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*, pp. 3-8. Menlo Park, CA and Cambridge, MA: AAAI Press/The MIT Press.

Criticism, Culture, and the Automatic Generation of Artworks[1]

Lee Spector and **Adam Alpern**

School of Communications and Cognitive Science
Hampshire College, Amherst, MA 01002
{lspector, aalpern}@hamp.hampshire.edu

Abstract

Researchers wishing to create computational systems that *themselves* generate artworks face two interacting challenges. The first is that the standards by which artistic output is judged are notoriously difficult to quantify. The larger AI community is currently involved in a rich internal dialogue on methodological issues, standards, and rigor, and hence murkiness with regard to the assessment of output must be faced squarely. The second challenge is that any artwork exists within an extraordinarily rich cultural and historical context, and it is rare that an artist who is ignorant of this context will produce acceptable works. In this paper we assert that these considerations argue for case-based AI/Art systems that take critical criteria as parameters. We describe an example system that produces new bebop jazz melodies from a case-base of melodies, using genetic programming techniques and a fitness function based on user-provided critical criteria. We discuss the role that such techniques may play in future work on AI and the arts.

Introduction: Constructing Artists

Applications of computers to the arts date from the earliest days of computing. The use of AI technologies in the arts has a long history as well, particularly in music (Balaban et al. 1992). The majority of these uses fall into two categories: systems that perform "art understanding" tasks of some sort (e.g., music analysis systems), and systems that function as "intelligent" tools for use by human artists (e.g., (Rowe 1993)). Recently, however, a new category of systems has begun to emerge; a category of systems that are designed to *be* artists. By this we mean that such systems, which we will call "constructed artists," are supposed to be capable of creating aesthetically meritorious artworks on their own, with minimal human intervention. Harold Cohen's Aaron system is an early example of this category, and one of its few clear successes to date (McCorduck 1991). Aaron is a system that creates original drawings, each unique and potentially surprising to Cohen, that have been exhibited in galleries internationally. Aaron was constructed through a laborious process of "tutoring" by Cohen, himself an accomplished painter, that spanned over a decade. More recently, work has proceeded on constructed artists that function as poets (Kurzweil 1990), music composers (Ames & Domino 1992), and aesthetic agents in virtual worlds (Bates 1992). A literature has also emerged on the computational underpinnings of artistic creativity more generally (e.g., (Boden 1991)).

Aesthetic Judgements

The philosophy of art, which in the Western tradition dates at least from Plato, has never been an area of widespread agreement (see, e.g., (Dickie & Sclafani 1977)). The range of theories regarding the bases of aesthetic value, judgement and criticism is extraordinary, and the debates show no signs of near-term resolution. This presents a problem for AI scientists wishing to produce computational artists: How do we know when we've got one? How do we know if version A is better than version B, or vice versa? Without the ability to answer such questions the science of artist construction cannot proceed, and these questions *seem* to be inseparably linked to the murky issues of aesthetic judgement. The larger AI community is currently involved in a dialogue on methodological issues, standards, and rigor; many are calling for the adoption of experimental methods from more traditional sciences, for the use of standard examples and criteria of assessment, etc. If those of us working on constructed artists cannot judge our systems without first resolving all of the open questions regarding the judgement of artworks, then we will be on shaky methodological ground indeed. Fortunately, it is possible to separate the two kinds of judgement; we describe one approach to doing so below.

The artworks of Cohen's Aaron have been judged by the artworld and by the museum-going public. According to some theories of art this is the best, or even the only, form of assessment by which to judge the quality of a work (Danto 1978). But this sort of judgement has a high price both in terms of human resources and in terms of time. The science of artist construction will proceed quite slowly if each iteration of each system can be assessed only by organizing a public show and by waiting for critical reviews.[2] Of course, Cohen himself also served as a critic of Aaron's performance, and he was presumably able to apply the results of his judgements to the improvement of the program

[1]The authors acknowledge the support of the Dorothy and Jerome Lemelson National Program in Invention, Innovation, and Creativity.

[2]An experiment combining public assessment with genetic techniques similar to those described later in this paper is currently in progress via mosaic on the internet. The address is: http://porsche.boltz.cs.cmu.edu:8001/htbin/mjwgenform.

in a reasonable amount of time. But it is not clear how these interactions can form the basis of a general theory of aesthetic judgement sufficient to ground a science of artist construction. At best they are instructive for other artists with an interest in applying their own critical faculties to the construction of new artists.

Another approach to this dilemma is to work in a genre with codified, formalized valuation criteria. This has been a popular approach in computer music, as rule-systems have been developed for many forms of music (e.g., (Ebcioglu 1992, Maxwell 1992)). There are three problems with this approach. The first is that existing formalizations are often of "dead" forms—it may be that we understand them well enough to codify them only because they have fossilized. If we want our constructed artists to produce creative works in live genres, such formalizations are of little value. The second problem is that it is not clear that adherence to the rules of a particular art form is a good indicator of aesthetic value; it might merely indicate inclusion in the genre, which might be compatible with aesthetic mediocrity. Third, it is not clear that work in genres with codified valuation criteria will generalize to other genres, many of which seem to resist the imposition of criteria upon which the art world can consense.

The alternative approach that we propose is to factor aesthetic judgement out of the systems that we develop. We don't need to know what the "right" criteria are for aesthetic judgement; we only need to know that our systems are capable of conforming to the range of such criteria that might be proposed. If we develop systems that take critical criteria as parameters, and if our systems work over a wide range of variation of these parameters, then we can safely ignore debates about which critical criteria are correct. We can then ask all opposing parties to submit sets of critical criteria; although they must all be formalizable, they may vary considerably. To the extent that we can keep everyone happy, by producing constructed artists to earn accolades from any formal critic, we will be making real progress in the science of artist construction.

Instances of the framework that we present below produce an artist as output when given a critic (and other data) as input. The constructed artist may not be able to adapt to *other* critics that it encounters later in its career; such adaptation is a subject for future work.

An Artist's Culture

Every artwork exists within a rich cultural and historical context, and many theorists have argued that good art can be neither produced nor assessed in ignorance of this context. It is not obvious, to say the least, how a deep appreciation of the human cultural context can be programmed into a constructed artist. Trurl, the robot who builds an electronic bard in a humorous story by Stanislaw Lem, is forced to repeat within the machine "the entire Universe from the beginning—or at least a good piece of it." (Lem 1974) In most real systems to date, features of the cultural context are implicit in analytical and generative rules, but there is no direct way to vary the culture experimentally.

We believe that the best approach for providing a cultural context for a constructed artist is to make a large case-base of prior works available as a library. In essence, we want to "factor out" the culture in the same way that we "factor out" the critic; by developing systems that take "cultures" as parameters, our systems will be culture-independent and we will be able to assess the success of our systems across cultures. The success of such systems should not depend on any *specific* cultural context; they should be sufficiently flexible to perform within a variety of cultures. The cultural case-base should be made available both to the constructed artist and to the critics that guide the artist construction process.

It may be argued that there is much more to a culture than a library of past works. We agree, but we also believe that a large case-base of successful artworks forms a good basis for cultural sensitivity.[3] Enhanced notions of culture may be incorporated into the framework, so long as all culture-dependent elements are provided as variable parameters to the artist construction system.

Genetic Programming

The framework sketched above calls for an artist construction system that takes as input a set of critical criteria and a case-base of past artworks. The system should produce as output a successful constructed artist—that is, a program that can be executed to produce successful new artworks relative to the given critic and culture.

The technology of *genetic programming* (Koza 1992) provides tools that make the implementation of this framework fairly straightforward. 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.[4] The technique is a variant of the *genetic algorithms* of (Holland 1992). The traditional genetic algorithm evolves fixed-length chromosome strings that encode behavior-producing systems, while genetic programming evolves behavior-producing computer programs directly. 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. For the production of constructed artists we can provide critical criteria as parameters to the system in the form of fitness functions.

A genetic programming system works with a problem-specific *function set* and *terminal set.* These sets contain the primitive elements out of which all of the output programs will be constructed. The genetic programming process starts by creating a large initial population of programs that are random combinations of elements from the function and terminal sets. One generally ensures that each function can

[3]Other case-based approaches to creative processes are presented in (Dartnall & Kim 1993).

[4]Other uses have been made of evolutionary methods in computational arts. See, e.g., (Todd & Latham 1992).

take, in any of its argument positions, any terminal and any value that might be returned by any function in the function set. This allows the use of a simple random function generator, since every combination of functions and terminals can be guaranteed to execute without signalling an error.

Each of the programs in the initial population is assessed for fitness. This is usually accomplished by running each program on a collection of inputs called fitness cases, and by a running a fitness function on the output of each of these runs; the resulting values are then combined to produce a single fitness value for the program.

The fitness values are used in producing the next generation of programs. The next generation may be produced from the current generation via a variety of *genetic operations* including reproduction, crossover, mutation, permutation, and others. We use only reproduction and crossover in the present project; (Koza 1992) describes a variety of genetic operations in detail. The reproduction operator selects a highly fit individual and copies it into the next generation; this is the most direct way to implement the notion of "survival of the fittest." Individuals are selected for reproduction randomly, but the selection function is biased toward highly fit programs.

Fitness-proportionate reproduction does not introduce any new individuals to the system—it merely propagates fit individuals from one generation to the next. The crossover operation, on the other hand, introduces variation by selecting two highly fit *parents*; it generates from them two *offspring,* which are produced by swapping random fragments of the parents. The resulting programs are copied to the next generation.

Over many generations of fitness assessment, reproduction and crossover, the average fitness of the population will tend to improve, as will 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.

Genetic programming searches the space of computer programs in an attempt to maximize fitness. It is fitness that determines the structure of the resulting programs, not the intuitions of a human programmer or algorithm designer. Koza presents applications of genetic programming to a wide range of problems, along with arguments to support its utility as a general automatic programming technique (Koza 1992).

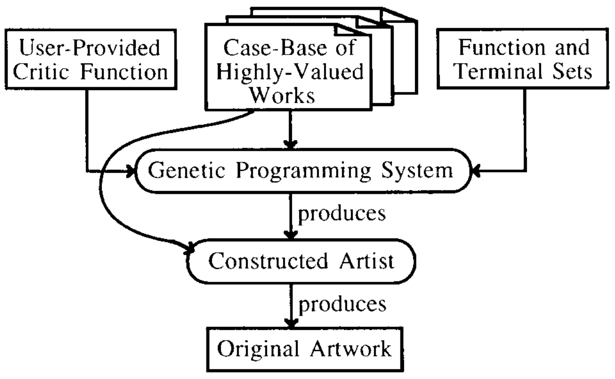

User-Provided Critic Function
Case-Base of Highly-Valued Works
Function and Terminal Sets
Genetic Programming System
produces
Constructed Artist
produces
Original Artwork

Figure 1. Diagram of the Genetic Artist Construction framework

Genetic Programming of Constructed Artists

Genetic programming provides an obvious method for building an artist construction system that takes critical criteria as input: We use an off-the-shelf genetic programming system for which we have crafted function and terminal sets adequate for the production of a wide range of artist programs within some given medium. We then allow the user to write a critic function that will be used as a fitness function by the genetic programming system.

Note that we have great freedom in designing the function and terminal sets. We may use any artwork-producing functions and terminals that we feel are appropriate for the given medium. In particular, we may include functions that access a case-base of prior, highly valued works. The case-base may contain works from the real history of art in the given medium, the results of prior runs of genetic programming, or any mixture of the two. Access to the case-base allows the functions in the function set to produce a range of results depending on the artist's cultural context. The case-base access functions should be made available to the critic functions as well, since many critical criteria may be best phrased in terms of comparisons to works in the prevailing culture.

Figure 1 shows a diagram of the resulting framework for the genetic programming of culturally-contextualized, critic-sensitive constructed artists. The arrow from the case-base to the constructed artist reflects the fact that a constructed artist is a program that may itself take input. This input might come from anywhere; it might, for example, come from a random number generator or from real-time interaction with an audience. In our current work we provide our constructed artists with input from the case-base; that is, the constructed artist takes a prior work from the case-base as input, and produces a new work as output.

Genetic Programming of a Bebop Musician

We illustrate the framework with a system that creates simple programs that produce Bebop jazz melodies. Jazz melody is a good medium for this sort of experimentation for several reasons. First, there are several simple ways to represent melodies in a form that is manipulable by simple programming constructs. Second, the jazz tradition includes several "call and response" forms, so the idea of producing a new work on the basis of an old work has established precedents within the genre. Third, the jazz literature contains several analytical works that enumerate critical criteria (e.g., (Coker 1964)), along with many works on technique that provide guidance in creating a function set (e.g., (Baker 1988)).

We decided to generate programs that produce four-measure melodies as output when given four-measure melodies as input. This corresponds to the popular practice of "trading four" in jazz improvisation. We used a weak representation for melodies: lists of 64 numbers, each of which represents a pitch that will be sounded for the duration of a sixteenth note. Rests are represented as -1, and equivalent adjacent pitches are merged into notes of longer duration. This representation is inadequate because it can accommodate neither thirty-second notes nor triplets of any kind, and because adjacent notes of equivalent pitch must be separated by a rest in order to sound individually. It is nonetheless sufficient for many simple melodies, and it has the advantages of simplicity and ease of manipulation.

We used Koza's LISP-based genetic programming code, which is presented in an appendix to (Koza 1992) and is available on the internet by anonymous FTP.

Our function set, inspired by a list of techniques in (Baker 1988), consists of the following 13 functions: **REP** takes a single melody and returns a new melody that consists of the first measure of the given melody repeated four times. **8VA** takes a single melody and returns it with every note transposed up an octave; notes that are transposed out of the two-octave range above middle C are wrapped to the bottom of the range. **IVA** is similar to **8VA**, but the transposition interval is determined by matching the given melody against the melodies in the case-base. **IVA** transposes the given melody by the average interval between itself and the most similar melody found in the knowledge base. Similarity is determined by computing the inter-note intervals for the pair of melodies to be compared, and by counting the number of times that three-interval sequences occur in both sequences. **EXTEND** takes a single melody and fills any trailing rests with the melody itself. If given a very short melody **EXTEND** may produce a melody with a large number of repetitions. **TRUNC** takes a single melody and replaces all notes following the last rest with additional rests. **DIMINUTE** takes a single melody and speeds it up. It removes every odd-numbered element of the melody list, compressing the remaining elements into the first half of the list and padding the end with rests. **AUGMENT** takes a single melody and slows it down, doubling each element in the first half of the melody, and discarding the entire second half. **FRAGMENT** takes two melodies and returns a melody that has parts taken from each. The returned melody consists of the first two beats of the first given melody, the second two beats of the second given melody, the third two beats of the first given melody, and so on. The **INVERT** function takes a single melody and returns it with each interval inverted. The first note is held constant, the second note differs from the first by the corresponding interval in the given melody *negated,* etc. Again, notes that would be outside of the two octave range above middle C are wrapped around. **RETROGRADE** takes a single melody and returns it reversed. **MOST-FAMILIAR** takes two melodies and returns the one that is most similar to those in the case-base, using the same similarity metric as in **IVA**. **COMPARE-TRANSPOSE** takes a single melody and returns it unevenly transposed, with each note transposed by half the difference between it and the corresponding note in the most similar melody from the case-base. **ROTATE** takes a single melody and returns it moved forward in time by one quarter note, with the last note wrapped around to the beginning.

Our terminal set consists of a single symbol, **CALL-MELODY**, which serves as the input to the programs produced by the system. One runs the resulting program by setting the variable **CALL-MELODY** to some input melody, and then evaluating the program in a LISP listener.

We ran our system with a case-base consisting of five four-measure fragments from Charlie Parker songs. We assessed the fitness of each program by running it with each of the melodies in the case-base as input. Each run produced a single melody that was assessed on the basis of a set of critical criteria inspired by those presented in (Baker 1988). **TONAL-NOVELTY-BALANCE** returns 0 if there is perfect balance between novel tonal material and tonal material that can be found in the case-base. It returns 1 if there is no balance, and intermediate values for intermediate levels of tonal novelty. Matching is performed with 3-note subsequences of the melodies. **RHYTHMIC-NOVELTY-BALANCE** is identical except that the rhythmic structure of the melody, rather than the tonal structure, is compared against the melodies in the case-base. **TONAL-RESPONSE-BALANCE** compares the melody produced by the program with the melody that was provided as input to the program (**CALL-MELODY**). It compares the two melodies point-for-point and returns 0 for a perfect balance of equality and inequality, 1 for complete mismatch *or* exact equivalence, and intermediate values for intermediate degrees of match. **SKIP-BALANCE** returns 0 if the melody perfectly balances diatonic movement (intervals of less than 3) with "skips" (intervals of size 3 or greater). **RHYTHMIC-COHERENCE** returns 0 as long as the melody contains no single sixteenth notes occurring between longer notes. If isolated sixteenth notes do occur in the melody, **RHYTHMIC-COHERENCE** returns the number such notes.

Four of these five critical functions return real numbers between 0 and 1, with lower numbers indicating better melodies. The last critical function returns 0 for a melody that meets an important constraint, and 1 or greater for melodies that don't. The fitness of a melody-producing program is calculated as the sum of the values returned by the critical functions, summed over all of the fitness cases. Assuming for the moment that **RHYTHMIC-COHERENCE** returns no greater than 1, the maximum (worst) fitness value is the number of critical criteria (5) times the number of fitness cases (5), or 25. The best programs will have fitness values considerably closer to 0. Since **RHYTHMIC-COHERENCE** may return greater than 1, it is possible to get fitness values higher than 25, but we have rarely seen such values in practice.

Results

We ran the genetic programming system with a population size of 250 for 21 generations. The best program from the initial, randomly-created population had a fitness of 7.43. The program was: **(FRAGMENT (AUGMENT CALL-MELODY) CALL-MELODY).** This simply interleaves, in two-beat-long sections, the input melody with a slowed down version of the input melody. Since many of the critic functions look for balance, and since the input melody is taken from the case base, this simple program actually performs very well.

As shown in Figure 2, the average fitness of the population improved over the next few generations, but the fitness of the best-of-generation program did not improve noticeably until generation 3, when the following was produced:

```
(FRAGMENT
  (COMPARE-TRANSPOSE
    (INVERT (COMPARE-TRANSPOSE CALL-MELODY)))
  CALL-MELODY)
```

This function performs a more complex manipulation of its input, including two calls to the case-sensitive **COMPARE-TRANSPOSE** function. As shown in Figure 2, the fitness of the best-of-generation program, along with the average fitness of the population, continued to improve through subsequent generations.

The best-of-run program for this run was found on generation 19 and had a fitness measure of 2.82. It was:

```
(FRAGMENT
  (COMPARE-TRANSPOSE (8VA (COMPARE-TRANSPOSE
    (FRAGMENT
      (IVA (DIMINUTE (EXTEND CALL-MELODY)))
      (FRAGMENT
        (EXTEND CALL-MELODY)
        (AUGMENT (RETROGRADE (RETROGRADE
          (ROTATE (FRAGMENT CALL-MELODY
                            CALL-MELODY)))))))))
  (MOST-FAMILIAR (INVERT CALL-MELODY)
                 (IVA CALL-MELODY)))
```

Figure 3 shows a call/response pair in music notation. This response pleases our critic very well—the sum of fitness components is 0.19, which is quite close to a perfect score of 0. This should multiplied by 5, producing 0.95, for comparison to the above-mentioned fitness values. (Recall that the above fitness values were summed over 5 fitness cases.) The sum-of-components values for the best-of-run program run on the 5 fitness cases were 0.19, 0.31, 0.65, 0.41, and 1.25. Although the response in Figure 3 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. The quality of the output vis-à-vis our aesthetic judgement is largely separable from the ability of the system to produce critic-pleasing programs. The former is a question to be argued in the philosophy of art; the latter is an element of the science of artist construction.

Our example system *does* have its weaknesses when assessed purely as a critic-pleaser. The best-of-run program pleases the critic when run on melodies that were used in the fitness cases, but it is not as *robust* as we would like. We ran the program on 3 Charlie Parker melodies that were not used in the fitness cases and produced sum-of-components values 0.81, 1.66, and 0.93. These are not terrible; in fact, two of these values are better than the worst sum-of-components value for a melody used as a fitness

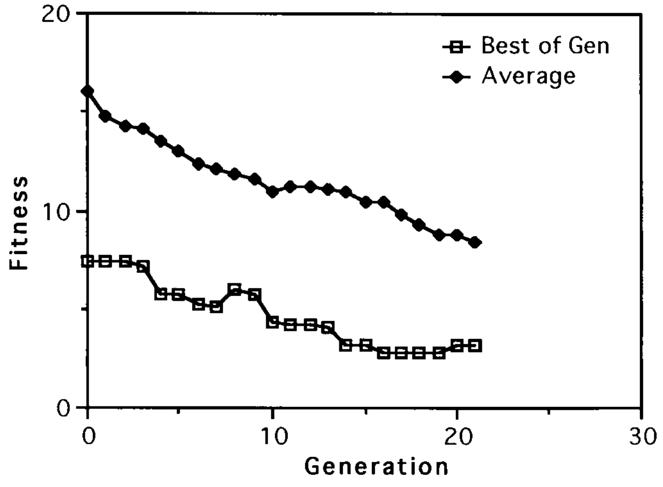


Figure 2. Best-of-generation and average fitnesses

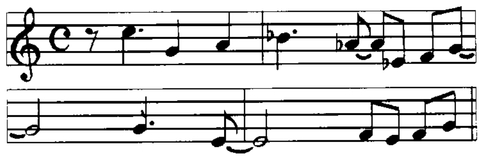
case. But on average the program performs better with input from the fitness cases—it can not yet be said to please our critic in responding to bebop melodies generally.

The lack of robustness is a weakness of our application of the framework to music, and we are exploring it experimentally. We are working with alternative music representation schemes, alternative function and terminal sets, and variations in other parameters of the genetic programming system, in an attempt to produce more robust constructed musicians. We must note, however, that variations in the critic and in the case-base must be explored as well. Although we would like our system to work well independently of changes in these parameters, they have an impact on the ability of the system to produce robust critic-pleasers.

The case for the separability of critical criteria, culture, and techniques for artist construction has been stated strongly in this paper. In fact, the character of a fitness function helps to determine the "fitness landscape" (Kinnear 1994) that is searched by genetic programming. Hence the choice of critic and the composition of the case-base will both have an impact on the effectiveness of the artist construction framework that we have described. For this reason we must work to develop systems that perform well over ranges of critical criteria that might be proposed. To the extent that these ranges depend on our interpretation of the philosophical discussions of aesthetic judgement, the clean separation that we would like to maintain between such discussions and the science of artist construction breaks down. We believe, however, that reasonable generalizations can be made in this area, enabling us to work on artist construction systems with clear, quantitative indicators of success. This belief can only be explored experimentally, by continuing to apply the framework to the construction of artists in various media, by working with various sets of critical criteria that we find in the literature, by providing our systems with access to various cultural contexts, and by assessing the robustness of the art-making programs that result.

The resulting research program presents several challenges. First there are issues of representation; these are

Yardbird Suite by Charlie Parker



Response generated by the constructed musician

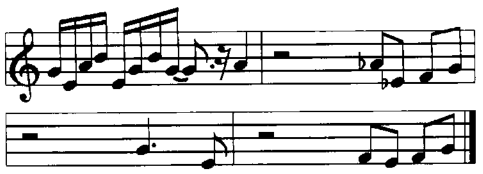


Figure 3. A call/response pair.

problematic even for music, and more so for other media. Then there are issues of scale; our example system uses a tiny case-base and simple critical criteria. While these suffice to demonstrate the framework, we cannot expect to be impressed with the output of systems built on such impoverished notions of culture and criticism. Finally, although our framework frees us from reliance on any *particular* critical criteria, it does require that critical criteria be encoded; some may question the feasibility of this enterprise. We believe that criteria can be extracted from the critical literature, and we are also investigating the automatic generation of critics from the case-base.

Conclusions

Johnson-Laird, in a computational study of jazz improvisation, notes that "neo-Darwinian" theories of creativity have long been espoused, but he rejects them because "their gross inefficiency renders them highly implausible as an account of any sort of mental process." (Johnson-Laird 1991, p.321) The new technologies of genetic algorithms and genetic programming offer the promise of tractable evolutionary processing, and hence theories of creativity-through-evolution may now be explored experimentally. The genetic programming framework for artist construction offers additional advantages in that it provides a relatively clean way to separate out issues of aesthetic judgement from issues of system judgement. Instances of our framework take critics and cultural contexts as parameters, producing constructed artists as output. This allows us to consider the ability of our system to please critics within cultures, without involving us in questions of aesthetics. The separation between the two forms of judgement is not quite as clean as we would like, but nobody said it would be easy to raise an artist.

Acknowledgments

Valuable feedback was provided by Rebecca S. Neimark, Joe Futrelle, and the members of the Propositional Attitudes Task Force at Smith College.

References

Ames, C.; and Domino, M. 1992. Cybernetic Composer: An Overview. In *Understanding Music with AI*, Balaban, M.; Ebcioglu, K.; and Laske, O., eds. 187–205. Cambridge MA: The AAAI Press/The MIT Press.

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

Balaban, M.; Ebcioglu, K.; and Laske, O., eds. 1992. *Understanding Music with AI*. Cambridge MA: The AAAI Press/The MIT Press.

Bates, J. 1992. Virtual Reality, Art, and Entertainment. *Presence* 1: 133–138.

Boden, M.A. 1991. *The Creative Mind: Myths & Mechanisms*. Basic Books (Harper Collins Publishers).

Coker, J. 1964. *Improvising Jazz*. New York: Simon and Schuster, Inc.

Danto, A. 1978. The Artworld. In *Philosophy Looks at the Arts*, Margolis, J., ed. 132–144. Philadelphia, PA: Temple University Press.

Dartnall, T.; Kim, S., eds. 1993. *AI and Creativity*, Working Notes, Spring Symposium. AAAI Technical Report.

Dickie, G.; and Sclafani, R.J., eds. 1977. *Aesthetics*. New York: St. Martin's Press.

Ebcioglu, K. 1992. An Expert System for Harmonizing Chorales in the Style of J. S. Bach. In *Understanding Music with AI*, Balaban, M.; Ebcioglu, K.; and Laske, O., eds. 295–333. Cambridge MA: The AAAI Press/The MIT Press.

Holland, J.H. 1992. *Adaptation in Natural and Artificial Systems*. Cambridge, MA: The MIT Press.

Johnson-Laird, P.N. 1991. Jazz Improvisation: A Theory at the Computational Level. In *Representing Musical Structure*, Howell, P.; West, R.; and Cross, I., eds. 291–325. New York: Academic Press.

Kinnear, K.E. Jr. 1994. Fitness Landscapes and Difficulty in Genetic Programming. In *Proceedings of EC94, The IEEE Conference on Evolutionary Computation*, IEEE.

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

Kurzweil, R. 1990. *The Age of Intelligent Machines*. Cambridge, MA: The MIT Press.

Lem, S. 1974. *The Cyberiad*. New York: Harcourt Brace Jovanovich, Publishers.

Maxwell, H.J. 1992. An Expert System for Harmonizing Analysis of Tonal Music. In *Understanding Music with AI*, Balaban, M.; Ebcioglu, K.; and Laske, O., eds. 335–353. Cambridge MA: The AAAI Press/The MIT Press.

McCorduck, P. 1991. *Aaron's Code: Meta-art, Artificial Intelligence and the Work of Harold Cohen*. New York : W. H. Freeman and Company.

Rowe, R. 1993. *Interactive Music Systems*. Cambridge, MA: The MIT Press.

Todd, S.; and Latham, W. 1992. *Evolutionary Art and Computers*. Academic Press.