionality code
3,485	neural.h	
29,519	t-auto-rater.c	The code to train the tournament auto-rater
16,146	t-stats.c	Code to parse the statistics output from the tournament auto-rater

Table 7- Auto-Rater Files and Descriptions

The files for the entire system are also available on-line. They can be found at:

http://www.cs.bham.ac.uk/~bej/gp-music/gp-music.html

The files are linked onto this page as a gzipped tar file. The page also has recordings of the sequences generated during the project and various other related information.

6. Improvements to the GP-Music System

Before any work was done on the automated fitness raters for the system, several enhancements were made to the system itself. All of these were designed either to make it easier and less tedious for the user during program runs, or to improve consistency of human ratings. The former changes were implemented so that longer program runs could be made, allowing a greater number of ratings to be collected for use in training the automatic raters. The latter changes were to insure that ratings were consistent (i.e. a sequence rated better than another won't be rated worse than a third which is worse than both of them). The changes can be divided into internal and interface changes, and are discussed in the remainder of this section.

6.1 Internal Changes

The three changes made to the internals of the system were to enforce consistency between generations, automatically eliminate unsuitable individuals from each generation, and to increase the octave range of sequences from one octave to two.

Enforced Inter-Generation Consistency

One of the problems in the original system was that ratings were relative, not only between runs, but 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. In a general sense, there is nothing that can be done about this since the user is the only one who knows whether the sequences are better or worse. One of the GP operators, however, is reproduction, where an individual is copied directly from the previous generation into the new one. In the old system this individual was quite likely to be rated differently from one generation to the next. To fix this problem, the system code was changed so that an individual's rating is locked in from generation to generation. When using the list interface (see Section 6.2.1) the user is also given the opportunity to listen to the previously rated piece and see its rating before rating the new generation to help them mentally recalibrate for the new set of sequences.

Automatic Elimination of Unsuitable Individuals

The second internal change was to change the code to automatically eliminate unsuitable individuals before they are placed into the new generation. One of the big problems in the old system was getting songs 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 length minimum and maximum for a run, as well as a minimum and maximum number of notes.[11] During the breeding phase of the genetic programming process, individuals that don't meet the criteria are automatically destroyed, and a new individual is bred. If this individual is also unsuitable, a new one is created, and this process is repeated until a satisfactory individual is created.

Two Octave Note Range

The final internal change was to increase the note range to two octaves. This was a relatively minor change, but turned out to improve the quality of generated sequence significantly. The change was made to include higher pitched notes, so that the new range was from a 4th octave

[11] Since a time slot in the sequence can contain a rest or a note, the sequence length and number of notes are not the same. By setting both sequence length and number of notes values, the amount of silence in a sequence can also be controlled.

C to a 6th octave B.[12] The number of pseudo chords used was also increased to include chords starting with a 3rd octave F up to a 5th octave E.

6.2 Interface Changes

Changes were also made to the users interface to the program. The previous interface (see Section 4.1.6) allowed the user to listen to the current sequence, rate it, and move on to another. This forced the user to evaluate early sequences before the whole generation had been heard, and also gave little indication of the break between generations. A list interface was created to address these issues. In the old interface it was also difficult for a user to decide what to rate individuals, so a tournament interface was also developed which allowed the user to rate individuals by only making comparisons between sequences. The two interfaces are discussed in the remainder of the section.

6.2.1 The List Interface

The principle of the list interface was to give the user all of the sequences in a generation in one big block which could be rated in any order that they chose. This allows the user to change their mind about a sequence's rating after they have heard what the 'competing' sequences sound like. A screenshot of the list interface is shown in Figure 4.

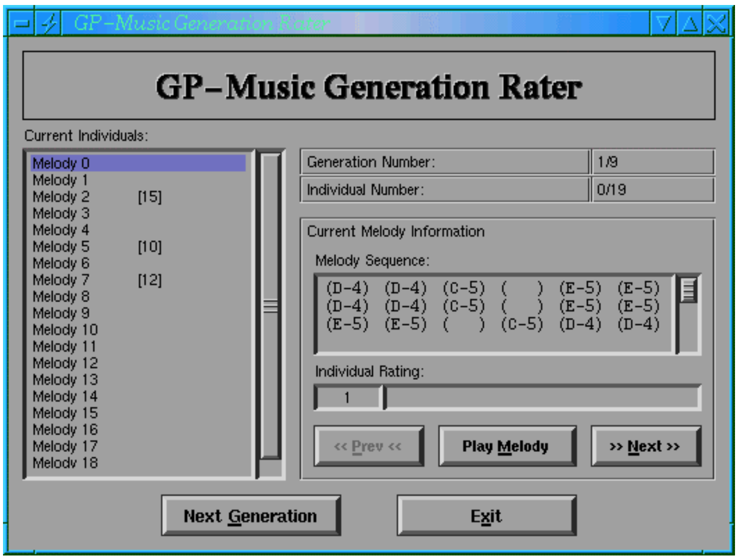


Figure 4- The List Rater Interface

[12] Previously the range was from a 4th octave C to a 5th octave B.

At the beginning of each generation the interface above appears on the screen. The list on the left indicates each melody, and when a melody is clicked with the mouse its details appear in the bottom right corner. The sequence of notes is shown along with its current rating (if any), and an option to play the song. The user can then adjust the rating bar, either with the mouse, or using the arrow keys. The 'Next' and 'Prev' buttons allow the user to jump to the next or previous melodies, and automatically play the new sequence. The up and down arrows can also be used jump forward or backward one sequence. Thus, using just the arrow keys, the user can rate all of the individuals in a generation. Also notice in Figure 4 that some of the individuals have a number in square brackets next to them. This is an indication that that individual was rated in a previous generation and is locked in to that value. As the user rates new individuals, their ratings appear in round brackets (not shown in figure).

6.2.2 The Tournament Interface

The tournament interface attempts to alleviate the problem of how to rate sequences which may all be similar and not very good. This is especially bad in early generations when quality may be particularly low. The tournament interface gets around this by only requiring the user to indicate which of two sequences is superior. This interface is shown below in Figure 5.

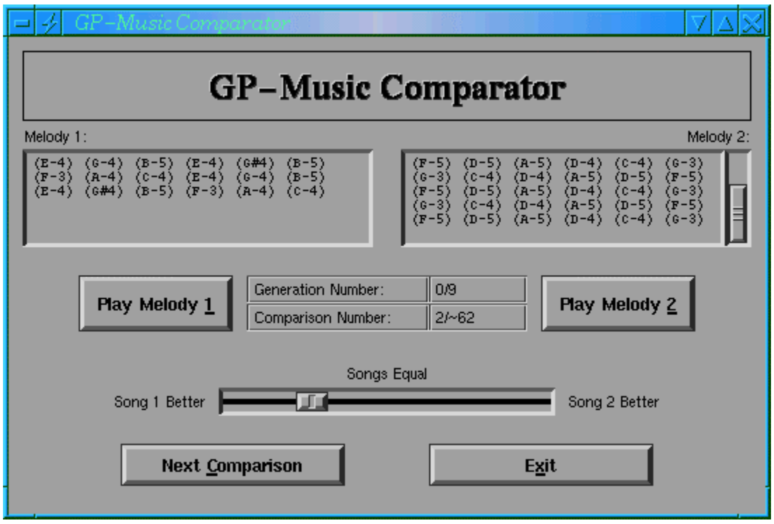


Figure 5- The Tournament Rater Interface

As the figure shows, one melody is shown on each side, with buttons that allow the user to play either of them. A slider at the bottom can be adjusted to indicate which song is better. If the slider is in the middle the songs are about equal. The slider has two notches to either side—the one closest to the middle indicates a small difference in quality, while the one all the way to the side indicates that that melody is much better. When the comparison has been made, the next comparison button is pressed to go to a new comparison. When a new comparison is made, a voice announces sequence one and it is played, and then sequence two is announced and it is played. This saves the user the need to press the 'Play' buttons for the melodies unless they need to listen again for a difficult comparison, or because they were not paying attention.

Internally, the GP-Music System uses comparison results to perform a merge sort on the sequences in the current generation. After the sort is complete the computer assigns a rating to the sequences, starting with 10 for the worst individual. The next individual is given a rating based on the comparison made between it and the worst individual. If the second worst individual was equal to the worst, it is given the same ranking. If it was slightly better it is given a ranking one higher, and if it was much better it is given a ranking two higher. A rating is then assigned in a similar fashion to all of the individuals in the generation. The generated rankings are then displayed to the user using the list interface with the rating slider deactivated. The user can then see what the final order is and listen to any of the sequences before going on to the next generation.

Although this system works well, a merge sort requires $O(n \times \log_2 n)$ ratings, so for a given generation size, it will take more comparisons than ratings to completely rank/rate the generation. For example, with 20 individuals it takes 20 ratings, or 86 comparisons.[13] Also for a comparison, two songs need to be listened to rather than one for a rating, so the number doubles. On account of this, rating a generation of individuals using tournament ranking can take up to eight times longer than using the list rater interface. This makes it debatable whether this interface is useful to human rating sequences. One of the main reasons for setting up this interface, however, was to provide data to train a neural network to select between two sequences, which was initially thought to be easier than training a network to assign a rating on a 1-100 scale.

[13] In actual practice it is around 60 comparisons due to constant factors.

7. Automated Rater Description

This section presents the architecture used to create the automatic fitness raters for the GP-Music System. Two automatic fitness raters, or auto-raters, were created—one to rate sequences on a 1-100 scale (called the list auto-rater[14]), and another to select which of two sequences is better (called the tournament auto-rater). Both share a similar architecture, so an overview is presented first. The rest of the section then describes the two auto-raters and their associated parameters.

7.1 Overview

In determining what to use for the automatic fitness raters for this project, several different ideas were considered, but in the end it was decided to use a neural network for its flexibility. The problem with neural networks, however, is that they are better suited to fixed size input patterns, but in the case of the GP-Music System, the note sequences vary widely in length. The initial idea to overcome this problem was to evolve the network weights using a genetic algorithm, and use the network to integrate over the pattern by summing its outputs for different sub-sequences in the melody. This was not very satisfactory, however, and fortunately a method of using standard back propagation (see Section 3.2) was which worked using the concept of shared weights.

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 details of how shared weights are used in this case will be explained along with the discussion of the network topology.

As mentioned earlier, the initial idea was to have the network serve as an integrator, with output being summed over the sub-sequences in the sequence. The network topology with shared weights that was used in the end follows the same principle, only instead of using the same network repeatedly during the evaluation, a single network with shared weights is used. The basic unit of topology for the network is shown below in Figure 6.

---

[14] It is called the 'list' auto-rater since it is designed to stand in for a human using the list rater interface.

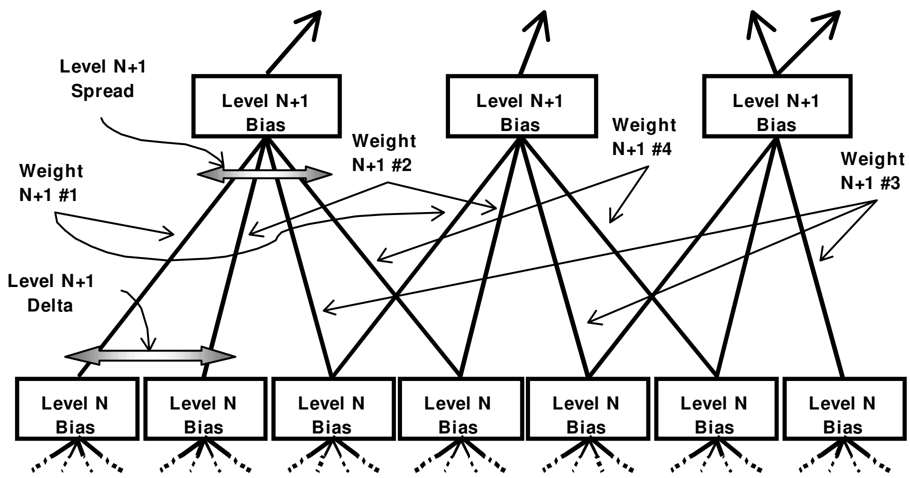


Figure 6- Basic Auto-Rater Network Layout

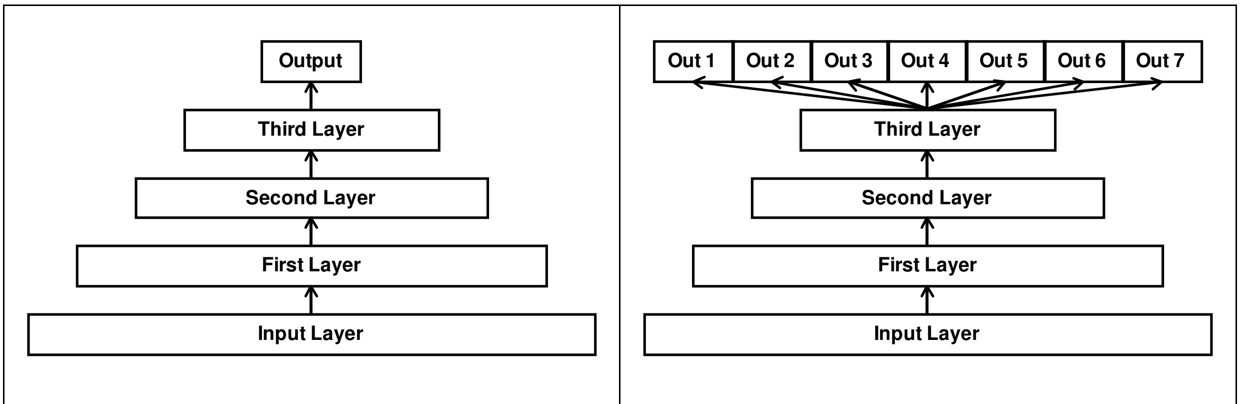
The diagram shows two layers of one of the auto-rater networks, and the connections between them. The bottom most level, labeled 'Level N', is closest to the inputs (or possibly is the input layer), and the upper level, labeled 'Level N+1', is closest to the output node, or nodes (or possibly is the output layer). Each node in the upper level receives input from the lower level nodes. The 'Level Spread', in this case four, 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. This overlap is determined by the 'Level Delta.'

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. In the diagram, this means that each lower level node affects two upper level nodes. Setting the 'level delta' to lower values increases the overlap, and 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 in Sections 7.2 and 7.3). The biases are also shared between all nodes on a given level, so in effect each node and its inputs are duplicate networks.

7.2 The List Auto Rater

The list auto-rater is trained to give the same ratings on a 1-100 score as the human user gives using the list rater interface (see Section 6.2.1). It uses the basic connections between layers as discussed in the previous section, and shown in Figure 6. The overall topology is shown below in Figure 7.



(a) One Output Node (b) Seven Output Nodes

Figure 7- Global Topology for List Raters

As the figure shows there are two topologies for the list auto-rater, one with a single output, and one with seven outputs. Each consists of five layers, including the input layer.

Since the note sequences being rated are variable in length, a new network is built up for each individual that 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 an A-0 is –2, an A#0 is –1, a B-0 is 0, and a C-0 is 1, etc.[15]

[15] The negative values are to take into account that a C-0 is actually the lowest note in the system.

- If the time slot contains a rest (RST), a –1 is loaded into the node.

The –1 is to help the network distinguish between notes and rests, which serve different purposes in the sequence, the note defining the melody, and the 'rest' defining the tempo.

Enough first layer nodes are then constructed to exactly match the number needed given the first level spread and delta amounts. For example, a sequence of length six, with a level one spread of four and a level one delta of two would need two nodes at the first layer. If the number of input nodes does not correspond to a whole numbered amount of first layer nodes, the amount used is rounded up (as is the case in Figure 6, for example). Once the nodes for the first level are created, the connections between the two levels are made as described in the previous section, 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. In the case of the seven output auto-rater, each output node is wired to all of the third layer nodes.

The construction of the network in this fashion allows it to telescope out to whatever the length of the input note sequence. Since the weights are the same going into each of the first layer nodes, each 'level one spread' width sub-sequence is evaluated in a similar manner. At higher levels, the correlations of the sub-sequences from lower levels are being evaluated in similar manners. One drawback of the system is that at the higher levels, some of the weights are not used as much, since the network never needs to 'expand' out far enough that there are many connections used. When the network is trained during back propagation, this results in some weights and layers getting more training than others.

Finally, as is shown in Figure 7, there are two types of output layer for the list auto-rater. The first is a single node which outputs a value between 0 and 1. For training, this is multiplied by 100 to create the appropriate rating on a 1-100 scale. When results with the first rater were initially not as good as expected, a second rater was created with seven output nodes. The output of each of the seven nodes is rounded to 0 or 1, and these seven binary values are treated as a seven bit binary number (0-128), which is then scaled to 1-100 for training and use in the GP-Music System.

In summary, the parameters that are available to be adjusted for the list auto-rater are:

- Level 1 Spread
- Level 1 Delta

- Level 2 Spread
- Level 2 Delta
- Level 3 Spread
- Level 3 Delta
- Output type: One or Seven Nodes

Each of these parameters can be modified to change how accurately the network is able to rate sequences in a training set.

7.3 The Tournament Auto Rater

The tournament auto-rater is designed to stand in for a human using the tournament rater interface (see Section 6.2.2). It looks at two sequences, and returns a value which is interpreted as indicating which of the two sequences is better. The tournament auto-rater works in a similar fashion to the list-auto rater, but has a slightly different topology since it has two sequences as input. The overall topology of the tournament auto-rater is shown below in Figure 8.

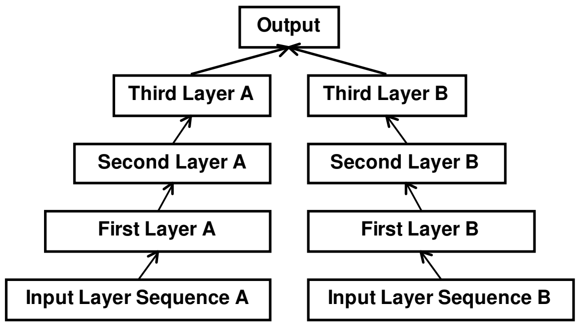


Figure 8- Global Topology for Tournament Rater

Fundamentally, the tournament auto rater works as two separate list auto-raters, except that connections are made from both sets of third layer nodes to the output node. Input layers A and B are made to have one node for each of the time slots in their input note sequence, and subsequent layers are built up from there.

Since note sequences being compared will seldom have the same length, each side of the network will fan out to a different length. It would be possible, therefore, to have different level spreads and deltas for the two sides of the network. Since the operation of choosing a better individual should be symmetric, however, the same level spreads and deltas are used on the two sides. Each side has its own weights and biases, however. To insure that the symmetry of the operation is learned by the network, during training comparisons had to be presented twice, once with the first sequence fed to the 'A' and once in the opposite order (see Section 8.2.2).

As discussed in Section 6.2.2, the tournament rater interface allows the user to select two degrees of how much better or worse one sequence is than another one. The tournament auto-rater was designed to be trained either to give a simple binary output indicating which Sequence, A or B, was better, or to give the more complex answer of just how much better or worse Sequence A was than Sequence B. In the case of the binary output, a value of less than 0.5 was treated as meaning Sequence A was worse than Sequence B, while anything greater than 0.5 was treated as meaning the opposite (output values vary between 0 and 1). For the complex output, five different values were allowed:

- 0: Sequence A is much worse than Sequence B
- 0.25: Sequence A is worse than Sequence B
- 0.5: Sequence A is the same as Sequence B
- 0.75: Sequence A is better than Sequence B
- 1: Sequence A is much better than Sequence B

For this complex output, there is a direct correspondence between the users rating on the slider in the tournament interface, and the network output. In the case of binary output from the network, the user's rating is treated as the logical decision of whether Sequence A is better than or equal to B. So, if the user indicated that Sequence A and B were either equal, or that A was better or much better than B, the network would be trained to output a value of '1'. When using the trained network in the GP-Music System, the output is chosen to mean either that A was better than B, or vice versa, and the much better or much worse superlatives are ignored.

In summary, the parameters that can be adjusted for the tournament auto-rater are as follows:

- Level 1 Spread
- Level 1 Delta
- Level 2 Spread
- Level 2 Delta
- Level 3 Spread
- Level 3 Delta
- Output type: Binary or Complex

By adjusting each of these, the ability of the tournament auto-rater to give the same answer to a comparison as the human user can be adjusted.

8. Research Results

This section discusses the results of research into how best to train the automatic fitness raters. For both the list and tournament auto-raters, the human data gathering runs and their results are presented. Then, the attempts to train the raters with the human rating data are discussed and analyzed. Finally, networks trained with the best parameter settings are used for completely automated runs of the GP-Music System with differing numbers of generations and population sizes, and the resulting sequences are presented. All of the note sequences presented in this section are available to listen to on the web site:

http://www.cs.bham.ac.uk/~bej/gp-music/gp-music.html

They are listed by the web site file name given in the table with each individual note sequence.

8.1 List Auto Rater Analysis

This part of the section discusses the attempts to train the list auto-rater. First the human run that was made to gather the training data is reviewed, then an analysis of training the network for various parameters is given. Finally, the best trained list auto-rater is used to evolve music, and the resulting pieces are listed.

8.1.1 Human Run

The training data was gathered by running the GP-Music System over 9 generations (10 including generation 0), with twenty individuals per generation. This led to the rating of a total of two hundred individuals. During the rating the list interface (see Section 6.2.1) was used, and the modifications to enforce consistency between generations were enabled, but automatic elimination of unsuitable individuals and the two octave range were not used (see Section 6.1). In all other respects the computer was configured as for the Pseudo Chord trial that was made during initial implementation of the system (see Section 4.2). The human rater was Anne Pearce, a retired music teacher. The best individual generated by the run is shown below in Table 8.

<table>
<tr><td colspan="3">List Rater Human Run Best Individual Summary</td></tr>
<tr><td rowspan="5">Individual Statistics:</td><td>generation</td><td>9/9</td></tr>
<tr><td>nodes</td><td>24</td></tr>
<tr><td>depth</td><td>9</td></tr>
<tr><td>sequence length</td><td>120</td></tr>
<tr><td>raw fitness</td><td>46</td></tr>
<tr><td colspan="3">Program Tree:<pre>
(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)))))))
</pre></td></tr>
<tr><td colspan="3">Note Sequence:

E-4, RST, G-4, RST, B-5, RST, B-5, RST, G-4, RST, E-4, RST, E-4, RST, G-4, RST, B-5, RST, B-5, RST, G-4, RST, E-4, RST, A-5, RST, C-5, RST, E-5, RST, A-5, RST, C-5, RST, E-5, RST, D-5, RST, B-5, RST, G-4, RST, G-4, B-5, D-5, G-4, B-5, D-5, C-5, A-5, F-4, F-4, A-5, C-5, D-5, B-5, G-4, D-5, B-5, G-4, E-4, RST, G-4, RST, B-5, RST, B-5, RST, G-4, RST, E-4, RST, E-4, RST, G-4, RST, B-5, RST, B-5, RST, G-4, RST, E-4, RST, A-5, RST, C-5, RST, E-5, RST, A-5, RST, C-5, RST, E-5, RST, D-5, RST, B-5, RST, G-4, RST, G-4, B-5, D-5, G-4, B-5, D-5, C-5, A-5, F-4, F-4, A-5, C-5, D-5, B-5, G-4, D-5, B-5, G-4</td></tr>
<tr><td>Web Site File Name:</td><td colspan="2">anne-list-best.au</td></tr>
</table>

Table 8- Human Run List Best Individual

The individual generated is actually quite nice, perhaps superior to any generated during previous work. 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 during attempts to train the auto-rater.

8.1.2 Training Analysis

The ratings generated during the human run were used to train the list auto-rater network described in Section 7.2. 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 average sum squared error (the square of the difference between the output and the desired output on a 0-1 scale) and average 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. Statistics were also kept for the average decimal error for a given sequence length, and the fraction of individuals falling into any particular error range (the error distribution). These latter distributions were kept for each training cycle.

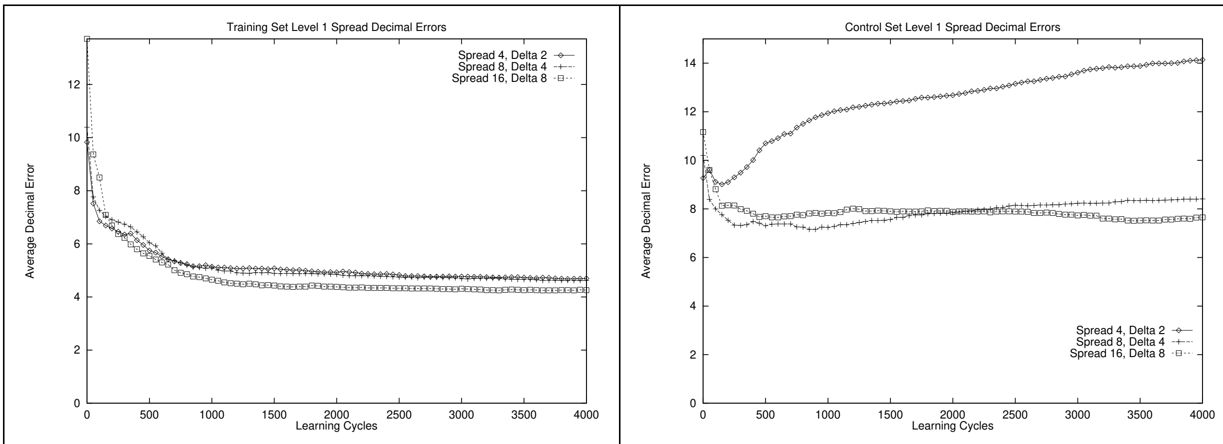
The list auto-rater network was trained over many combinations of the parameters listed at the end of Section 7.2. The default values used for the parameters were:

- Level 1 Spread = 8
- Level 1 Delta = 4
- Level 2 Spread = 8
- Level 2 Delta = 4
- Level 3 Spread = 4
- Level 3 Delta = 2
- Output type = One Node

The default values were chosen in a fairly arbitrary fashion, although some thought did go into the choices. The level for the first two levels was chosen to be eight, since that is about the shortest viable sequence length. The delta value for the first two levels allows each part of the sequence to contribute to two higher nodes, allowing a moderate amount of correlation. The values for the third level were chosen to be half of the first two, since only very long sequences have more than a few second level nodes with which the third level can connect. Overall the default choices give 24 degrees of freedom—8 level one weights, 8 level two weights, 4 level three weights, and biases for levels one, two, three, and the output node. The choice of one node was made based on early runs which showed that the seven output node network did not work very well.

Network training runs were then made in sets, each designed to isolate one of the parameters in order to determine the best value for it. All other parameters were held constant at their default values during the runs for that set. The rest of the section presents the graph output for each parameter set. In each case, two graphs are shown, one showing the average decimal error for the training set over learning cycles, and another showing the average decimal error for the control set.

The first parameter set was designed to determine the best level one spread for the list auto-rater. Three runs were made, one with a spread of 4, one with a spread of 8 (the default), and a third with a spread of 16. In each case the delta was held at ½ the amount of the spread, giving an overlap of two on all input nodes. The graphs of the result are shown in Figure 9.

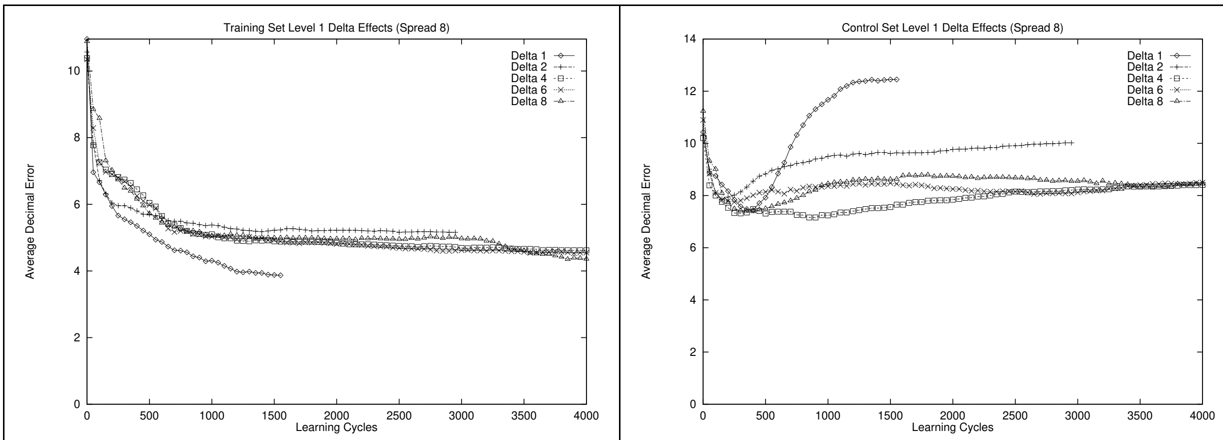


(a) Training Set (b) Control Set

Figure 9- Effects of Level 1 Spread on List Auto Rater Error

Looking at the training set graph, all three permutations steadily decrease as they are trained, with the spread 16 configuration fairing the best. The control set graph is more important, however, as it shows how well the trained network might fair on new individuals. It shows that the spread 8 configuration reaches the lowest average decimal error value, and is therefore the best. Note that this also happens to be the default level one spread.

The next parameter set was configured to identify the best value for the level one delta. It used the default values, including a level one spread of 8, and deltas of 1, 2, 4 (the default), 6 and 8. The graphs are shown below.

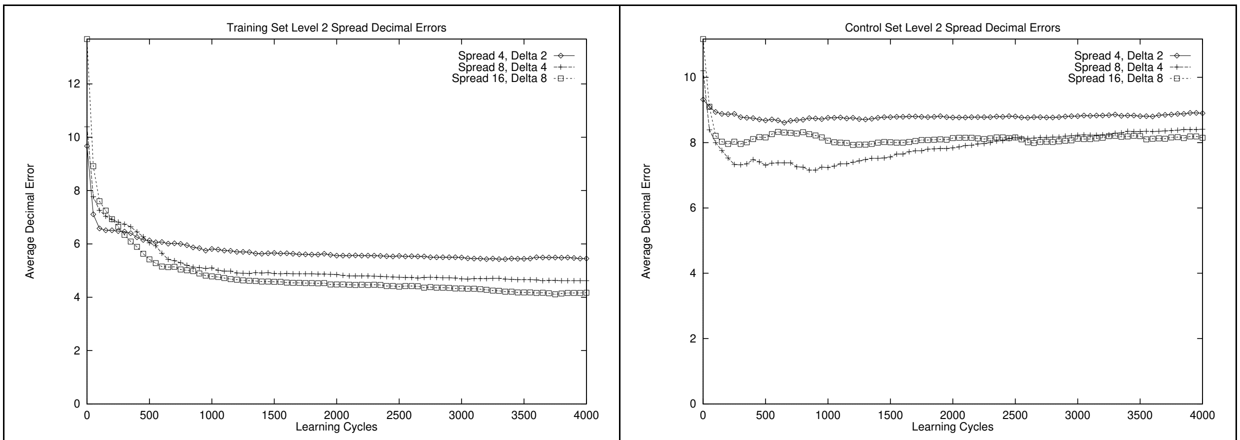


(a) Training Set (b) Control Set

Figure 10- Effects of Level 1 Deltas on List Auto Rater Error

As before the training set error steadily decreases, this time showing that a delta of one does the best. The control set graph reveals, however, that a delta of four is able to achieve the lowest error rate for individuals that have not been used in the training. Again, this happened to be the default value.

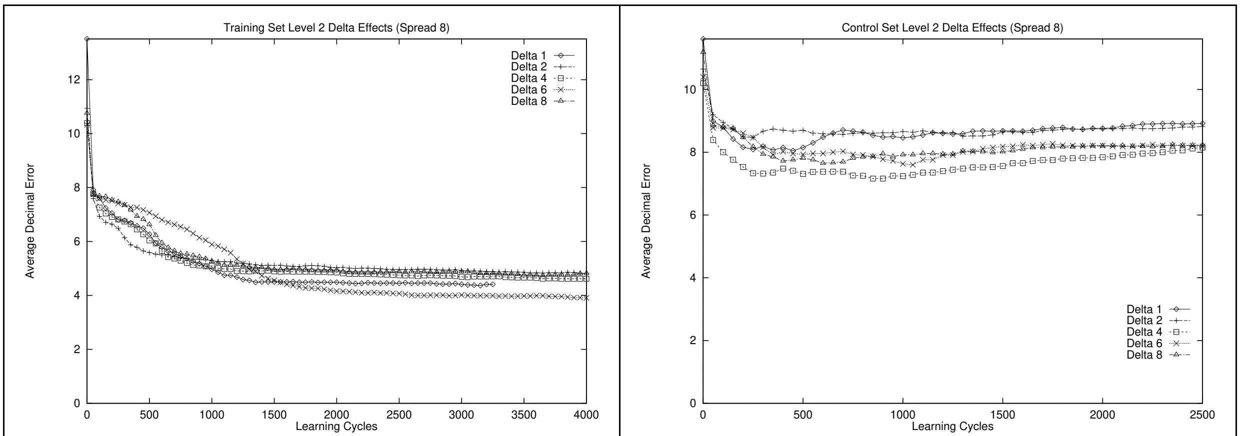
Next, the spread and delta values for the second level were investigated, and the results are shown in Figure 11 and Figure 12, respectively. They parameters were varied from the defaults in a similar fashion to the level one parameter sets.



(a) Training Set (b) Control Set

Figure 11- Effects of Level 2 Spread on List Auto Rater Error

The first figure reveals that like level one, the training set performs best with a spread of 16, while the control set does better with a spread of 8, showing that the default value is the best for the second level spread.

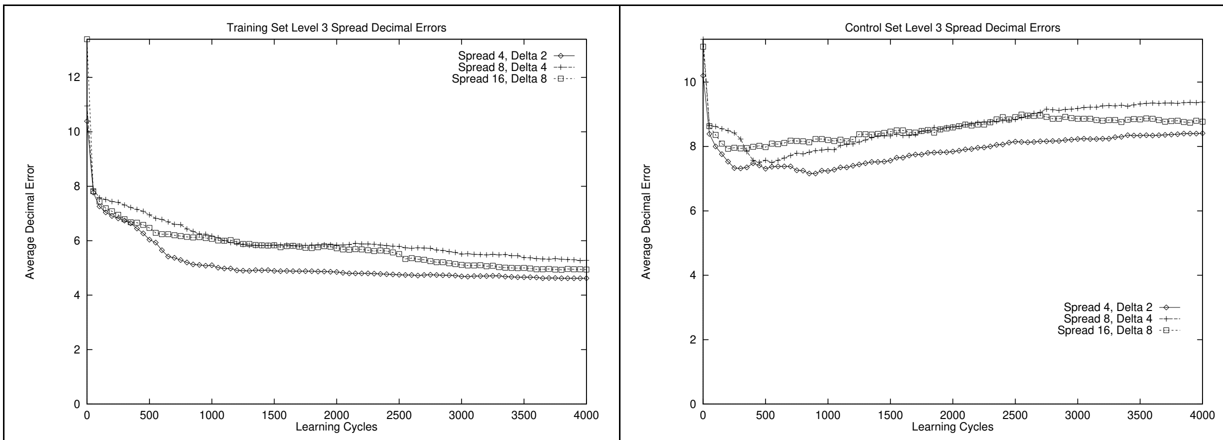


(a) Training Set (b) Control Set

Figure 12- Effects of Level 2 Deltas on List Auto Rater Error

For the second level deltas, a delta of six performs the best on the training set, and a delta of four, the default, does best on the control set.

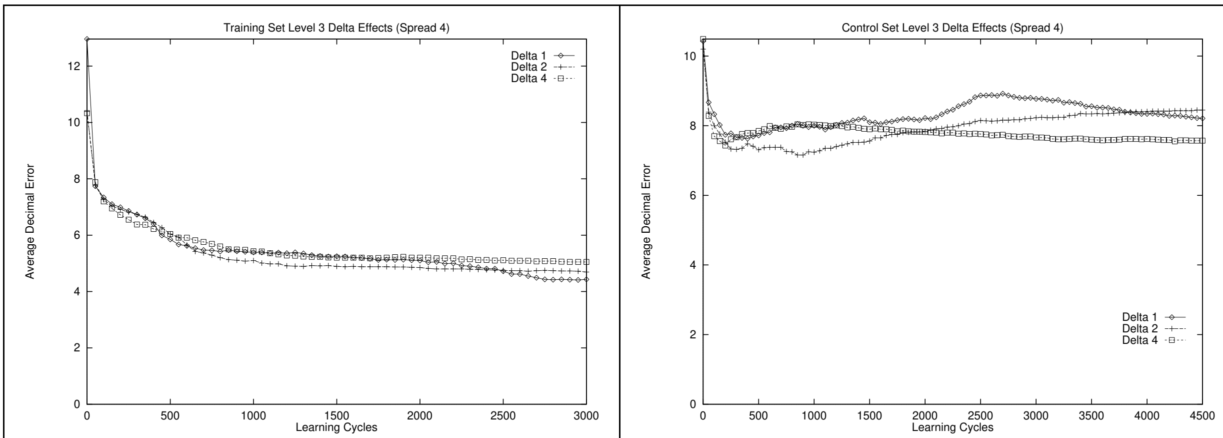
The next two parameter sets looked at values for the spread and delta on the third level of the list auto-rater. The configurations tested were similar in principle to the previous two levels, although the details varied slightly since the default spread and delta for the third level are four and two respectively—half the values for the two lower levels. On account of this, the spread values tested were the same as for the previous levels, but the delta values tested were 1, 2 (default) and 4 on a spread of 4. The results are shown in Figure 13 and Figure 14.



(a) Training Set (b) Control Set

Figure 13- Effects of Level 3 Spread on List Auto Rater Error

For this level, we see that both the training set and control set decimal error rates are minimized by using a spread of four, which was the default value chosen for this level.

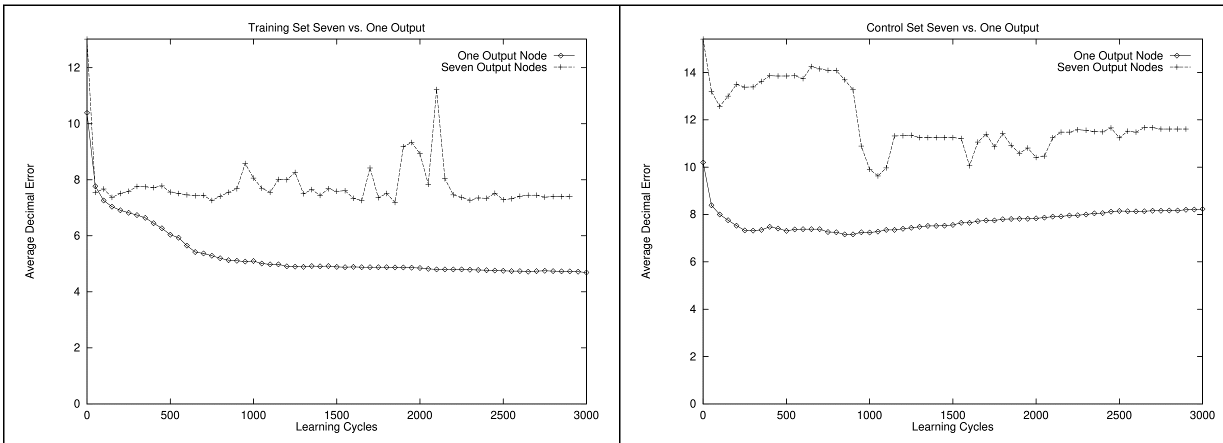


(a) Training Set (b) Control Set

Figure 14- Effects of Level 3 Deltas on List Auto Rater Error

For the third level deltas, the training set seems to do best with a delta of two for most of the learning cycles, but at the very end the delta of one configured parameter begins to do better. In the control set, however, the delta value of two reaches the lowest decimal error, and is therefore the best. As before, this is the default value for the delta.

The final parameter set measures the differences between using the seven and one output versions of the list auto-rater. All other values are held at their defaults. The results are shown below in Figure 15.



(a) Training Set (b) Control Set

Figure 15- Seven vs. One Output List Auto Rater Error

As can be seen in the graphs, the one output node version of the network performs better on both sets. The seven output version of the network was implemented in the hope that more output nodes would give greater accuracy since there would be more weights being used. Unfortunately in practice this did not work well, as the error back propagation was not set up to back propagate more error based on the significance of the bit. Thus, during some training cycles the most significant bit output was modified heavily, greatly affecting the decimal error.

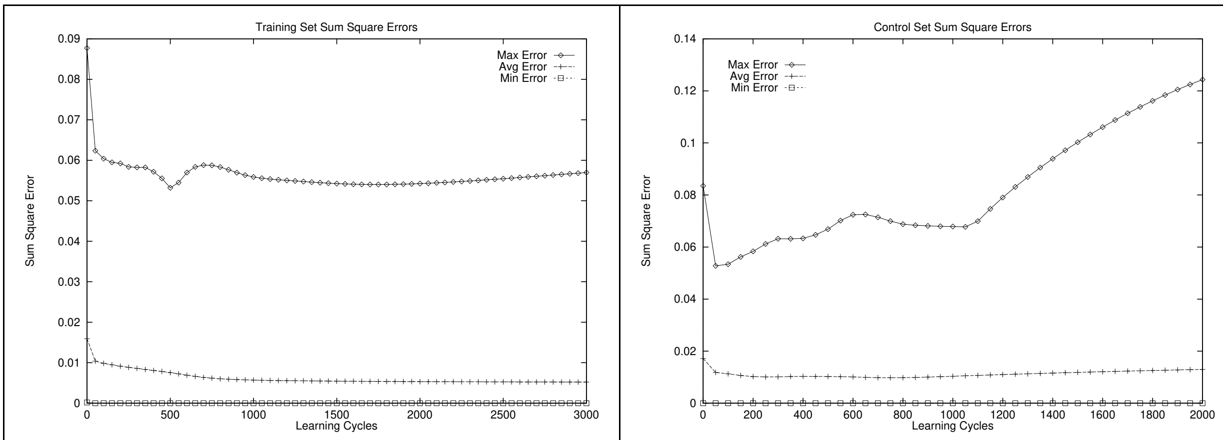
The result of running all the parameter sets was to show that the default values were the best ones to use. This was somewhat surprising, as the defaults were chosen somewhat arbitrarily, but the data presented shows that they are the best. Once again, the default, and best parameters are:

- Level 1 Spread = 8
- Level 1 Delta = 4
- Level 2 Spread = 8
- Level 2 Delta = 4

- Level 3 Spread = 4
- Level 3 Delta = 2
- Output type = One Node

The run statistics for the default individual are now presented in somewhat more detail than was available in the graphs comparing the different parameter configurations.

The first set of graphs, shown in Figure 16, shows the Average, Minimum, and Maximum sum square error for individuals in the training and control sets. This is the square of the difference between the desired and actual network output (in the range 0-1).



(a) Training Set (b) Control Set

Figure 16- Best List Individual Sum Square Error Rates

Notice that for both training and control set the maximum error is much greater than the average error. Also, the minimum error stays constant at almost zero. The average error stays at about twice the training set level for the control set, although due to overfitting the control set error gradually starts to go up after a point. The average sum square errors for both sets are plotted on the same graph below.

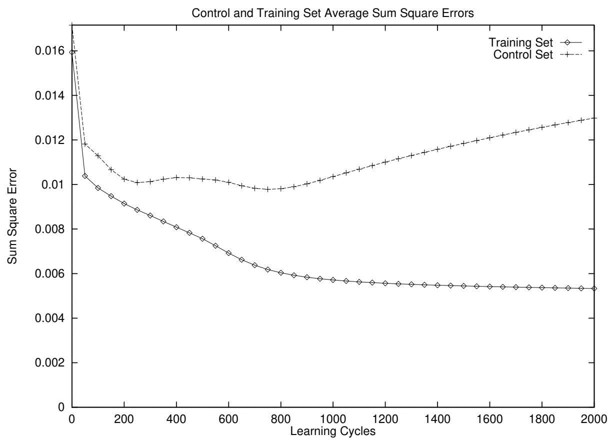
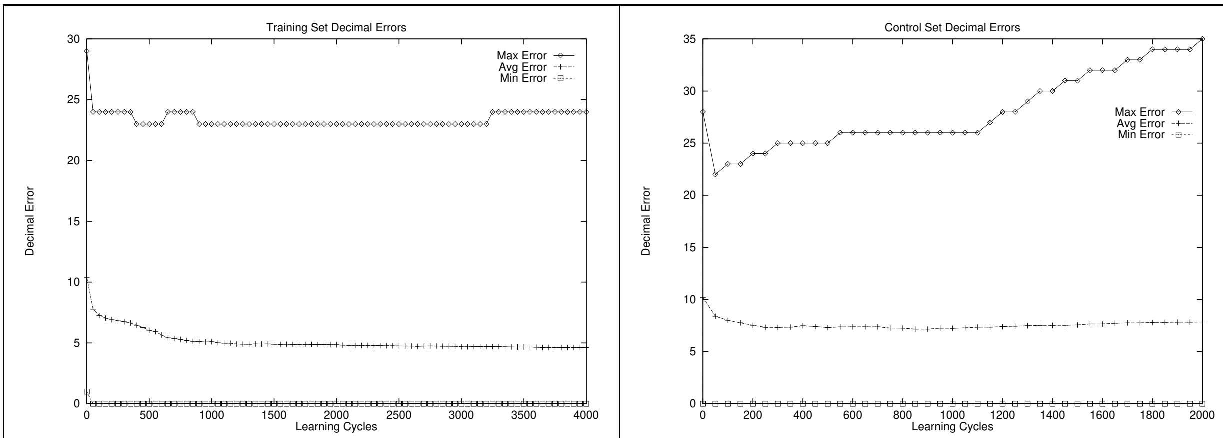


Figure 17- Best List Individual Control and Training Set Sum Square Error

From this figure it is easier to see the training set and control set both get dramatically better initially, after which the training set error value continues to improve, while the control set error flattens out and then starts to get worse around the 800th learning cycle. The reason that the error on the control set is getting worse is that the network is becoming too fitted to the training set data, and is becoming less general.

Although interesting to look at, it is perhaps more relevant and more comprehensible to look at the decimal error rates, which indicate how close the network is able to get the rating on the 1-100 scale that the human user must use. Figure 18 and Figure 19 show the decimal error rates.



(a) Training Set (b) Control Set

Figure 18- Best List Individual Decimal Error Rates

The data here reveals much the same thing as was seen for the average sum square error graphs, although the numerical values are more meaningful. The average error on the training set goes down to +/- 5, which is quite good when the rating is out of 100. Unfortunately, the maximum error is +/- 23 which is not so good. Of course the control set is more interesting, since the purpose of the rater is to evaluate individuals with which it has not been trained. Surprisingly, the error here is not much worse than the training set during the first 1000 learning cycles, suggesting that the songs have a fair amount in common. This indicates that it is reasonable to train a network to rate sequences in the GP-Music System. A closer look at the difference between the average decimal error in the training and control sets is shown in Figure 19.

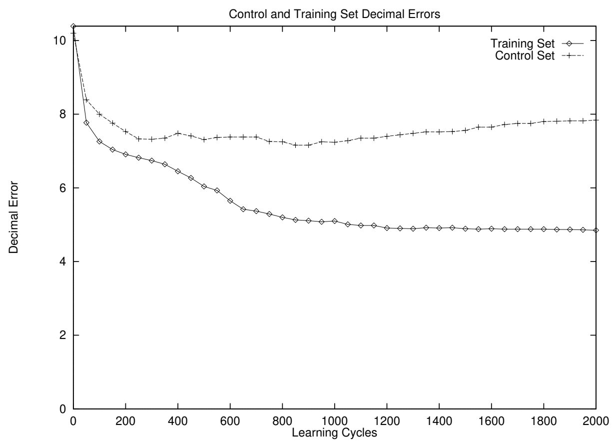


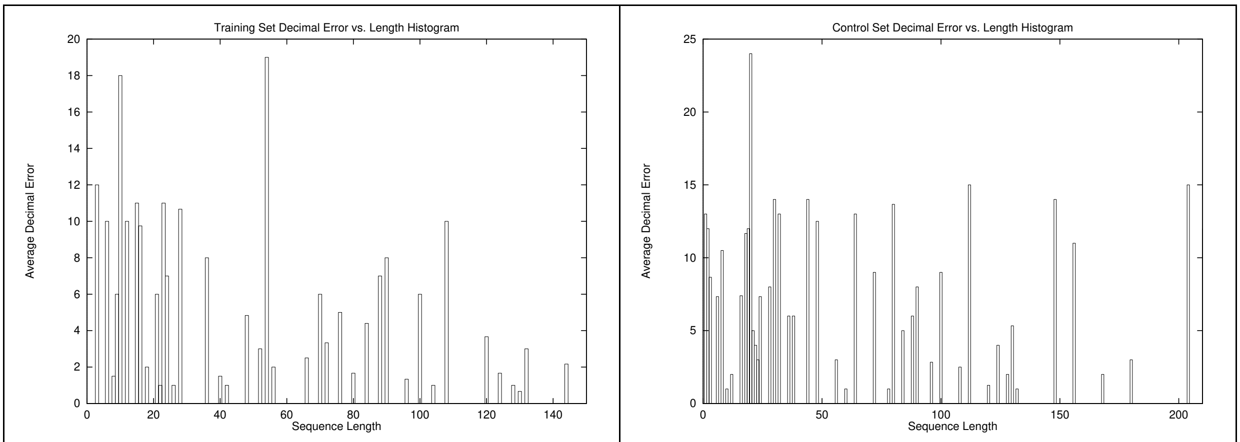
Figure 19- Best List Individual Control and Training Set Decimal Error

The main reason to look at this graph is to determine at which cycle the control set reaches its minimum point. Inspection of the graph, and the data used to generate the graph, reveal that the average error on the control set reaches a minimum at cycle 850, with a value of 7.16. This is not to bad on a 1-100 scale, but is still not great considering the close grouping of ratings of sequences in early generations which are frequently quite poor.

Having identified the best parameters, and now also the point where the network has reached a minimum value on the control set, but overfitting has not begun, it was possible to determine which sets of weights and biases to use for a run of the GP-Music System. Before the run is discussed, however, some statistics for cycle 850 are presented.

In addition to the average error rate, it is also important to look at where the error is coming from. One early hypothesis was that the network was perhaps becoming quite good at rating certain sequence lengths, while it was quite bad at others. This hypothesis was arrived at by considering that the network telescopes out to size determined by the sequence. This might have meant that short sequences weren't using enough of the weights to be accurately rated, or that weights used only for long sequences never got enough training to allow the network to become very accurate.

To assess whether or not size was a factor in the error rate, two length vs. average decimal error histograms are presented in Figure 20, below.

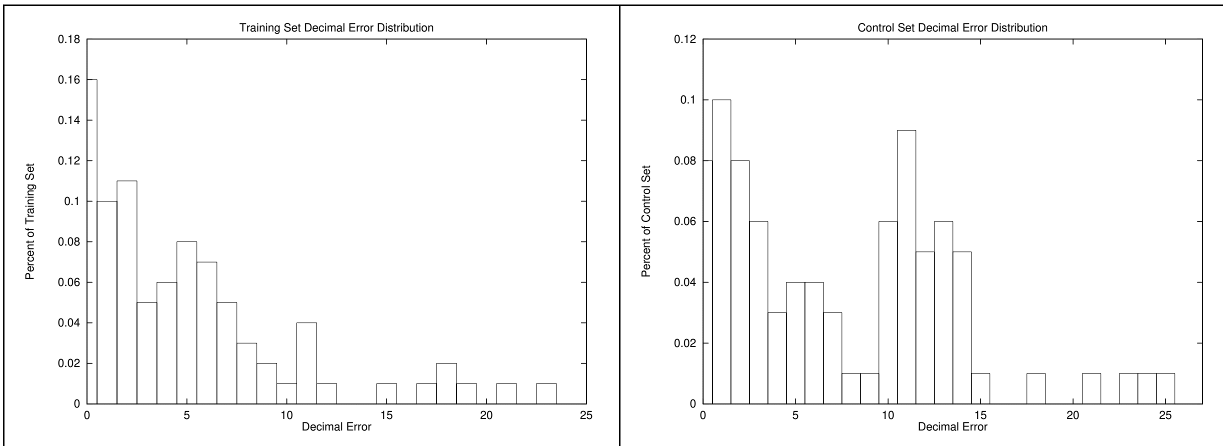


(a) Training Set (b) Control Set

Figure 20- Best List Individual Length vs. Decimal Error Histogram at Cycle 850

As the figure shows, there is no clear correlation between sequence length and the average decimal error for a sequence. High errors are not clustered toward very long or very short sequences. There are some lengths that do have high error rates for both the training and control sets, but they don't seem clustered in any particular way, and are probably due to the specifics of the internal structure of the particular sequences of that length.

It is also important to look at the error distribution in addition to the average error, since this shows whether most errors are very small with a few large errors that are bringing the average error rate up. Figure 21 shows the decimal error distribution for cycle 850. Each point on the X axis shows the fraction of the total number of individuals in the control set which had that amount of error.



(a) Training Set (b) Control Set

Figure 21- Best List Individual Error Distribution at Cycle 850

The error distribution for the training set seems to approximate a normal distribution quite well. The control set distribution seems to be almost bi-modal, with a cluster around errors of zero and errors of twelve. Neither shows an extreme bi-modal distribution with most individuals at zero and one or two with large errors, justifying the use of the average decimal error in choosing the learning cycle to use for the permanent weights and biases in the network.

8.1.3 Trial Run

The weights and biases for the list auto-rater network using the default parameters, with the training set data generated by Anne Pearce, trained for 850 cycles, were used in several runs of the GP-Music System. To facilitate this, the GP-Music System was modified to query the list auto-rater for the ratings of sequences at each generation. The first run was made with the same parameters used during the human run that generated the training set data, and the best individual created is shown here in Table 9:

List Auto Rater Run Best Individual Summary		
Individual Statistics:	generation	7/9
	nodes	35
	depth	12
	sequence length	118
	raw fitness	45.38

Program Tree:

```
(mirror (shift-up (play-two
                     (shift-down
                        (play-twice
                            (play-twice
                                (mirror
                                    (shift-up
                                        (play-two
                                            (play-and-mirror
                                                (play-and-mirror
                                                    (add-space G-4)))
                                            (add-space F-Chord)))))))
                             (shift-up
                                 (play-two
                                     (play-twice
                                         (play-twice
                                             (shift-up
                                                 (play-two
                                                     (play-and-mirror
                                                         (play-and-mirror add-space G-4)))
                                                     (add-space F-Chord)))))
                                     (shift-up
                                         (shift-down
                                             (mirror
                                                 (mirror
                                                     (play-twice
                                                         (mirror
                                                             (shift-up E2-Chord)))))))))))
```

Note Sequence:

A-6, B-6, B-6, A-6, B-6, B-6, RST, F-5, RST, D-5, RST, B-5, C-5, RST, RST, C-5, C-5, RST, RST, C-5, RST, F-5, RST, D-5, RST, B-5, C-5, RST, RST, C-5, C-5, RST, RST, C-5, RST, F-5, RST, D-5, RST, B-5, C-5, RST, RST, C-5, C-5, RST, RST, C-5, RST, F-5, RST, D-5, RST, B-5, C-5, RST, RST, C-5, C-5, RST, RST, C-5, A-5, RST, RST, A-5, A-5, RST, RST, A-5, G-4, RST, B-5, RST, D-5, RST, A-5, RST, RST, A-5, A-5, RST, RST, A-5, G-4, RST, B-5, RST, D-5, RST, A-5, RST, RST, A-5, A-5, RST, RST, A-5, G-4, RST, B-5, RST, D-5, RST, A-5, RST, RST, A-5, A-5, RST, RST, A-5, G-4, RST, B-5, RST, D-5, RST

Web Site File Name: anne-200-best.au

Table 9- List Auto Rater Best of Run Individual (9 Generations, 20 individuals per Generation)

This individual actually sounds quite nice, although not as good as the one generated during the human generated run, it is still quite a bit better than many sequences generated in past work with the GP-Music System. The sequence is, perhaps, a bit repetitive in the middle, but overall sounds quite good.

Of course, a run with 200 individuals is still feasible with a human user. The primary goal of using the automated rater is to complete runs with larger populations over a greater number of generations. To evaluate how well the auto-rater works in larger runs, runs with 100 and 500 sequences per generation over 50 generations was made. The resulting best individuals are shown in Table 10 and Table 11.

List Auto Rater Run Best Individual Summary		
Individual Statistics:	generation	48/50
	nodes	27
	depth	11
	sequence length	184
	raw fitness	57.03

Program Tree:

```
(play-twice (mirror (play-two (add-space (add-space (add-space E2-Chord)))
                              (play-twice (play-two (add-space E2-Chord)
                                                    (play-two
                                                        (add-space
                                                            (play-two
                                                                (add-space (mirror RST))
                                                                (play-and-mirror G1-Chord)))
                                                        (mirror
                                                            (mirror
                                                                (play-twice
                                                                    (play-and-mirror
                                                                        (shift-down
                                                                          E2-Chord)))))))))))
```

Note Sequence:

D-5, G-5, A-6, A-6, G-5, D-5, D-5, G-5, A-6, A-6, G-5, D-5, RST, G-3, RST, B-4, RST, D-4, RST, D-4, RST, B-4, RST, G-3, RST, RST, RST, RST, RST, B-6, RST, G#5, RST, E-5, D-5, G-5, A-6, A-6, G-5, D-5, D-5, G-5, A-6, A-6, G-5, D-5, RST, G-3, RST, B-4, RST, D-4, RST, D-4, RST, B-4, RST, G-3, RST, RST, RST, RST, RST, B-6, RST, G#5, RST, E-5, RST, RST, RST, RST, RST, RST, RST, B-6, RST, RST, RST, RST, RST, RST, RST, G#5, RST, RST, RST, RST, RST, RST, RST, E-5, D-5, G-5, A-6, A-6, G-5, D-5, D-5, G-5, A-6, A-6, G-5, D-5, RST, G-3, RST, B-4, RST, D-4, RST, D-4, RST, B-4, RST, G-3, RST, RST, RST, RST, RST, B-6, RST, G#5, RST, E-5, D-5, G-5, A-6, A-6, G-5, D-5, D-5, G-5, A-6, A-6, G-5, D-5, RST, G-3, RST, B-4, RST, D-4, RST, D-4, RST, B-4, RST, G-3, RST, RST, RST, RST, RST, B-6, RST, G#5, RST, E-5, RST, RST, RST, RST, RST, RST, RST, B-6, RST, RST, RST, RST, RST, RST, RST, G#5, RST, RST, RST, RST, RST, RST, RST, E-5

Web Site File Name: anne-5000-best.au

Table 10- List Auto Rater Best of Run Individual (50 Generations, 100 individuals per Generation)

Unfortunately, this 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. It was found at generation 47, however, which was quite late in the run, so it does show that the fitness was going up. A human would probably not have rated this piece so highly, however.

List Auto Rater Run Best Individual Summary		
Individual Statistics:	generation	50/50
	nodes	25
	depth	12
	sequence length	184
	raw fitness	43.04

Program Tree:

```
(add-space
    (shift-up
        (shift-up
            (mirror
                (shift-up
                    (play-two
                        (play-twice
                            (play-two
                                (play-twice
                                    (play-twice
                                        (play-two E2-Chord RST)))
                                (play-twice E2-Chord)))
                        (mirror
                            (play-and-mirror
                                (add-space (play-two (mirror (add-space C2-Chord))
                                                     (play-twice
                                                         (shift-up E2-Chord)))))))))))
```

Note Sequence:
RST, RST, RST, RST, B-6, RST, RST, RST, RST, RST, RST, RST, A-6, RST, RST, RST, RST, RST, RST, RST, F-5, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, B-6, RST, RST, RST, F-5, RST, RST, RST, RST, RST, RST, RST, A-6, RST, RST, RST, RST, RST, RST, RST, B-6, RST, RST, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST, RST, RST, B-6, RST, B-6, RST, A-6, RST

Web Site File Name:	anne-25000-best.au

Table 11- List Auto Rater Best of Run Individual (50 Generations, 500 individuals per Generation)

This sequence is better than the one generated during the previous run with 100 individuals per generation. 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, it sounds almost like something that would be played on drums. Nonetheless, it is not unpleasant to listen to, and is considered to be a success for the system.

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

8.2 Tournament Auto Rater Analysis

This section reviews the work that was done on determining the optimum configuration for the tournament auto-rater as described in Section 7.3. It is laid out in a similar fashion to the discussion for the list auto-rater in the previous section. Since much of the information is the same, the results are presented with much less discussion.

8.2.1 Human Run

As with the list auto-rater, a human run had to be made, this time using the tournament rater interface (see Section 6.2.2) to the GP-Music System. As with the list interface run, ten generations of twenty individuals were rated. In this run, the automatic elimination of unsuitable individuals was enabled insuring that sequences had a length of between 5 and 200. The number of notes was also set to the same range, or, in other words, there was no restriction placed on the amount of silence. The two octave range was also enabled. Both of these features were discussed in Section 6.1. Other than that all parameters for the GP-Music System were the same as for the list auto-rater run (see Section 8.1.1). This time the run was made by the author, Brad Johanson. As mentioned earlier, the number of comparisons required to rate the individuals is greater than the number of ratings required using the list interface. Over the ten generations of the run, 620 comparisons were made, and sequences were played back 1240 times. The run took over eight hours to complete. Clearly this is not a good interface for humans! The best individual for the run is shown in the following table:

Tournament Rater Human Run Best Individual Summary		
Individual Statistics:	generation	0/9
	nodes	11
	depth	7
	sequence length	56
	raw fitness	48
Program Tree: `(shift-up (mirror (play-and-mirror (play-and-mirror (play-two (play-twice (shift-down A-6)) (play-and-mirror (add-space C2-Chord)))))))`		
Note Sequence: A-6, A-6, D-5, RST, F-5, RST, A-6, RST, RST, A-6, RST, F-5, RST, D-5, D-5, RST, F-5, RST, A-6, RST, RST, A-6, RST, F-5, RST, D-5, A-6, A-6, A-6, A-6, D-5, RST, F-5, RST, A-6, RST, RST, A-6, RST, F-5, RST, D-5, D-5, RST, F-5, RST, A-6, RST, RST, A-6, RST, F-5, RST, D-5, A-6, A-6		
Web Site File Name: bej-tourn-best.au		

Table 12- Human Run Tournament Best Individual

This is probably the best sounding individual ever created by the GP-Music System. It has a distinctive almost Latin American sound to it. The individual was from generation 0, the very first one, so the evolution process did not factor into its creation. This suggests that part of the reason nice melodies are encountered is the particular function set being used.

One of the problems with the tournament method of rating individuals is that the top rating in any generation is always approximately the same. This arises from the way ratings are assigned from the sorted list—the lowest individual always gets a 10, and each better individual is given an incrementally higher ranking based on the comparison between it and the next lower individual (see Section 6.2.2 for more detail). On account of this the maximum rating is pre-determined, and there is no way to tell which generation really had the best individual. If evolutionary forces are working well, however, the best of generation individual in later generations should be better than the earlier ones. So, throughout this section, both the best of run and the best of final generation individuals are presented.

Table 13 shows the best individual of the ninth (tenth including generation zero) generation for the human run. Note that due to the way run information is kept, some of the information about the individual is lost, so only the program length and generated note sequence are displayed.

Tournament Rater Human Run Last Generation Best Individual Summary		
Individual Statistics:	generation	9/9
	sequence length	64
Note Sequence: F-5, A-6, B-6, F-5, A-6, B-6, F-4, F-4, F-4, F-4, RST, B-6, RST, B-6, RST, G-5, F-5, A-6, B-6, F-5, A-6, B-6, F-4, F-4, F-4, F-4, RST, B-6, RST, B-6, RST, G-5, G-5, RST, B-6, RST, B-6, RST, F-4, F-4, F-4, F-4, B-6, A-6, F-5, B-6, A-6, F-5, G-5, RST, B-6, RST, B-6, RST, F-4, F-4, F-4, F-4, B-6, A-6, F-5, B-6, A-6, F-5		
Web Site File Name:	bej-tourn-best-09.au	

Table 13- Human Run Tournament Generation 9 Best Individual

This individual only sounds so-so, as it suffers from large changes in pitch, similar to the best of run individual for the 100 individual x 50 generation run with the list auto-rater. It is definitely not as nice as the best of run individual found in the very first generation. This suggests that there is always a danger in the GP-Music System of the populations descending to a point where they are all strange hybrids of nice sequences in previous generations, none of them as nice as the originals.

8.2.2 Training Analysis

After the human run, the ratings generated during the run were used to train the tournament auto-rater network described in Section 7.3. The ratings were divided up into two sets of 310 comparisons; one served as a training set and one as a control set. Since comparisons are applied to the network with both orderings of the sequences being compared, this actually leads to 620 comparisons per set. Training cycles were made as they were for the list auto-rater. Statistics were kept at the end of each learning cycle measuring the average sum squared error and the fraction of comparisons which were decided appropriately for both the control and training sets. Statistics were also kept for the average decimal error for a given set of sequence lengths in a comparison, as well as the fraction of individuals falling into any particular error range (the error distribution). These two distributions were generated for each learning cycle.

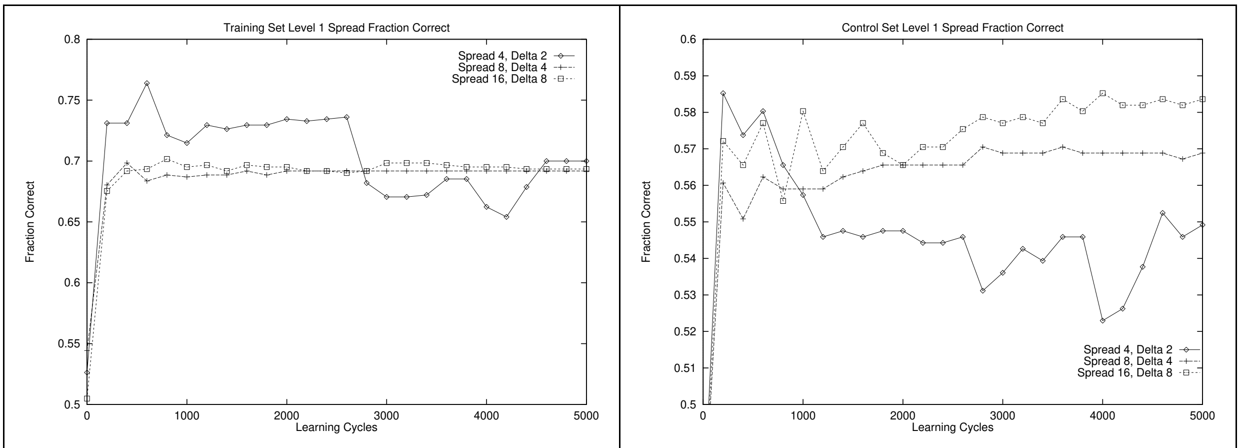
As with the list auto-rater, the tournament auto-rater was trained over variations from a set of default values. The default values used for the parameters were:

- Level 1 Spread = 8
- Level 1 Delta = 4

- Level 2 Spread = 8
- Level 2 Delta = 4
- Level 3 Spread = 4
- Level 3 Delta = 2
- Output type = Binary

The default values were chosen to be the same as those for the list auto-rater, except for the output type, which did not directly correspond between the two auto-raters. The default choices give 47 degrees of freedom—for both the A and B sequence sides there are 8 level one weights, 8 level two weights, 4 level three weights, and biases for levels one, two, and three. There is also one output bias. The choice of binary output was arbitrary.

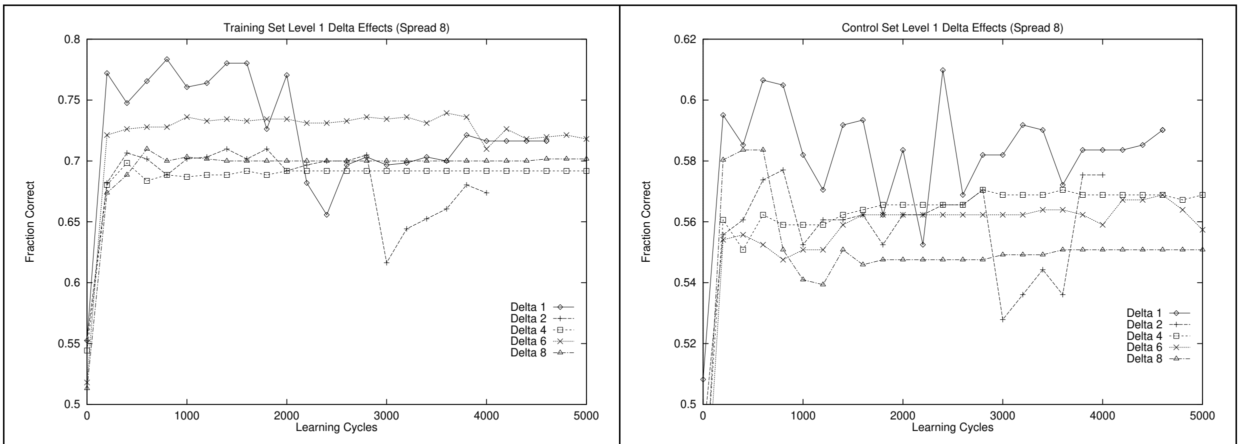
As before, various parameter sets were tried to determine the best values for the different tournament auto-rater parameters. Again, two graphs are shown for each parameter set, but this time the fraction of comparisons which agree with the human's ratings is given instead of the decimal error. Note that unlike for decimal error, the goal is to maximize the fraction correct, so higher values are better. All of the graphs are set display values above 0.5, which is the rate that would be achieved by randomly guessing which sequence was better. Figure 22 shows the runs to determine the level one spread.



(a) Training Set (b) Control Set

Figure 22- Effects of Level 1 Spread on Tournament Auto Rater Fraction Correct

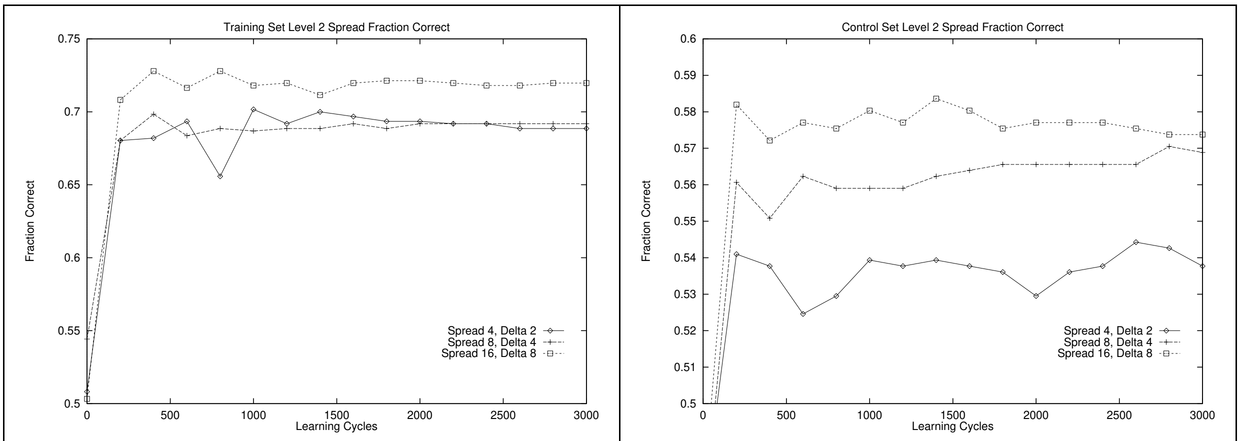
For the training set, a spread of four is the best for most of the learning cycles, although at the end it becomes worse. Note that the fraction correct graph for the training set does not steadily get better like an error graph. This is because the network is being trained to minimize the difference from the values of 0 and 1, not to maximize the number of comparisons which fall on the correct side of 0.5. Thus, in lowering the error for many comparisons which are already being made correctly, it might force some border line comparisons to switch to being incorrect. In the control set the spread of sixteen achieves the best fraction correct at around 59% of the comparisons made correctly. So, the best spread seems to be sixteen, which is not the default value.



(a) Training Set (b) Control Set

Figure 23- Effects of Level 1 Deltas on Tournament Auto Rater Fraction Correct

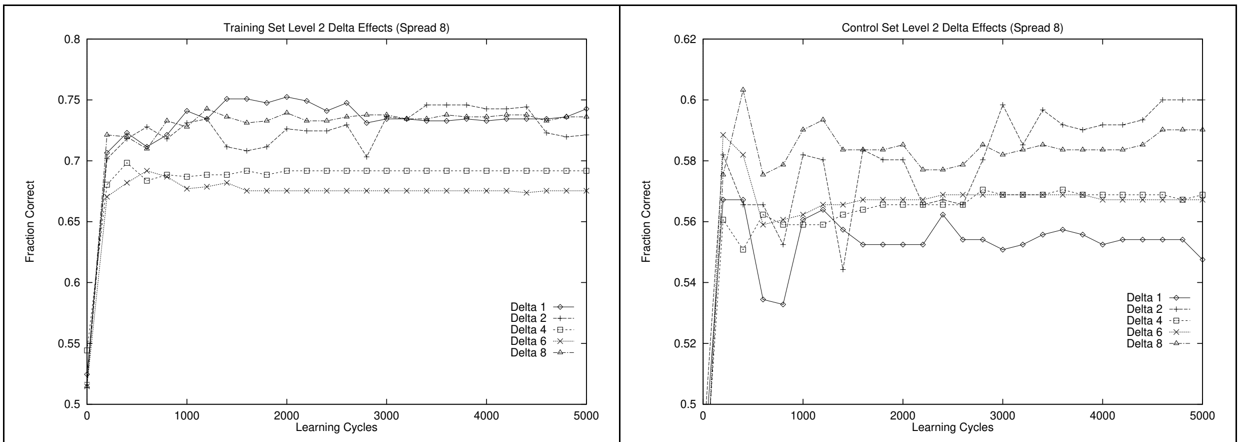
The graphs above show the runs to determine the value of the level one delta. For the training set, the highest fraction correct is achieved by a delta of one, although it falls off rapidly as learning cycles pass. Delta of one also is the best for the control set. Since the spread for the networks being tested was all eight, this indicates that a delta of 1/8 of the spread value is the best value for the level one delta.



(a) Training Set (b) Control Set

Figure 24- Effects of Level 2 Spread on Tournament Auto Rater Fraction Correct

For the level two spread parameter set, the spread of sixteen does best for both the training and control sets. A value for the level two spread of sixteen is therefore chosen to be best, again a departure from the default value.

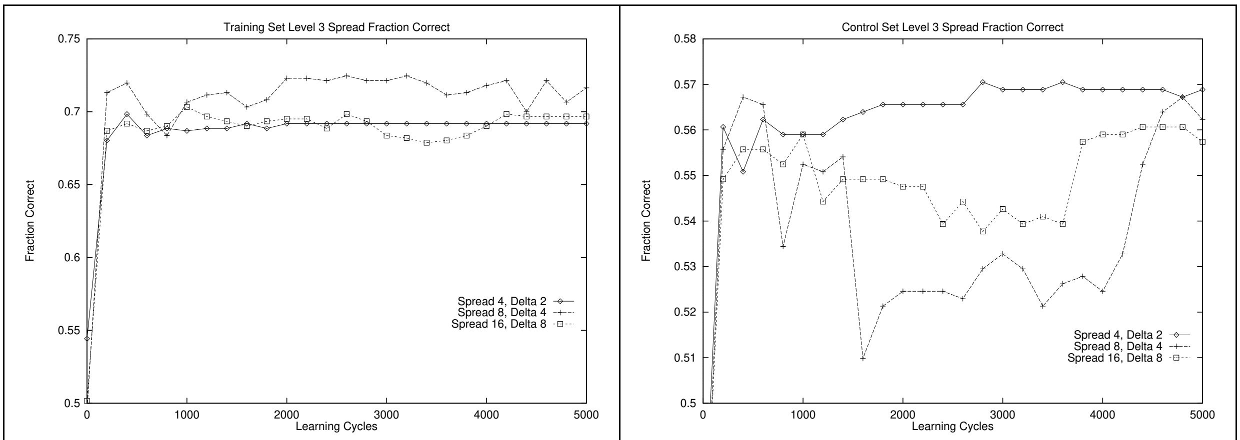


(a) Training Set

(b) Control Set

Figure 25- Effects of Level 2 Deltas on Tournament Auto Rater Fraction Correct

For the level two delta graphs (above) a delta of one does the best for the training set. For the control set, a delta of eight reaches the highest peak early on, although the delta of two network is a close second. This implies that a delta equal to the spread is the best.

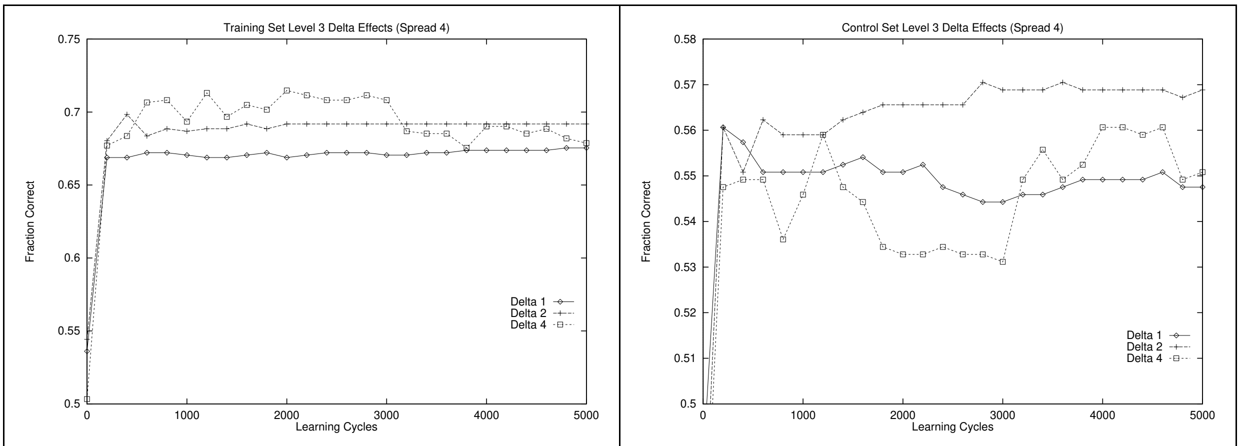


(a) Training Set

(b) Control Set

Figure 26- Effects of Level 3 Spread on Tournament Auto Rater Fraction Correct

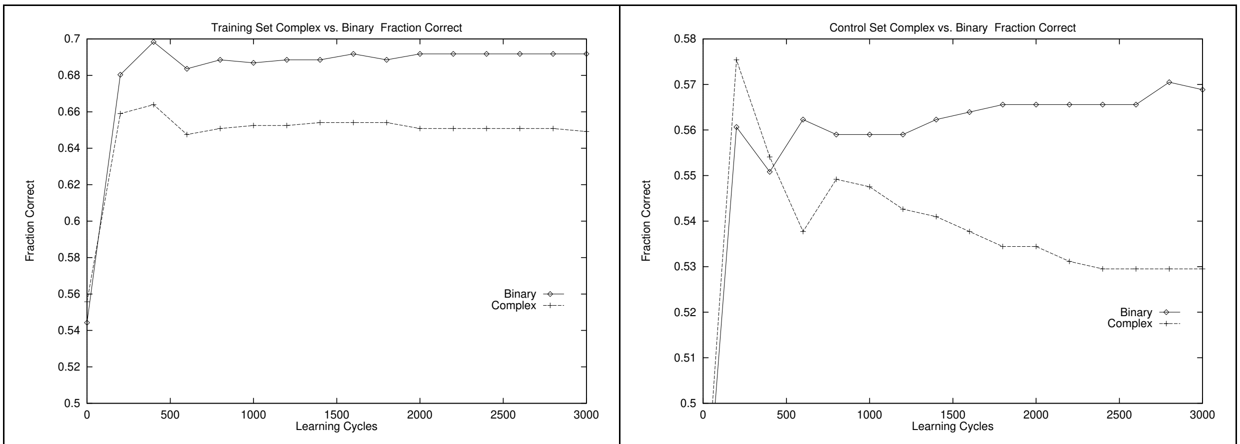
The third level parameter set tests show that a spread of eight does the best for the training set. The control set does the best with the default spread of four.



(a) Training Set (b) Control Set

Figure 27- Effects of Level 3 Deltas on Tournament Auto Rater Fraction Correct

The graphs shown above for the runs to determine the level three deltas show that a delta equal to the spread is best for the training set, while a delta of two, or half the spread is much better for the control set data.



(a) Training Set (b) Control Set

Figure 28- Binary vs. Complex Tournament Auto Rater Fraction Correct

The final parameter set run was to determine if the total fraction correct was better using the binary output and training option, or using the complex option where the degrees of 'betterness' were used. For the training set, the binary output method works consistently better, but for the control set, the complex setting gives a higher peak, even though in later cycles it is worse.

Since the parameters that were shown to the be best were different in all cases except for the level three spread and deltas, a new network was created with better parameter settings. The values used are shown below:

- Level 1 Spread = 16
- Level 1 Delta = 2 (1/8)
- Level 2 Spread = 16
- Level 2 Delta = 4 (1/4)*
- Level 3 Spread = 8*
- Level 3 Delta = 4 (1/2)
- Output type = Binary*

Three of the values used for the best combination were not the optimum values found (they are marked with a '*'). The level two delta value should have been 16, or the same as the spread value, but it was decided that it was better to go with the second best value which maintained some degree of overlap between inputs to adjacent nodes. The level three spread value should have been picked as the default value of four, but early on the data was misinterpreted, and a new run with the correct value has not yet been made. Finally, binary output was chosen since it faired better over a greater numbers of learning cycles. The fraction correct for the control set using the new 'best' parameter individual and the three best from the other runs are plotted below in Figure 29.

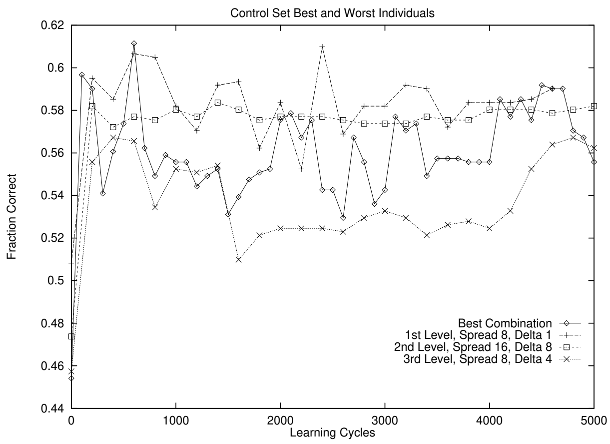
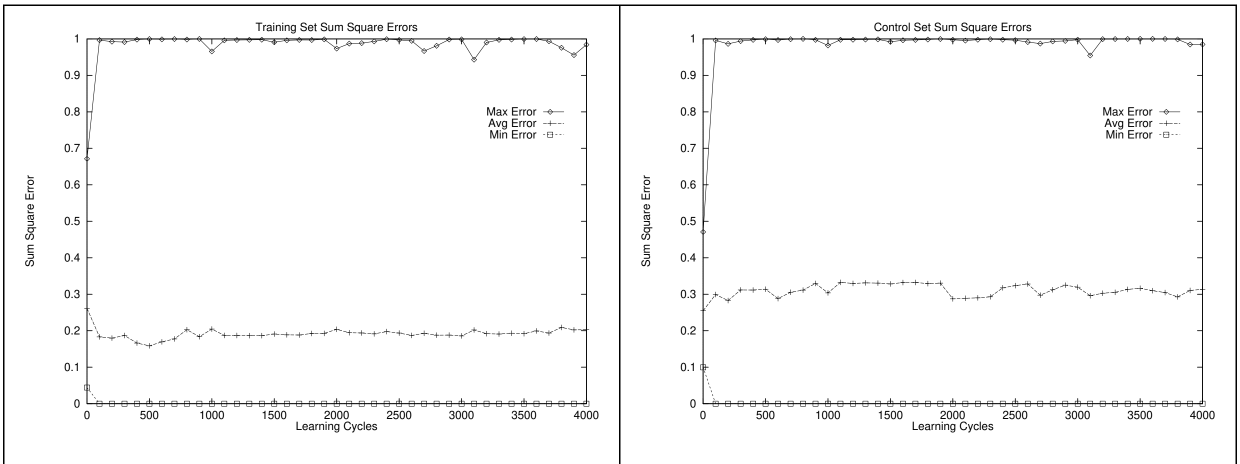


Figure 29- Control Set Fraction Correct for Top Tournament Individuals

From the above graph, one can see that the hybrid 'best' individual does better than any of the others, although the individual with default parameters except for a level one delta of one does almost as well. Looking at the data the best individual gets 61.1% of the comparisons correct, while the other gets 60.9% of the comparisons correct. The next set of graphs look at the performance of the 'best' individual in more detail, starting with Figure 30, below, which shows the minimum, maximum, and average squared error for the training and control sets.



(a) Training Set (b) Control Set

Figure 30- Best Tournament Individual Sum Square Error Rates

For both the training and control sets, the maximum squared error is 1, which means a complete miss on the comparison. The minimum for both is a zero, meaning the network is getting the comparison exactly correct. The next figure shows the average sum square error for both the training and control set on the same graph.

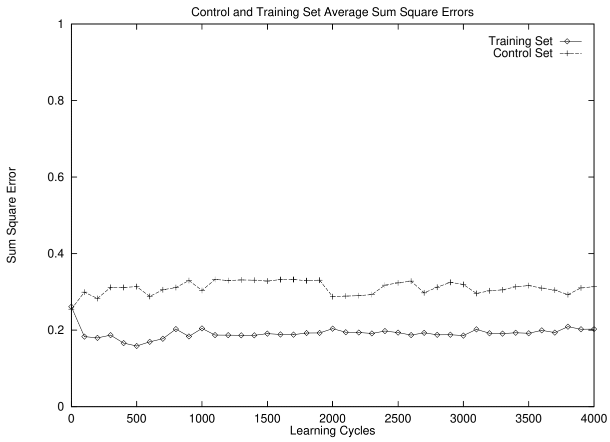


Figure 31- Best Tournament Individual Control and Training Set Sum Square Error

From the graph it can be seen that the average sum squared error stays consistently a bit worse for the control set. It also doesn't seem to improve very much for either set. The next graph shows the fraction correct over many learning cycles for the training and control sets.

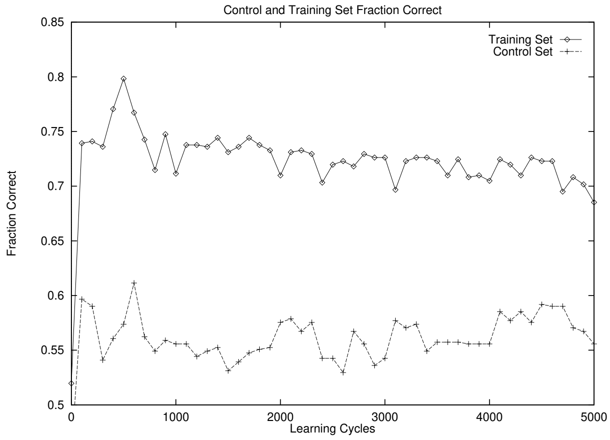
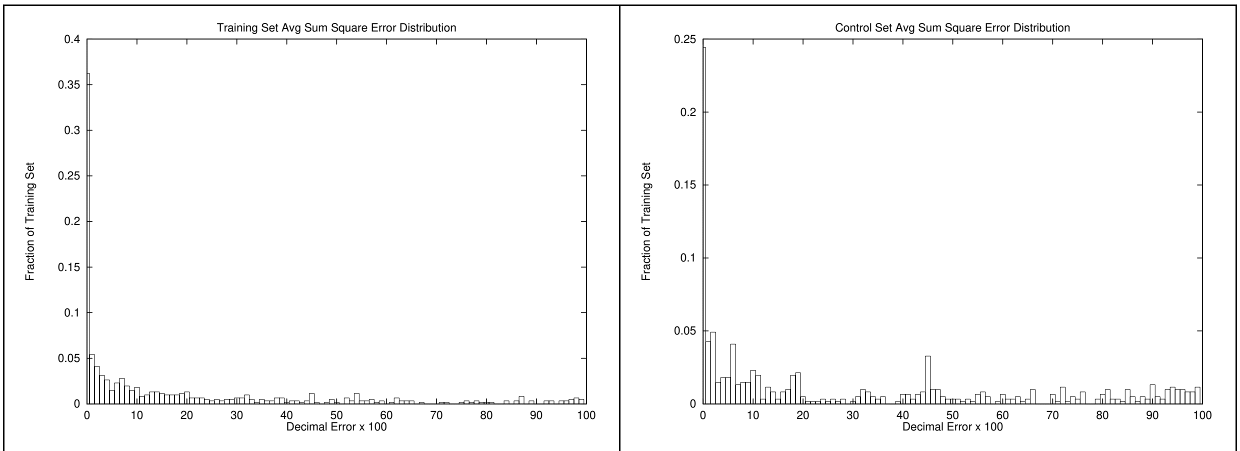


Figure 32- Best Tournament Individual Control and Training Set Fraction Correct

Notice that for both the training and the control sets the peak for the fraction of comparisons correct is made quite early in the learning cycles, and at about the same time. The training set reaches a maximum percent correct of 79.8% at learning cycle 500, while the control set reaches its maximum of 61.1% at cycle 600. As explained earlier, the fraction correct values vary more erratically as border line individuals may increase their error and become incorrect when the error is decreased for other individuals that are already correct.

As with the list auto-rater, there are some statistics that are particular to a given learning cycle, and these were taken for cycle 600, the best for the control set, of the 'best' parameter combination. The first statistic was an indication of the average sum square error for a given combination of lengths in a comparison (for example the error when an individual of size two is compared to an individual of size 40). The interesting thing about this statistic was to see if the network always faired poorly when used to compare sequences mismatched in size. Unfortunately, since graphs of this data are inherently three dimensional, it was almost impossible to interpret the graphs when plotted, so they are not shown here. Inspecting the graph from various rotation angles revealed that there was no special correlation between error and the sequence lengths being compared.

The second statistic taken at a particular cycle was the error distribution for the cycle. This is shown below in Figure 33.



(a) Training Set (b) Control Set

Figure 33- Best Tournament Individual Error Distribution at Cycle 600

The graphs show for each error range (shown on the graph multiplied by 100) what fraction of the set had that amount of sum square error. The data shows that about 35% of the comparisons for the training set, and 25% for the control set have an error of between 0 and 0.01, while the rest is distributed fairly evenly between 0 and 1. Note that the sum square error value is somewhat strange since a squared error of 0.01 is equivalent to a straight error of 0.1.

8.2.3 Trial Run

Even though even the best tournament auto-rater parameter combination was never able to achieve much better than a 60% success rate on comparisons, several runs of the GP-Music System were made using the weights and biases from cycle 600 of the training of the 'best' parameter network. These were made to see how critical it was for the network to get comparisons correct.

The first run used the same settings for the GP-Music System as the human run that generated the tournament rater training data. The best of run individual is shown below in Table 14.

Tournament Auto Rater Run Best Individual Summary		
Individual Statistics:	generation	1/9
	nodes	35
	depth	8
	sequence length	148
	raw fitness	44
Program Tree: (add-space (play-two (play-two (shift-down (shift-down (play-and-mirror F-5))) (play-two (play-twice (play-and-mirror (play-and-mirror (add-space D-Chord)))) (play-and-mirror (play-two (shift-up (play-two G1-Chord D-4)) (add-space (shift-up D-5)))))) (play-two (mirror (mirror (mirror (shift-down (add-space F-Chord))))) (add-space (shift-up (shift-up (shift-down (mirror G-Chord))))))))		
Note Sequence: D-5, RST, D-5, RST, D-4, RST, RST, RST, F#4, RST, RST, RST, A-5, RST, RST, RST, RST, RST, A-5, RST, RST, RST, F#4, RST, RST, RST, D-4, RST, D-4, RST, RST, RST, F#4, RST, RST, RST, A-5, RST, RST, RST, RST, RST, A-5, RST, RST, RST, F#4, RST, RST, RST, D-4, RST, D-4, RST, RST, RST, F#4, RST, RST, RST, A-5, RST, RST, RST, RST, RST, A-5, RST, RST, RST, F#4, RST, RST, RST, D-4, RST, D-4, RST, RST, RST, F#4, RST, RST, RST, A-5, RST, RST, RST, RST, RST, A-5, RST, RST, RST, F#4, RST, RST, RST, D-4, RST, A-4, RST, C-4, RST, E-4, RST, E-4, RST, E-5, RST, RST, RST, RST, RST, E-5, RST, E-4, RST, E-4, RST, C-4, RST, A-4, RST, RST, RST, B-5, RST, RST, RST, G-4, RST, RST, RST, E-4, RST, E-5, RST, RST, RST, C-5, RST, RST, RST, A-5, RST, RST, RST		
Web Site File Name: bej-200-best.au		

Table 14- Tournament Auto Rater Best of Run Individual (9 Generations, 20 individuals per Generation)

The best of run individual is decent, but not very exciting. It was found in generation one, although, as mentioned earlier this means very little when using tournament rating as all generations have the same range of ratings. The best individual of the final generation is shown in the next table.

Tournament Auto Rater Run Last Generation Best Individual Summary		
Individual Statistics:	generation	9
	sequence length	128
Note Sequence: G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4, RST, G-4		
Web Site File Name:	bej-200-best-09.au	

Table 15- Tournament Auto Rater Generation 9 Best Individual (9 Generations, 20 individuals per Generation)

This melody has almost nothing going for it, and almost certainly would have been sorted to the bottom of the list of melodies for any given generation. It has only a single note, and the only thing that makes it even tolerable is some variation in the rhythm. Overall, both sequences created in the 10 generation 20 individual run were much worse than anything created during the equivalent human run.

The second run was made over 50 generations with 100 individuals per generation. The best of run individual is shown in Table 16 and the last generation best individual in Table 17.

Tournament Auto Rater Run Best Individual Summary		
Individual Statistics:	generation	46/50
	nodes	36
	depth	12
	sequence length	52
	raw fitness	44.8

Program Tree:

```
(shift-up
    (shift-down
        (shift-down
            (play-two
                (play-and-mirror (shift-up (add-space (play-two (mirror (shift-up G-Chord))
                                                                (shift-down
                                                                    (shift-down
                                                                       (play-twice D-5)))))))
                (play-two
                    (mirror
                        (play-and-mirror
                            (play-and-mirror
                                (mirror (shift-up (shift-up (mirror E-Chord)))))))
                    (play-and-mirror
                        (play-two (shift-down (play-twice (play-twice (shift-up D-5))))
                                  (shift-up (add-space (shift-up (shift-up E-Chord))))))))))
```

Note Sequence:
E-5, RST, C-5, RST, A-5, RST, B-5, RST, B-5, RST, RST, B-5, RST, B-5, RST, A-5, RST, C-5, RST, E-5, F-4, A-5, C-5, C-5, A-5, F-4, F-4, A-5, C-5, C-5, A-5, F-4, C-5, C-5, C-5, C-5, G-4, RST, B-5, RST, D-5, RST, RST, D-5, RST, B-5, RST, G-4, C-5, C-5, C-5, C-5

Web Site File Name: bej-5000-best.au

Table 16- Tournament Auto Rater Best of Run Individual (50 Generations, 100 individuals per Generation)

Tournament Auto Rater Run Last Generation Best Individual Summary		
Individual Statistics:	generation	50
	sequence length	55

Note Sequence:
A-6, RST, F-5, RST, D-5, RST, A-6, RST, F-5, RST, D-5, RST, RST, D-5, RST, F-5, RST, A-6, RST, D-5, RST, F-5, RST, A-6, E-4, G-4, B-5, A-6, A-6, A-6, A-6, A-6, A-6, A-6, A-6, A-6, A-6, E-5, E-5, D-5, D-5, D-5, D-5, E-5, E-5, A-6, A-6, A-6, A-6, A-6, A-6, A-6, A-6, A-6, A-6, A-6, A-6

Web Site File Name: bej-5000-best-50.au

Table 17- Tournament Auto Rater Generation 50 Best Individual (50 Generations, 100 individuals per Generation)

Neither of the individuals from this run are very good either, both exhibiting stylistic changes throughout their length. The best of run individual is nice at the beginning, but then degenerates for the last part of the sequence.

A final run was made with an even larger population size of 500, again over 50 runs. The best of run individual and best of generation 50 individual are shown below in Table 18 and Table 19, respectively.

Tournament Auto Rater Run Best Individual Summary		
Individual Statistics:	generation	8
	nodes	25
	depth	10
	sequence length	184
	raw fitness	39.6

Program Tree:

```
(mirror
    (play-and-mirror
        (play-and-mirror
            (mirror
                (play-two (add-space
                              (shift-up
                                  (play-twice
                                      (mirror (play-two (play-twice G-5)
                                                        (play-and-mirror E2-Chord))))))
                          (play-and-mirror
                              (play-two (mirror (play-twice (play-two (play-two C-4 C-5)
                                                                      (mirror B-5))))
                                        (mirror G-4))))))))
```

Note Sequence:

B-5, C-5, C-4, B-5, C-5, C-4, G-4, G-4, C-4, C-5, B-5, C-4, C-5, B-5, RST, A-6, RST, A-6, RST, F-5, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, F-5, RST, A-6, RST, A-6, RST, F-5, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, F-5, F-5, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, F-5, RST, A-6, RST, A-6, RST, F-5, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, F-5, RST, A-6, RST, A-6, RST, B-5, C-5, C-4, B-5, C-5, C-4, G-4, G-4, C-4, C-5, B-5, C-4, C-5, B-5, B-5, C-5, C-4, B-5, C-5, C-4, G-4, G-4, C-4, C-5, B-5, C-4, C-5, B-5, RST, A-6, RST, A-6, RST, F-5, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, F-5, RST, A-6, RST, A-6, RST, F-5, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, F-5, F-5, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, F-5, RST, A-6, RST, A-6, RST, F-5, RST, A-6, RST, B-6, RST, B-6, RST, A-6, RST, F-5, RST, A-6, RST, A-6, RST, B-5, C-5, C-4, B-5, C-5, C-4, G-4, G-4, C-4, C-5, B-5, C-4, C-5, B-5

Web Site File Name: bej-25000-best.au

Table 18- Tournament Auto Rater Best of Run Individual (50 Generations, 500 individuals per Generation)

Tournament Auto Rater Run Last Generation Best Individual Summary		
Individual Statistics:	generation	50
	sequence length	76
Note Sequence: RST, E-5, RST, G-5, RST, A-6, RST, C-5, RST, E-5, RST, G-5, RST, A-6, RST, C-5, B-5, F-4, F-4, C-4, RST, F-4, RST, G-4, RST, RST, G-4, RST, F-4, RST, C-4, F-4, F-4, A-5, C-5, E-5, G-5, G-5, G-5, G-5, A-5, D-5, F-5, A-5, D-5, F-5, D-5, RST, E-4, RST, G-4, RST, B-5, RST, RST, B-5, RST, G-4, RST, E-4, B-6, F-4, F-4, E-4, RST, G-4, RST, B-5, RST, RST, B-5, RST, G-4, RST, E-4, B-6		
Web Site File Name: bej-25000-best-50.au		

Table 19- Tournament Auto Rater Generation 50 Best Individual (50 Generations, 500 individuals per Generation)

Both of these individuals are much better sounding than the other runs with the tournament auto-rater. The best of run sequence, found in the eighth generation, is quite interesting to listen to, although not a ‘typical’ melody by any means. The best individual of the 50th generation is also pretty nice to listen to, although it has some of the problem of rapid changes to different styles of music.

Overall, the use of the tournament auto-rater was not very successful. Mostly this is due to the network never getting higher than a rate of 60% in classifying which of two sequences in a comparison was better. Despite this some nice individuals were generated in the run with 500 individual populations and 50 generations. Perhaps finding better ways of training the tournament auto-rater could make it a viable agent to stand in for the human in making the comparisons between sequences.

9. Discussion and Future Work

This section contains some discussion of observations made during the course of the project. It also contains some ideas about how this research could be continued in the future.

9.1 Discussion

The training of the two auto-raters brought up several interesting issues. The first thing of note was that the level spreads and deltas for the two raters were different. Where a spread of eight worked best for the list auto-rater on the lower levels, a spread of 16 worked better for the tournament auto-rater. It would seem that the two should have similar values since the tournament auto-rater is effectively two list auto-raters put together. The only reason for this seems to be that the tournament rater is doing a job that is twice is hard, and needs a correspondingly higher number of degrees of freedom to attempt to correctly classify the comparisons.

Another factor is that in determining optimal parameters the best value for each parameter was determined while holding the others constant. The problem with this is that it assumes that the parameters are orthogonal. It is possible that a level one spread of sixteen might change the optimal level two spread from eight to four, for example. Unfortunately it is very difficult to determine the exact relationship between the parameters since the run times for training the network can be several hours each.

A more general observation can be made about the GP-Music System as a whole. It became clear, especially on the runs made with the auto-raters and many generations, that there is a problem in the system generating offspring sequences from two pleasant sounding sequences that are not very good. In some runs these individuals come to dominate the population through chance, and subsequent generations are quite mediocre by comparison. This is exactly what happened during the human run to generate the tournament auto-rater data. The best individual was found in the first generation, and while pieces of it appeared in subsequent generations, they were mixed in with incompatible sequences, and all of the future generations were poorer than the first. Perhaps increasing the rates of mutation and reproduction could address this problem.

9.2 Future Directions

9.2.1 General Ideas

Although it proved possible to train the list auto-rater to be fairly accurate, the tournament auto-rater never was very good. It would be nice to find a better way to train the tournament auto-rater. One obvious idea is to find a way to propagate the fraction correct during back propagation in order to prevent comparisons which are already being made in the correct direction 'more' correct at the expense of borderline comparisons. This could be done by only back-propagating the error for comparisons being made incorrectly, or back-propagating at a higher rate depending on how far the answer is from the correct one.

Another idea to improve the tournament auto-rater would be to use two copies of the list auto-rater with the output node of each feeding into a new higher level output node which figures out which sub-network is returning a higher rating. In this topology the weights and biases for the two sides would also be kept the same. Effectively the network would be learning a consistent rating scale for the sequences that allows it to answer the question of which sequence is the best.

One idea that could also be tried to improve both auto-raters would be to feed in a separate input, or inputs that give the length of the sequence. While no analysis has been made to determine if there is a correlation between length and the rating for the human raters, personal experience shows that short note sequences with the same note repeating, and long sequences with the same sequence repeating are always rated low. Perhaps there is some way for the networks to be connected to use the sequence length information

A problem with the GP-Music System, rather than the auto-raters is that rating individuals with the tournament interface is extremely long and tedious. If a useful tournament auto-rater is created, it might be easier to let users rank the sequences using the list interface, and then use that information to come up with comparisons that can be used to train the tournament auto-rater.

A final idea that could be attempted if the auto-raters ever become robust enough is to analyze the weight and bias values generated during the network training. They might yield rules about what makes a ‘good’ sequence that could then be used to write a more direct program for creating these short melodies.

9.2.2 Practical Uses

This project has been interesting to undertake from an Artificial Intelligence perspective, since it addresses two important questions:

- Can a computer learn to rate music the same way that a human does?
- Can a computer compose reasonable music entirely on its own?

In the limited context of this project it seems that the answer to both questions is a qualified ‘Yes’. The computer was able to rate music within seven of a humans ranking on a 1-100 scale, and using that information it was able to evolve reasonable short sequences, in effect composing music on its own.

Despite this, a better question is of what practical use is this technology. The answer, and a possible direction to take this research in the future, is that the artificial intelligence described in this report, and embodied in the software written for the project, could become part of a computer assisted compositional tool. An initial human run could be made, and then the computer could train an auto-rater and make longer runs. Best of generation individuals could be collected and pieced together by the user to make longer melodies, or used to form multiple parts of a more complex work. A simple graphical interface could be put together to control how musical sequences are put together. It would also be possible to evolve new individuals from a collection of nice sequences put together to create a starting population. Since the human user does not need any knowledge of music theory, anybody could use this tool.

10. Conclusion

The goal of this project was to create automatic fitness raters to rate musical sequences as stand-ins, or agents, for a human user during runs of the GP-Music System. Further, the hope was to actually run the system using these 'auto-raters' and generate reasonable melodies. Both of these goals were accomplished. This statement must be qualified, however, as the automatic raters were never able to rate precisely as the human user, and the note sequences generated during the runs with the automatic fitness raters varied dramatically in quality. Further, the comparative, or tournament, auto-rater was never able to get better than 60% accuracy when trying to determine which of two sequences was better, and runs with the tournament auto-rater yielded almost no songs of merit. Overall, the use of automatic raters and Genetic Programming to automatically compose music has been proven to be possible in a limited sense, but there is still much work to be done before a computer can be said to compose music of any sophistication or intricacy.

Special Thanks

In addition to my supervisor, special thanks are deserved by Anne Pearce and Philip Underwood. Both participated in runs of the GP-Music System in order to generate rating and comparison information with which to train the automatic fitness raters.

11. Bibliography

1. Beale, R., *Neural Computing: An Introduction,* Institute of Physics Publishing, Bristol and Philadelphia, 1990
2. Benjamin, T., Horvit, M., Nelson, R., *Techniques and Materials of Tonal Music, 2nd Edition,* Houghton Mifflin, Boston, 1979
3. Biles, J. A., "GenJam: A Genetic Algorithm for Generating Jazz Solos"
4. Burton, A. R., Vladimirova, T., "Genetic Algorithm Utilising Neural Network Fitness Evaluation for Musical Composition," *ICANNGA 97 Abstracts*, p. 23, 1997
5. Cope, D., "An Expert System for Computer-assisted Composition," *Computer Music Journal,* Vol. 11, No. 4, pp. 30-46, 1987
6. 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
7. Johanson, B. E., "The GP-Music System: Interactive Genetic Programming for Music Composition," University of Birmingham, Second-Semester Mini-Project Report, 1997
8. Koza, J. R., *Genetic Programming 2: Automatic Discovery of Reusable Programs,* The MIT Press, Cambridge, MA., 1994
9. Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection,* The MIT Press, Cambridge, MA., 1992
10. Poli, R., Cagnoni, S., "Evolution of Pseudo-colouring Algorithms for Image Enhancement with Interactive Genetic Programming," University of Birmingham, Technical Report CSRP-97-5, 1997
11. Rahn, J., "The Lisp Kernel: A Portable Software Environment for Composition," *Computer Music Journal,* Vol. 14, No. 4, pp. 42-58, 1990
12. Rowe, R., *Interactive Music Systems: Machine Listening and Composing,* The MIT Press, Cambridge, MA., 1993
13. Schottstaedt, B., "PLA: A Composer's Idea of a Language," *Computer Music Journal,* Vol. 7, No. 1, pp. 11-20, 1983

14. Spector, L., Alpern, A., “Induction and Recapitulation of Deep Musical Structures,” *Proceedings of the IJCAI-95 Workshop on Music and AI*

15. Todd, P. M., “A Connectionist Approach to Algorithmic Composition,” *Computer Music Journal,* Vol. 13, No. 4, pp. 27-43, 1989

16. Zhao, T.C., Overmars, M., *Xforms library for Xlib,* 1995

17. Zongker, D., Punch, B., Rand, B., *lil-gp 1.01*, Michigan State University, 1996