Induction and Recapitulation of Deep Musical Structure [*]

Lee Spector and Adam Alpern
School of Cognitive Science and Cultural Studies
Hampshire College
Amherst, MA 01002
U. S. A.
{lspector, aalpern}@hampshire.edu

Abstract

We describe recent extensions to our framework for the automatic generation of music-making programs. We have previously used genetic programming techniques to produce music-making programs that satisfy user-provided critical criteria. In this paper we describe new work on the use of connectionist techniques to automatically induce musical structure from a corpus. We show how the resulting neural networks can be used as critics that drive our genetic programming system. We argue that this framework can potentially support the induction and recapitulation of deep structural features of music. We present some initial results produced using neural and hybrid symbolic/neural critics, and we discuss directions for future work.

1 Introduction

In previous work we developed a framework for the automatic generation of art-making programs on the basis of user-provided critical criteria [Spector and Alpern, 1994]. Our implementation of this framework used *genetic programming* technology developed by Koza [Koza, 1992] and in principle could be applied to the production of artworks in a variety of arts media. To demonstrate our framework we produced a system that automatically constructs an interactive jazz improvisation program when presented with a user-provided music critic. We ran the system with a music critic that judged compliance with a small number of rules found in a popular jazz method book [Baker, 1988].

The resulting programs performed acceptably but not spectacularly. We attributed the weakness of the constructed jazz musicians to the weakness of the music critics that guided their construction. In our current work we are attempting to improve the performance of our automatically generated improvisation programs by improving our music critics. We are using connectionist techniques to produce critics that make judgements about the quality of a musical fragment relative to a corpus. We use the resulting critics to drive the evolution of new music-making programs.

Previous work in algorithmic composition has produced programs that induce musical structure from a corpus and then recapitulate that structure in new pieces of music (for example [Cope, 1991] and [Rowe, 1993] — Chapter 7 of [Rowe, 1993] contains a good survey of related work). The induced and recapitulated structure generally captures relatively shallow surface features of the input (e.g., frequently occurring interval sequences). We view our current project as an extension of such work to deeper structural levels. Neural network-based critics are, in principle, capable of inducing rich and complex structural features of their musical input. Our genetic programming framework is, in principle, capable of producing *any* music-processing program; it is Turing complete. We believe that our combination of neural and genetic techniques will allow for the induction and recapitulation of deeper structural features of music than have been captured in previous systems.

A similar marriage of neural and genetic techniques has been reported by Baluja et al. [Baluja *et al.*, 1994]. Our project differs from theirs in that we are evolving music-making programs while Baluja et al. evolved images. In addition, we are training networks to judge similarity to a corpus of well-known works, while Baluja et al. trained their networks to replicate user preferences.

In the remainder of this paper we describe our genetic programming framework for the automatic generation of music-making programs and the connectionist techniques that we are using for our new critics. We then provide an example and discuss directions for additional work.

2 Automatic Generation of Artists

Our intention in developing a framework for the automatic generation of art-making programs was to separate those components of an AI art-making system to

[*]To appear in the proceedings of the IJCAI-95 Workshop on Artificial Intelligence and Music, 1995.

which aesthetic judgement should apply from those to which scientific judgement should apply [Spector and Alpern, 1994]. The essential move was to suggest that AI art systems take aesthetic critical criteria *as parameters*. Such systems can produce art-making programs to satisfy a range of different aesthetic criteria; we call the resulting programs *constructed artists*. The extent to which a particular constructed artist satisfies a particular set of explicit aesthetic criteria is a matter of scientific, rather than aesthetic, judgement. The impact of culture on the assessment of artworks presents problems similar to those of varying aesthetic criteria; therefore we also suggested that AI art systems take "cultures" (usually case-bases of past works) *as parameters*. Instances of our framework take critics and cultural contexts as input and produce constructed artists as output.

3 Genetic Programming of Artists

The technology of genetic programming [Koza, 1992] provides a straightforward way to implement our framework. Genetic programming is a technique for the automatic generation of computer programs; in our case we can use the technique to automatically generate computer programs that will function as constructed artists. Genetic programming is an evolutionary method in which programs are evolved using a process modeled on Darwinian natural selection. The process of natural selection is driven by fitness; that is, by some assessment of the quality of each individual. Genetic programming systems take fitness functions as parameters. Any function that maps programs to fitness values may be used as a fitness function. For the production of constructed musicians we can provide music critics as parameters to the system in the form of fitness functions [Spector and Alpern, 1994].

The genetic programming process starts by creating a large initial population of programs that are random combinations of elements from problem-specific function and terminal sets. Each of the programs in the initial population is assessed for fitness. This is usually accomplished by running the fitness function on each program with a collection of inputs called fitness cases. The fitness values are used in producing the next generation of programs via a variety of genetic operations including reproduction, crossover, and mutation. Individuals are randomly selected for participation in these operations, but the selection function is biased toward highly fit programs.[1] The reproduction operator simply selects an individual and copies it into the next generation. The crossover operation introduces variation by selecting two parents and by generating from them two offspring; the offspring are produced by swapping random fragments of the parents. The mutation operator produces one offspring from a single parent by replacing a randomly selected program fragment with a newly generated random fragment.

Over many generations of fitness assessment, reproduction, crossover, and mutation, the average fitness of the population may tend to improve, as may the fitness of the best-of-generation individual from each generation. After a preestablished number of generations, or after the fitness improves to some preestablished level, the best-of-run individual is designated as the result and is produced as the output from the genetic programming system.

4 Task and Representation

In our previous work we generated programs that produced four-measure melodies as output when given four-measure melodies as input. This corresponds to the popular practice of "trading four" in jazz improvisation. In our current work we are generating single-measure responses to single-measure calls.

We represent melodies as vectors of (articulation, note) pairs, 48 per measure, where each articulation is above a threshold if a new note starts in the corresponding quantum and below the threshold otherwise, and where each note is a scaled MIDI note number or 0 for a rest. This allows for all standard durations down to 16th notes and 32nd-note-triplets. It has the additional virtue of being a fixed-length representation; while neither our genetic programming framework nor criticism by neural networks *requires* fixed-length representations, both can be simplified if fixed-length representations are used.

5 Function and Terminal Sets

In our previous work we evolved programs consisting of a single terminal, CALL-MELODY, and collection of special-purpose melody transformation functions such as RETROGRADE, DIMINUTE and FRAGMENT. Some of the functions, such as MOST-FAMILIAR, accessed a case-base of known melodies. Each function took one or more melodies as input and returned a melody as output. Each program was a nested expression of function calls, and the melody returned by the entire expression was interpreted as the response for the given CALL-MELODY.

In our current work we are using more generic function and terminal sets; our terminals are integers and our functions are numerical operators, control structures, and memory accessors. The call melody is provided in the form of an indexed memory [Teller, 1994]; a program may access the call melody through the use of a CALL-COPY function. Similarly, the response melody is constructed in a separate indexed memory. The case-base of known melodies is represented as arrays of indexed memories accessible through CASE-CALL-COPY and CASE-RESPONSE-COPY. All numerical values in the system are in the range from 0 to 95, inclusive, and all indexed

[1]For the work described in this paper we used tournament selection [Koza, 1992] with tournament group sizes between 4 and 7.

memories are of length 96 and use 0-based addressing. The full function set used for the runs described in this paper was as follows:

- +: a two-argument addition operator that returns the sum of its arguments modulo 96.

- IF-LESS: a four-argument conditional control structure that evaluates and returns the result of the body of code that appears as its third argument if the value of its first argument is less than that of its second argument; it evaluates and returns the result of the body of code that appears as its fourth argument otherwise.

- DO-TIMES: a two-argument iteration control structure that evaluates the body of code that appears as its second argument as many times as is specified by the first argument. The value of the last iteration is returned as the result of DO-TIMES. During each iteration the variable i is set to the iteration number, starting with 0.

- COPY: a three-argument function that copies a block of the response to another region in the response. The first two arguments determine the endpoints of the block to be copied, and the third argument specifies the index for the new copy. The value copied from the high endpoint of the block is returned as the value of the function call.

- CALL-COPY: a two-argument function that copies a block of the call to the response. The arguments determine the endpoints of the block to be copied; the block is copied to the *same* location in the response. The value copied from the high endpoint of the block is returned as the value of the function call.

- CASE-CALL-COPY: a three-argument function that copies a block of a call from the case base to the response. The first argument specifies the case number and the second and third arguments determine the endpoints of the block to be copied. The block is copied to the *same* location in the response. The value copied from the high endpoint of the block is returned as the value of the function call.

- CASE-RESPONSE-COPY: a three-argument function that copies a block of a response from the case base to the response. The first argument specifies the case number and the second and third arguments determine the endpoints of the block to be copied. The block is copied to the *same* location in the response. The value copied from the high endpoint of the block is returned as the value of the function call.

- TRANSPOSE: a three argument function that adds the value of its first argument to all elements of the response between the indices specified by the second and third arguments. The transposition interval is returned as the value of the function call.

The terminal set consists of the integers from 0 to 95 and the loop variable i (which has the value 0 outside of any loop).

Each evolving program also includes two *automatically defined functions* [Koza, 1994], ADF0 and ADF1, each of which takes three arguments, ARG0, ARG1, and ARG2. The function set for the main program therefore also includes ADF0 and ADF1. The function and terminal sets for the automatically defined functions are identical to those for the main program except that ADF1 can call ADF0 but not itself, ADF0 cannot call any automatically defined functions, and the terminal sets for both ADF0 and ADF1 include the arguments ARG0, ARG1, and ARG2.

Our reformulation of the function and terminal sets has several advantages over our previous scheme. First, it allows for significant efficiency improvements with respect to memory allocation and the complexity of case-base access. Second, it removes the biasing influence of preestablished melody transformations. Third, since a response can be read from the response memory at any point during a run, it allows for the production of "anytime" response-generating programs.

6 Fitness Assessment

In our previous work we assessed the fitness of each program by running it with a collection of Charlie Parker melodies as input. Each run produced a response melody and each call/response pair was assessed on the basis of a set of critical criteria inspired by those presented in [Baker, 1988]. The criteria were TONAL-NOVELTY-BALANCE, RHYTHMIC-NOVELTY-BALANCE, TONAL-RESPONSE-BALANCE, SKIP-BALANCE, and RHYTHMIC-COHERENCE [Spector and Alpern, 1994].

Our genetic programming system produced a response-generating program with behavior that satisfied the provided critic. It was not, however, completely satisfactory; about one of the program's responses we wrote:

> Although the response ... pleases the critic, it does not please us (the authors) particularly well. This is not an indication of weakness of the genetic programming approach to musician construction. Nor is it an indication that we made improper choices (of function set, terminal set, etc.) in applying the technique; it just means that we should work to improve the critical criteria that we provide as parameters to the system. [Spector and Alpern, 1994]

The primary goal of our current work is to improve the music critic that drives the evolutionary process. We are using connectionist techniques to automatically induce structural principles underlying a corpus of jazz melodies. The resulting trained networks are used as music critics in our genetic programming framework. The genetic programming system is run with the networks as fitness functions, producing response-generating programs. Note that neither the neural networks nor the genetic operators appear in the final response-generating programs.

The task that we intend our networks to perform is general and difficult. We want the networks to induce sufficiently many relevant structural features of a corpus to distinguish reasonable from unreasonable melodies. It is reasonable to assume that this task would be achieved more easily if divided into subtasks such as pitch structure and rhythm analysis. A wide range of network architectures have been used for related music classification tasks [Page, 1994; Todd and Loy, 1991], and it is clear that even these subtasks are quite difficult.

The work described in this paper is only a first step in the use of neural network critics in our genetic programming framework, and we used a simple network that is clearly not adequate for the full task of inducing the structure of jazz melody. It is nonetheless useful for demonstrating our amalgam of genetic and connectionist techniques, and it can be put to good use as one component of a multi-network critic or a hybrid connectionist/symbolic critic (see below).

We used a three layer network with 192 input units (one for each articulation and for each note value over a two measure fragment), a single layer of 96 hidden units, and 2 output units. We trained the networks on four categories of input. The first category consisted of two-measure fragments of Charlie Parker melodies, the second consisted of single measures of Charlie Parker followed by single measures of silence, the third consisted of single measures of Charlie Parker followed by single measures of random melody, and the fourth consisted of single measures of Charlie Parker followed by reversed and randomly manipulated Charlie Parker continuations. Our networks were trained to respond in a positive way only to inputs from the first category; the intention was to train the networks to recognize reasonable continuations to reasonable fragments of jazz melody.

We trained the network, using Fahlman's Quickprop algorithm [Fahlman, 1988], to respond with the vector `(1 0)` to the "good" inputs and `(0 1)` to the "bad" inputs. We presented a total of 100 input patterns during training and our network rapidly converged to minimal error. Good inputs from the training set reliably produce outputs close to `(1 0)` and bad inputs from the training set reliably produce `(0 1)`.

In spite of the rather simple network architecture and modest training set, our network appears to have achieved some degree of generalization. Newly generated random inputs reliably produce outputs close to `(0 1)`. Fragments of Charlie Parker melodies not used in the training set produce outputs that are more affirmative than negative; for example, the first two pairs of measures from Ornithology, which were not used during training, produce approximately `(0.55 0.00)` and `(1.00 0.00)` respectively. The network is *not* merely recognizing randomness or the lack thereof; good melodies from the training set produce outputs close to `(0 1)` if they are presented to the network reversed. This holds even when the "reverse" operation is modified to maintain odd-numbered positions in the input vector as articulation positions and even-numbered positions as note-value positions.

It appears, however, that our network is not so much recognizing Charlie Parker melodies as it is recognizing instances of some more generalized notion of musicality. For example, the first pairs of measures from "Purple Haze" and "Bold as Love" by Jimi Hendrix produce `(0.99 0.00)` and `(0.97 0.08)` respectively.

7 Breeding a Fit Musician

We used our trained network as the fitness function for the genetic programming system described above. The two-component network outputs were converted into "standardized fitness" values [Koza, 1992], for which lower values are "more fit," by adding the second component to the result of subtracting the first component from 1.0. Melodies judged to be terrible by the network produce outputs close to `(0 1)`; such outputs translate to standardized fitness values close to 2. Melodies judged to be excellent by the network produce outputs close to `(1 0)`; such outputs translate to standardized fitness values close to 0.

We ran the genetic programming system with a population size of 400 and assessed each program with respect to four fitness cases. Each fitness case was a call melody from the case base, which contained 29 two-measure Charlie Parker fragments.[2] The fitness of each program was assessed by running it on each of the four call melodies. The resulting call/response pairs were then submitted to the neural network critic, and the resulting standardized fitness values were summed. This produced a final fitness value for each program between 0.0 and 8.0.

The genetic programming system produced a program with a perfect fitness value of 0.0 after only one generation of reproduction, crossover and mutation. The evolved program was the following:

```
(TRANSPOSE
```

[2] `CASE-CALL-COPY` and `CASE-RESPONSE-COPY` take their case indices modulo the actual number of cases, so any number of cases (up to 96) is acceptable.

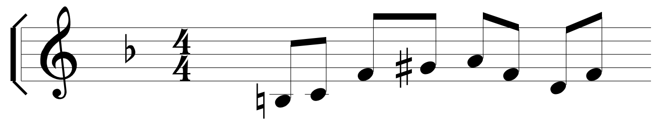
```
 (+ (IF-LESS (IF-LESS 16 14 35 86)
             (CASE-RESPONSE-COPY 38 i i)
             (IF-LESS 57 33 60 i)
             (ADF0 i 39 6))
    (CASE-RESPONSE-COPY
     (TRANSPOSE i i i)
     (IF-LESS i 67 94 86)
     95))
 (ADF1 78 86 41)
 (DO-TIMES (IF-LESS
            20
            (DO-TIMES 10 i)
            (TRANSPOSE i 11 i)
            (CASE-RESPONSE-COPY i 63 i))
           (COPY 28 (ADF0 67 i i) (+ i i))))
```

The two automatically defined functions that evolved as a part of this program were the following:

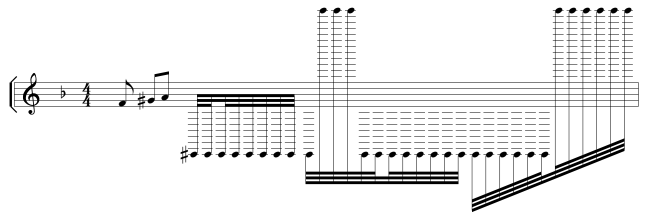
```
(DEFUN ADF0 (ARG0 ARG1 ARG2)
  (CALL-COPY
   ARG2
   (COPY (COPY i ARG0 (IF-LESS i i i ARG1))
         (TRANSPOSE
          ARG1
          (CASE-CALL-COPY 0 79 ARG2)
          (+ ARG2 ARG1))
         i)))

(DEFUN ADF1 (ARG0 ARG1 ARG2)
  (+ (CASE-RESPONSE-COPY
      (COPY ARG1 (IF-LESS i 65 ARG2 66)
            (ADF0 18 57 22))
      (ADF0 (DO-TIMES ARG1 ARG0)
            (CASE-RESPONSE-COPY ARG1 i ARG2)
            (DO-TIMES ARG0 i))
      i)
     (CALL-COPY
      (COPY (ADF0 i i ARG2)
            (CASE-RESPONSE-COPY i ARG1 ARG1)
            60)
      (+ ARG2 (CALL-COPY i ARG0)))))
```

As is often the case with the results of genetic programming, this program is difficult to assess by visual inspection. Unfortunately, it turns out that its behavior is quite unsatisfactory. Its responses jump erratically between octaves, include notes that fall irrespective of beat divisions, and vary little between calls. The following is a representative call/response pair.

The call, the first measure of Billie's Bounce, was used as one of the fitness cases during evolution:



The response, notated approximately (all notes except the second and third are actually triplets):



In retrospect it is clear that the network had far too small a training set to learn about many of these kinds of errors; perhaps it would have been a better critic if it had also been trained to reject melodies that are reasonable *except* for bizarre rhythmic groupings, etc. In any event it is clear that the network is far less generalized than one might have guessed from the casual testing that we performed prior to running the genetic programming system. The genetic programming system is a strong test on the generalization of a critic; if there is a simple way to exploit a weakness of the critic then it is likely that genetic programming will find it.

8 A Community of Critics

One implication of the failure of our first experiment is that far more competent criticism is required if our musician-producing system is to be properly constrained. One way to get more competent criticism is to provide a neural network critic with a more capable architecture and learning procedure, and to train it on larger training sets. Another way to get more competent criticism is to form a *community* of critics, each of which has a complementary type of expertise.

The symbolic criteria that we used in our earlier work had the virtue of transparency and conformance to intuition. While it would be difficult to argue that they captured "deep" structural features of music, it is clear that music that satisfies these criteria would likely be "well formed" at least at a shallow structural level. Our neural network critic has complementary weaknesses and strengths; it is opaque and produces music with inappropriate surface structure, but it seems to have learned *something* about the deeper structural principles of melody. Whatever knowledge it embodies has been automatically induced from a corpus and is therefore relatively free from the biases of the programmer.

A nice feature of our framework is that multiple critics can be combined by simple addition of fitness values. This works with symbolic critics as well as with other neural network-based critics. We therefore tried running our genetic programming system using a hybrid symbolic/neural critic. We used the same network as in our previous experiment and then added the following quantities to the standardized fitness:

- *skip-balance*: an indication of the balance between diatonic movement (intervals of less than three) and "skips" (intervals of size 3 or greater). The value is 0 if there is are the same number of each type of interval, 1 if all intervals are of one type or the other, and an intermediate value for intermediate degrees of balance.

- *call-response-balance*: an indication of the point-for-point similarity of call and response. The value is 0 if there is a perfect balance of matching and non-matching points, 1 if the call and response are either identical or completely dissimilar, and an intermediate value for intermediate degrees of balance.

- *response-response-balance*: an indication of the point-for-point similarity of responses to different calls, calculated once for each *pair* of call/response pairs. The value is 0 if there is a perfect balance of matching and non-matching points between responses, 1 if the two responses are either identical or completely dissimilar, and an intermediate value for intermediate degrees of balance.

The genetic programming system had a more difficult time finding a program to satisfy the hybrid critic, but the resulting program shows far more promise than that produced from the neural network critic alone. On one run with a population size of 250 the system produced the following program on generation 42:

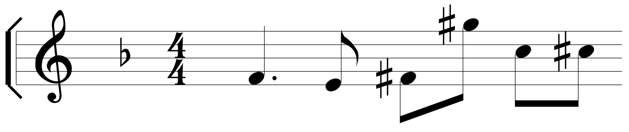
```
(CASE-RESPONSE-COPY
 (IF-LESS
  (COPY (COPY i 53 i)
        (TRANSPOSE
         (CALL-COPY (+ i 79) i)
         (CALL-COPY i 95)
         (ADF0 (CASE-RESPONSE-COPY 59 81 i)
               (TRANSPOSE i i i)
               (DO-TIMES
                (CASE-CALL-COPY 42 77 i)
                (CASE-CALL-COPY i i i))))
        (+ 36 37))
  i
  (DO-TIMES (CALL-COPY i 95)
            (IF-LESS i 56 i 8))
  (DO-TIMES (ADF0 i 34 i)
            (IF-LESS i 51 i i)))
 i
 (TRANSPOSE
  (CALL-COPY (+ i 79) (COPY i 53 i))
  (COPY i 53 i)
  (ADF0 (CASE-RESPONSE-COPY 59 81 i)
        (TRANSPOSE i i i)
        (COPY i 20 i))))
```

The automatically defined function ADF1 is not called in this program. ADF0, which is called several times from the main program, evolved the following complex definition:

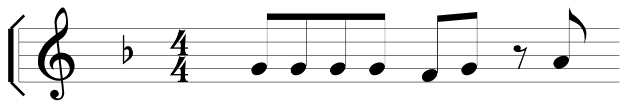
```
(DEFUN ADF0 (ARG0 ARG1 ARG2)
  (CASE-RESPONSE-COPY
   32 (TRANSPOSE
       (COPY ARG1 (+ i ARG1)
             (TRANSPOSE 67 i ARG2))
       i i)
   (TRANSPOSE
    (CASE-CALL-COPY
     (COPY i i i)
     (CASE-RESPONSE-COPY
      (CASE-CALL-COPY
       (COPY i i i)
       (CASE-RESPONSE-COPY
        (CASE-RESPONSE-COPY
         (+ (CASE-CALL-COPY ARG1 i ARG2)
            (CALL-COPY ARG1 ARG1))
         (CASE-RESPONSE-COPY
          (+ 25 ARG1) (+ 7 ARG0)
          (TRANSPOSE i ARG2 ARG2))
         (TRANSPOSE i ARG2 ARG2))
        (CASE-RESPONSE-COPY
         (+ 25 ARG1) (+ 7 ARG0)
         (CASE-CALL-COPY
          (COPY i i i)
          (CASE-RESPONSE-COPY
           (+ (CASE-CALL-COPY ARG1 i ARG2)
              (CALL-COPY ARG1 ARG1))
           (CASE-RESPONSE-COPY
            (+ 25 ARG1) (+ 7 ARG0)
            (TRANSPOSE i ARG2 ARG2))
           (TRANSPOSE i ARG2 ARG2))
          (CASE-RESPONSE-COPY
           ARG2 ARG2 ARG2)))
        (TRANSPOSE i ARG2 ARG2))
       (CASE-RESPONSE-COPY ARG2 ARG2 ARG2))
      (CASE-RESPONSE-COPY
       (+ 25 ARG1) (+ 7 ARG0)
       (TRANSPOSE i ARG2 ARG2))
      (CASE-CALL-COPY
       (CALL-COPY ARG1 i)
       (CASE-CALL-COPY 44 ARG2 ARG2)
       (+ 7 ARG0)))
     (CASE-RESPONSE-COPY ARG2 ARG2 ARG2))
    (COPY ARG2
          (CASE-CALL-COPY ARG0 ARG0 13) 63)
    (CALL-COPY 1 (CALL-COPY 59 54)))))
```

This run was conducted with two fitness cases. The highest (worst) possible fitness value was therefore 9: for each fitness case we produced a value from the neural network (maximum 2 each), a value from the skip-balance function (maximum 1 each) and a value from the call-response-balance function (maximum 1 each); we then added the result of a single call to the response-response-balance function (maximum 1). The best-of-run program, listed above, had a fitness value of 2.027. A perfect score of 0.0 may in fact be unobtainable, as various components of the hybrid fitness function may be inconsistent with one another in practice.

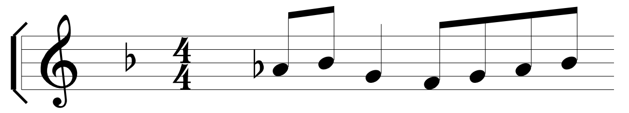
The program that was evolved using the hybrid critic behaves better than the program evolved using the pure neural network-based critic, although it still leaves much to be desired. Its responses appear to conform to reasonable constraints on musical structure and to relate to the calls in interesting if unusual ways. For example, the program produces the following response to the first measure of Billie's Bounce, which was again used as a fitness case during evolution:



The following call, the first measure of My Little Suede Shoes (also by Charlie Parker), was *not* used as a fitness case during evolution:



The program that was evolved using the hybrid critic produces the following response for this call:



9 Conclusions and Future Work

While we have not yet succeeded in inducing and recapitulating the deep structure of jazz melody, we believe that our framework holds promise for the eventual achievement of this goal. The quality of our automatically generated music-making programs is driven by the quality of the music critics that serve as fitness evaluators for genetic programming. We have shown how such critics can be constructed using explicit symbolic rules, neural networks, and hybrids of the two. The use of neural networks in this context is particularly interesting in that it allows us to *automatically* induce the structural principles that genetic programming will later *automatically* recapitulate.

As we have seen, the networks must be quite competent if they are to provide the necessary guidance to the genetic programming process; genetic programming will often find and exploit bizarre niches produced by weaknesses in fitness functions. One short-term solution to this problem is to augment neural network critics with symbolic critical criteria that help to ensure some degree of "well-formedness." The longer-term solution is to improve the network critics by drawing on the considerable past work on connectionist models of music processing [Todd and Loy, 1991]. This is one of our immediate research priorities. As we improve our neural network architectures we also intend to begin working with longer musical fragments.

A virtue of our framework is that any number of independent critics, each of which may use a different set of techniques and may specialize in different aspects of musical structure, can be combined to allow their collective wisdom to guide the evolutionary process. The only constraint on the critics in such a "community" is that they each must be capable of producing a single numerical value fitness value. We intend to collect critic functions from others in the research community and to allow communities of these critics to drive the evolution of new music-making programs.

We are also experimenting with enhancements to the genetic programming component of our system. For example, we are exploring trade-offs in general vs. special purpose function and terminal sets, and we are examining a possible role for automatically defined macros [Spector, 1995].

Acknowledgments

Jay Garfield first suggested the use of neural network-based critics in our framework. James Carlson and Garth Zenie provided valuable feedback on the project as it progressed. Wil Bonner and Aaron Culich helped with neural network simulation software and with system integration problems.

References

[Baker, 1988] Baker, D. 1988. *David Baker's Jazz Improvisation*, Revised Edition. Alfred Publishing Co., Inc.

[Baluja *et al.*, 1994] Baluja, S., Dean Pomerleau, and Todd Jochem. 1994. Towards Automated Artificial Evolution for Computer-generated Images. In *Connection Science*, Vol. 6, No. 2 & 3, 325-354.

[Cope, 1991] Cope, D. 1991. *Computers and Musical Style.* Madison, Wisconsin: A-R Editions, Inc.

[Fahlman, 1988] Fahlman, S.E. 1988. An Empirical Study of Learning Speed in Back-Propagation Networks. In *Proceedings of the 1988 Connectionist Models Summer School.* Morgan-Kaufmann.

[Koza, 1992] Koza, J.R. 1992. *Genetic Programming.* Cambridge, MA: The MIT Press.

[Koza, 1994] Koza, J.R. 1994. *Genetic Programming II.* Cambridge, MA: The MIT Press.

[Page, 1994] Page, M.P.A. 1994. Modelling the Perception of Musical Sequences with Self-organizing Neural Networks. In *Connection Science*, Vol. 6, No. 2 & 3, 223-246.

[Rowe, 1993] Rowe, R. 1993. *Interactive Music Systems: Machine Listening and Composing.* Cambridge, MA: The MIT Press.

[Spector, 1995] Spector, L. 1995. Evolving Control Structures with Automatically Defined Macros. Submitted to the 1995 AAAI Fall Symposium on Genetic Programming.

[Spector and Alpern, 1994] Spector, L., and A. Alpern. 1994. Criticism, Culture, and the Automatic Generation of Artworks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94*, 3-8. Menlo Park, CA and Cambridge, MA: AAAI Press/The MIT Press.

[Teller, 1994] Teller, A. 1994. The Evolution of Mental Models. In K. Kinnear, Jr., Ed., *Advances in Genetic Programming*, 199-219. Cambridge, MA: The MIT Press.

[Todd and Loy, 1991] Todd, P.M., and D.G. Loy. 1991. *Music and Connectionism*. Cambridge, MA: The MIT Press.