TECHNICAL ARTICLE

The Evolution of Musical Organisms

Bruno Degazio

ABSTRACT

The interactive application of genetic algorithms (GAs) can be used to search through the vast control spaces of computer-based musical composition in a highly efficient manner, allowing coordination of interacting parameters to be handled automatically. Even for control paradigms that are well understood, the genetic algorithm can efficiently search out settings that the composer would have probably not have found. The author has developed a software program, the Musical Organism Evolver (MOE), which employs a GA to produce "musical organisms." MOE is written in MIDIFORTH, the author's computer-assisted composition software, and employs functions he developed in previous research. The first and second phases of the software development have been completed and are described here. Future work will extend the GA searching technique to abstract, subjective musical concepts such as "density" and "smoothness."

GENETIC ALGORITHMS

A genetic algorithm (GA) is a computer procedure that searches for an answer to a problem by simulating the cellular reproduction of living organisms. GAs have been shown to solve types of problems for which conventional search techniques are not suitable. Andrew Horner and David Goldberg give an excellent summary of the operation of the standard genetic algorithm:

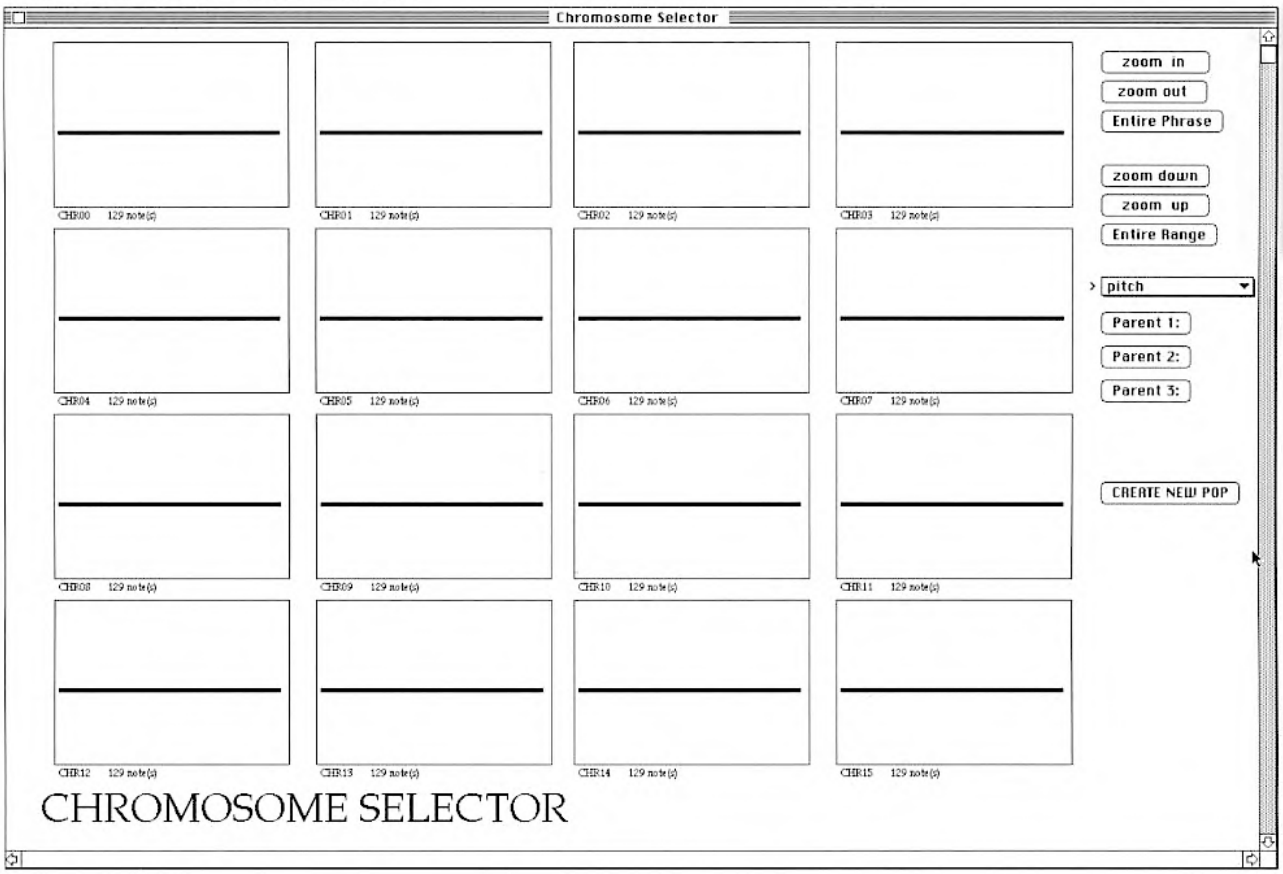
> Genetic algorithms are a randomised optimisation technique which searches the problem space using populations of candidate solutions. . . . The procedure begins with a random initial population of potential solutions, and genetic operators are applied to produce new populations over successive generations. The best candidate solution seen over all generations is then chosen as the answer to the problem. No guarantees are made as to the optimality of this answer, but generally it is a very good solution.
>
> The standard 3-operator GA uses selection, crossover and mutation. These genetic operators work together to efficiently explore the solution space by propagating and recombining the best individuals in each generation [1].

As stated here, three processes of cellular reproduction are important to the operation of a genetic algorithm: selection, crossover and mutation. These processes are simulated by software operations on data structures that themselves simulate the gene and chromosome structures of living organisms.

A GA attempts to find a solution to a problem by picking the

Bruno Degazio (composer), Artificial Evolution Studio, 545 Concord Avenue, Toronto, M6H 2R2 Canada. Email: <degazio@mail.north.net>.

Fig. 1. Screen shot from the author's genetic-algorithm composition system, the Musical Organism Evolver (MOE) (1997). The Chromosome Selector with 16 "blank" chromosomes.



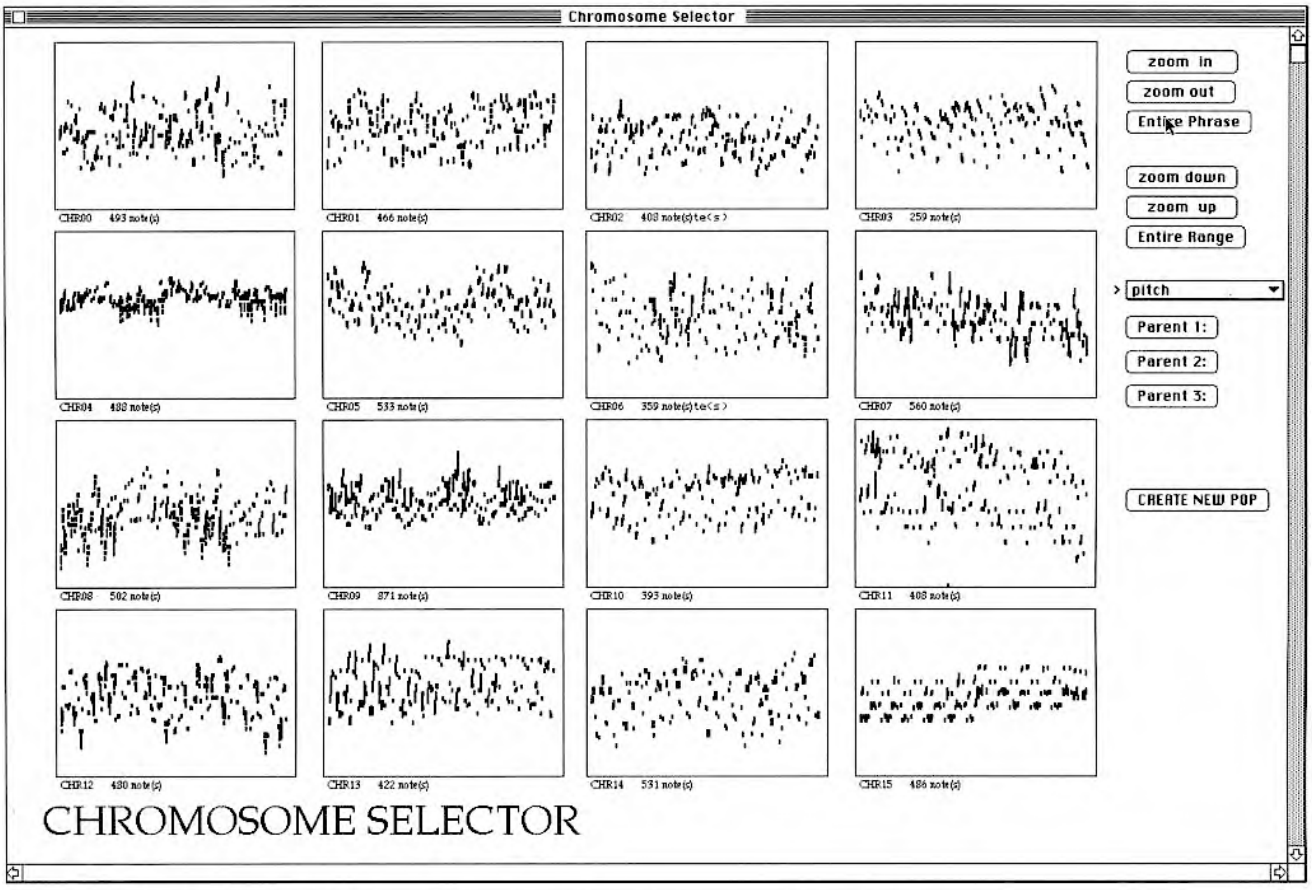


Fig. 2. Screen shot from MOE. An initial population of musical organisms generated at random.

best from a small number of initial solutions. The GA creates this initial "population" of solutions at random. Choosing which members of the population to use is called "selection" and corresponds to "natural selection" or "the survival of the fittest" in the natural world. For some problems, the best solution can be evaluated easily; a software routine known as a fitness function can be written to perform this task. Software applications such as the one described in this paper assume that the "fitness" of a musical "organism" cannot be evaluated entirely by the computer. The fitness function is therefore replaced by the user's personal choice, along with some assistance from the computer in the form of statistical calculation.

The best solutions found in a group (typically the top 5–10%) are "bred" through an operation known as crossover, which simulates reproduction, or the re-combination of DNA. Chosen solutions are paired off and their defining data strings (chromosomes) are split in two, exchanged and rejoined. From every pair of solutions, two new solutions therefore arise. Crossover is the primary source of new solutions and the engine of optimization.

In addition to crossover, mutation is implemented by the random alteration of individual bits in the chromosome string. This ensures that fresh material occasionally enters the gene pool, but the amount is kept to a minimum—typically, there is only one bit of new material introduced per several thousand. Mutation does not have a significant effect on the overall direction of optimization, which is directed primarily by crossover. It serves to inject fresh genetic material into the population, which is then directed toward optimality though crossover.

There are a number of points to bear in mind regarding GAs:

- GAs are *not* a random search. Though beginning at a random origin, the optimization search engine is highly directed.
- GAs are very *general*, because all knowledge of the specific problem to be solved is "hidden" in the fitness function.
- GAs avoid a serious shortcoming of other search strategies: by evaluating many potential solutions in parallel, they avoid becoming stuck on local maxima.

The optimization techniques underlying GAs rest on a solid theoretical foundation (the schema theorem) formulated by John Holland [2]. David Goldberg, a prominent researcher in the field, has some interesting thoughts on the relation of genetic algorithms to creativity:

> What is it we are doing when we are being innovative or creative? Often we take a set of solution features that worked well in one context and a set of solution features that worked well in a different context and bring them together—possibly for the first time—to try to solve the problem at hand. This emphasis and juxtaposition of human creativity is similar to the selection and re-combination of genetic algorithms [3].

Thus, in a limited and mechanical fashion, genetic algorithms provide a means of automating creativity. Or, to put it in a more human-centered way, they may help us to understand our own creativity.

In 1989, Karl Sims of Thinking Machines in Boston used the 65,536-processor Connection Machine to "grow" images that had the complexity of natural

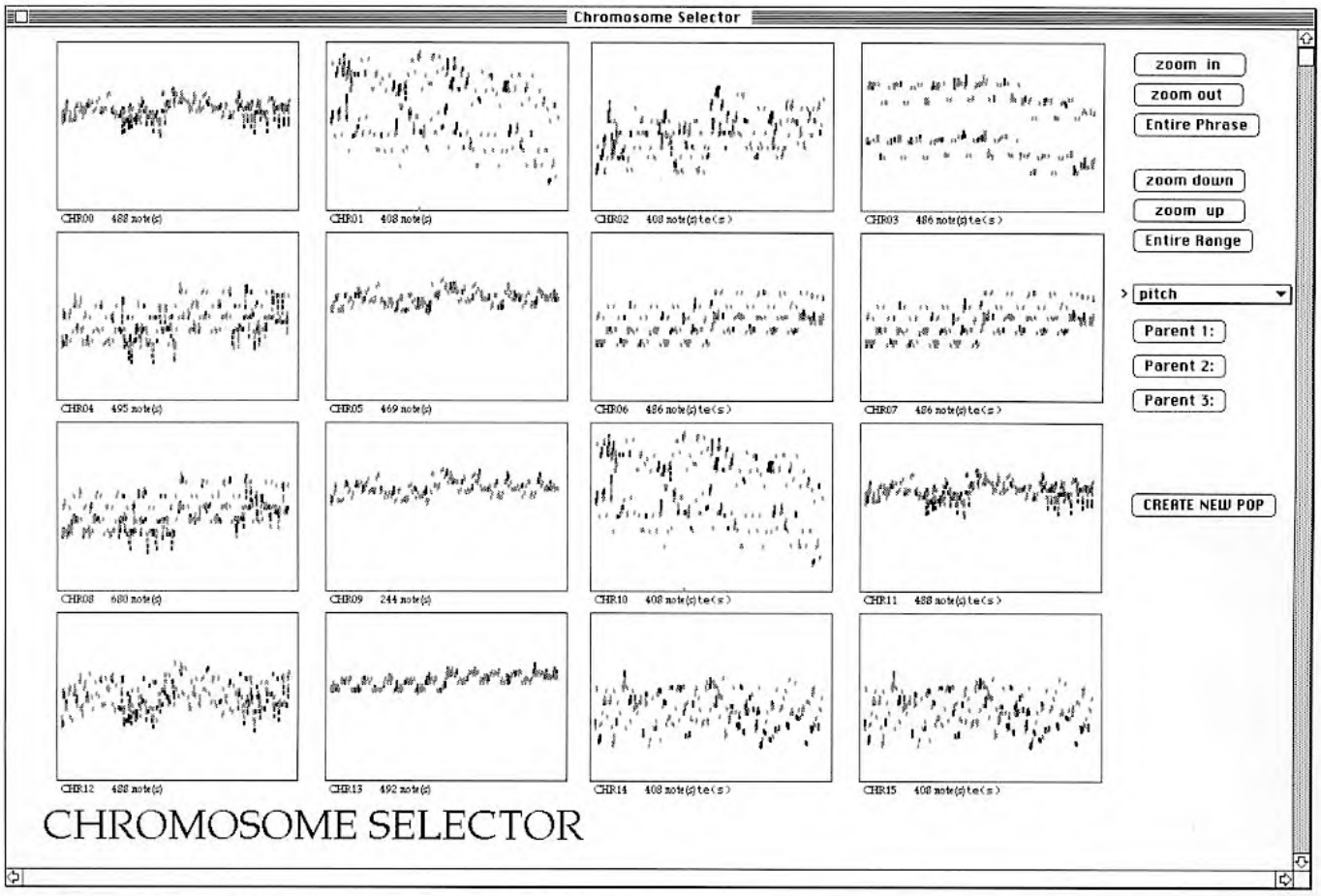


Fig. 3. Screen shot from MOE showing the offspring of the population of musical organisms shown in Fig. 2 after one generation of interbreeding.

objects [4]. In *Artificial Life,* Steven Levy described Sims's efforts:

> Sims amplified the power of Dawkins' methodology by merging it with Holland's genetic algorithm techniques. . . . This arsenal of biologically inspired tools allowed Sims to search through the space of possible artificial trees—a near-infinite range—with blazing speed. . . . This winnowing process was a kind of super-amplified version of the game Twenty Questions, in which a series of true/false queries narrowed the space of all possible humans to a single individual. . . . By simply evolving in a direction that felt right, the designer eliminated the vast number of unsuitable trees and very quickly got to a desirable forest [5].

Sims included as genes a number of ad hoc factors that derived from graphics functions available on his supercomputer. These included parameters such as *branching factor, growth rate, twistiness* and *budding behavior.*

In Sims's system, the fitness-function evaluation of a standard genetic algorithm is replaced by the "unnatural selection" of the user—the user simply chooses whichever descendant he or she prefers. I use a similar selection procedure in Phase Three of Musical Organism Evolver (MOE) (discussed below), a GA software system that I am developing.

Previous Work in the Field

To date, genetic algorithms have not been used heavily in musical applications. Tapio Takala, James Hahn, Larry Gritz, Joe Geigel and Jong Won Lee briefly describe the use of a GA to search through "timbre trees" for desirable sounds in a hybrid physical-model/additive-synthesis system [6]. Their system employs selection by hand (or ear) rather than through automated evaluation of a fitness function. Andrew Horner, James Beauchamp and Norman Packard present a similar application that they call "timbre breeding" [7]. Additive synthesis is the paradigm under control in their system. Jarkko Vuori and Vesa Valimaki also discuss the application of a GA to determine parameters for a physical-modeling synthesizer (parameters for this kind of synthesizer are notoriously difficult to control) [8].

Horner and Goldberg describe the use of a GA to generate melodies that evolve from a given origin melody to a specified destination melody [9]. They deliberately limit the operation set to a small number of very simple musical operations (i.e. delete a note, mutate a note, rotate a note). The GA then finds the series of these operations that most closely effects the transformation of the origin melody into the destination melody. This transformation process seems to have been chosen both for its intrinsic musical interest and because the fitness of the individuals is easy to evaluate: each one is simply compared note for note against the melodic goal.

Except for this use by Horner and Goldberg, all of the uses mentioned above are typical applications of GAs to parameter search and optimization. My project differs fundamentally from these in that it applies GAs to much higher-level musical structures. Horner and Goldberg's use does deal with musical structures per se, but only in a very limited fashion. Of recent interest in this regard is John Biles's program GenJam, which applies genetic-algorithm techniques to the generation of music in a conventional jazz style [10].

SOFTWARE ARCHITECTURE

I developed MOE in order to investigate compositional applications of GAs. This software is written in the programming language Forth using musical data operations derived from MIDIFORTH, a music software system of my own making, which has itself been evolving for several years. MIDIFORTH began life as a set of Forth language extensions that I produced to experiment with musical applications of chaotic and recursive processes [11]. It has since accumulated a large number of musical data operators employing these and other processes [12]. Of particular importance to the MOE software are the many operators based on Heinrich Schenker's theories of contrapuntal prolongation [13].

MOE consists of two principal components: the evolver and the renderer.

The evolver carries out the processes of chromosome pairing, gene crossover and mutation, implementing the essential features of a GA process. Its "front end," or user interface, is the chromosome selector (Fig. 1), which allows for the interactive selection of a small number of parents (three, in the current version). The graphic view is a conventional pitch-time representation, with time running from left to right in each of the 16 organism graphs and pitch represented as discrete MIDI notes proceeding from low to high along the vertical axis. In future versions of the system, the selector will also present some statistical information about each organism and will perform an application-specific fitness evaluation. Types of evaluation may include

- calculation of the percentage of notes that are "contrapuntally correct," i.e. that obey counterpoint rules vis à vis a given *cantus firmus* (pre-existing melody)
- calculation of the percentage of notes that are "harmonically correct," i.e. that harmonize with a given chord progression
- calculation of the percentage of notes that meet statistical criteria for durations, melodic leaps, pitch intervals and other musical percepts.

The renderer turns a chromosome, which is basically a series of instructions, into a musical data structure. This could be a MIDI file for playback on a standard MIDI synthesizer or a Csound score file for compiling directly to digital audio using MIT's Csound software. The renderer examines every gene in the evolved chromosome and applies a selected process to the degree specified by that gene's content. The renderer demands most of the central-processing-unit resources of the host system. To date, only the MIDI renderer has been implemented in the system described here.

WORKING PROCEDURE

The basic procedure for breeding musical organisms is illustrated with computer screenshots from the current system. First, the software generates a small population of chromosomes at random (Fig. 2). Note that in this initial population, individuals are quite distinct from one another. From this group, the user can select three parents for further breeding. The choice is made both by eye and by ear: the graphic view in the chromosome selector allows for a quick grasp of the organism's musical structural features, and musical details can be auditioned by playing the rendered chromosomes on a MIDI synthesizer simply by clicking on them with the mouse. The software breeds each of these three selected individuals with the remaining organisms at a rate proportional to the selected individual's fitness, i.e. its ranking as parent 1, 2 or 3. It is important to note, however, that even "unfit" individuals breed so that potentially useful genetic information is not lost too early in the evolutionary process. After interbreeding and rendering as MIDI data, the offspring appear (Fig. 3). Features of the selected parents now begin to appear in several offspring, causing the software to group them together as "species" or "families." For example, the individuals numbered 00, 05, 09 and 11 in Fig. 3 form one family of similar (but not identical) individuals clearly deriving from the parent numbered 04 in the preceding generation (Fig. 2). Other families apparent in this graph consist of individuals 01 and 10; 04 and 08; and 02, 14 and 15. The cycle of selection, breeding and rendering is repeated until an individual is generated that is acceptable as a musical composition (or as thematic material on which to base a composition).

IMPLEMENTATION PLAN

In order to manage the complexity of this project, I divided the software development into three phases.

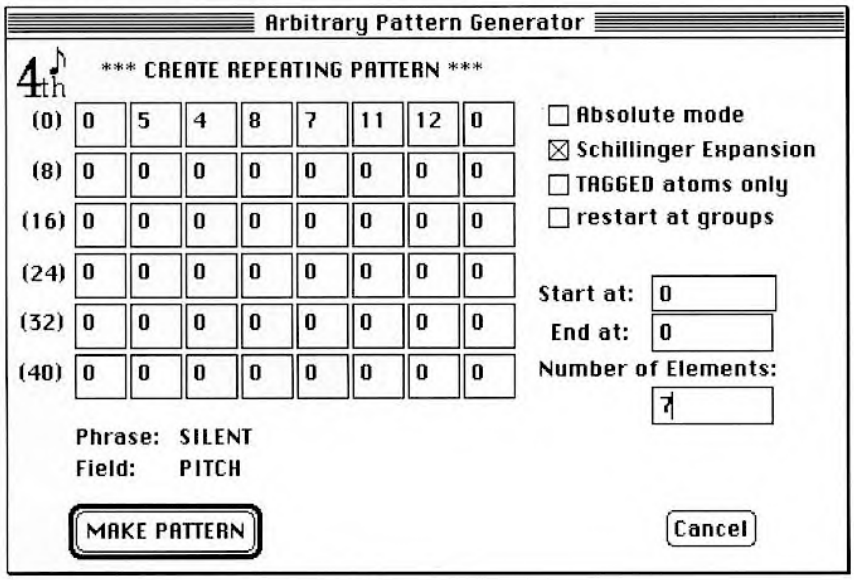
Phase 1

Phase 1 of designing MOE involved the implementation of GAs as control algorithms for existing individual MIDIFORTH processes.

As a simple example, consider the control panel for the MIDIFORTH function called the Arbitrary Pattern Generator (Fig. 4). The dialogue box is dominated by 48 fields of pattern elements. There are also fields representing the number of elements in the pattern and a small number of relevant flags (in the upper right) that control the operating mode of the function. Not shown in the control panel itself is an additional parameter: the MIDI data type with which the function will operate. This is selected from a separate menu.

The 48 element fields are typical MIDI parameters with seven-bit dynamic ranges

Fig. 4. Screen shot from MOE. The Arbitrary Pattern Generator control screen.



(i.e. they can take on values from 0 to 127). These fields comprise the largest part of the control space, taking up 336 bits (48 × 7). The Number_of_Elements field can take on a value of 0 to 48 and therefore requires six bits to define, while the four mode flags require one bit each. The Starting Note and Ending Note parameters can reasonably take on values from 0 to a few tens of thousands, so 16 bits are adequate to define each of them. Finally, the MIDI data type pointer must choose from a list of 22 items and therefore requires five bits. This results in the total control space shown in Table 1.

The number of different settings for this simple function is therefore 2^{585}, which is inconceivably large—far larger than the number of atoms in this universe. Admittedly, most of these settings are useless (as are, for example, all of the settings whose value is 0 for each one of their 48 element fields). However, it is still possible that a GA search through this space will arrive at unique applications of other patterns.

The chromosome for this function consists of a binary string 383 bits long; in it, the individual genes control the parameters listed above. The standard genetic operations of selection, crossover and mutation work on populations of these strings. The net result after a given number of generations is an Arbitrary Pattern Generator setting that produces a unique and desirable result.

I intended this phase of the software to evaluate basic concepts in a fairly straightforward way and to test implementation of the GA process itself.

Phase 2

The second phase of designing MOE involved implementation of GAs as a macro language for strings of MIDIFORTH processes.

GAs used for musical composition do not come into their own until applied to longer sequences of operations. When more operations are involved, the GA becomes a control structure for a macro language controlled through genetic programming techniques [14]. The necessary gene data structure is simple:

```
1 byte:    operation_code
1 byte:    grouping_structure
126 bytes: operation-dependent
           parameter fields
```

The chromosome structure becomes a sequence of these genes rather than a sequence of bits. This sequence is of variable rather than fixed length, which is the mark of a genetic programming application rather than a simple genetic algorithm. The chromosome structure is

gene0(128 bytes), gene1(128 bytes), gene3(128 bytes) . . . gene*N*(128 bytes)

In the gene structure, the operation_code is a single-byte character representing one of MIDIFORTH's many built-in functions. These include many functions common to most MIDI-oriented music-processing programs (transpose, invert, quantize, crescendo/decrescendo, time scale, etc.); functions based on chaos theory and recursive processes (Strange Generator, Recursive Pattern Generator, Mandelbrot Pattern Generator, randomize); functions based on conventional musical theory and practice (modalize, Interval Corrector, arpeggiator, harmonizer, trill); and functions based on Heinrich Schenker's theory of contrapuntal prolongation (neighbor note, escape tone, passing tone, subdividing tone, suspension, cambiata, scale run).

The grouping_structure field requires a few words of explanation. Grouping structure is a concept of temporal structure developed in Fred Lerdahl and Ray Jackendoff's *A Generative Theory of Tonal Music* [15]. Based on theories of formal grouping originally explored by Gestalt psychologists in the 1920s, Lerdahl and Jackendoff's work attempts to explain in a formal fashion our intuitive notions of perceived temporal structures in music. I believe that the imposition of this structure, above and beyond the structure evolved by the genetic algorithm process, is essential to the musical success of this project. The grouping_structure field is a pointer to a predefined list of perceptually relevant temporal groups that may be either simple or compound. A simple group consists of a starting time and ending time. For example, a group may begin on measure 3, beat 1 and end on measure 4, beat 4. This allows the specification of unique temporal regions anywhere throughout the organism. A compound group is a list of simple groups. An example of a compound group representing all the consequent phrases in a piece would be

```
compound_GROUP_A:
  group a: measure 3, beat 1 to measure 4, beat 4
  group b: measure 7, beat 1 to measure 8, beat 4
  group c: measure 11, beat 1 to measure 12, beat 4
  group d: measure 15, beat 1 to measure 16, beat 4
  (etc. to end of piece)
End compound_GROUP_A
```

The 126 bytes of data that follow the grouping structure possess a meaning dependent on the particular operation code. For example, an operation code of 1 indicates the transpose function.

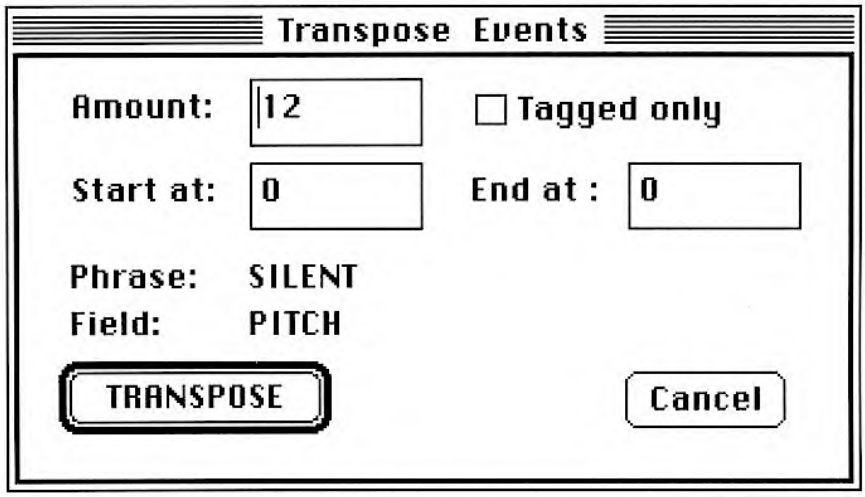


Fig. 5. Screen shot from MIDIFORTH, the author's computer-assisted composition software, showing the Transpose Events control panel.

Table 1. The total control space for Phase 1 of MOE.

Element fields (48 x 7)	336 bits
#_of_elements	6 bits
ABSOLUTE mode	1 bit
Schillinger expansion mode	1 bit
tagged notes only mode	1 bit
restart at groups mode	1 bit
MIDI data type (implied)	5 bits
Starting note	16 bits
Ending note	16 bits
total	383 bits

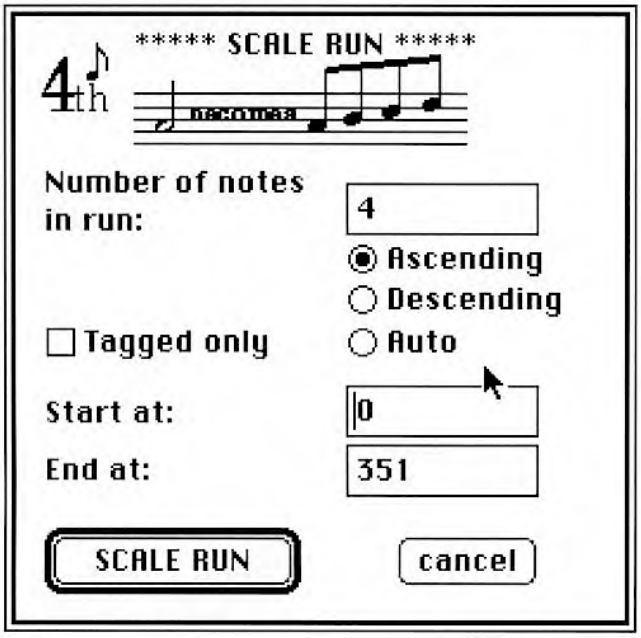


Fig. 6. Screen shot from MIDIFORTH showing the Scale Run ornament control panel.

For this function, the following bytes have the meanings indicated (bytes are numbered beginning with 0—i.e. byte 0 is the operation code and byte 1 is the grouping-structure code).

byte 2:	transposition amount (– 128 to +127)
byte 3:	transpose only selected notes/transpose all notes
bytes 4–127:	undefined

These correspond to the controls available in the standard MIDIFORTH interactive control panel for this function (Fig. 5). Another example is the MIDIFORTH Scale Run function (Fig. 6). The corresponding gene meaning in this case is

byte 2:	number of notes in scale run (1–255)
byte 3:	run only selected notes/ run all notes
byte 4:	direction (up, down, or automatically determined by following note)
byte 5:	modal pattern (one of eight predefined by user)
bytes 6–127:	undefined

Phase 3: Future Work

Phase 3 of MOE development will involve the implementation of processes to control high-level musical parameters such as activity, density and clarity.

In phase three of MOE, the genes of the musical organism will relate to high-level musical abstractions such as

- general musical concepts, including figuration, dissonance, mode (major, minor, artificial), chromaticism, texture (degree of rhythmic continuity) and roughness (degree of harmonic continuity)
- ad hoc parameters derived from existing functions of MIDIFORTH. These include embellishment; activation; contrapuntal "correctness"; tonal continuity (modulation); repetitiveness; and techniques derived from fractal geometry, including melodic recursion (von Koch curves), Mandelbrot mapping and Strange Attractors.
- elements derived from the work of Lerdahl and Jackendoff and from Schenkerian musical analysis involving accentuation, rhythmic structure and grouping structure [16].

With the successful implementation of the preceding phases, combinations of MIDIFORTH functions can be grouped together as meta-operations to directly specify high-level musical or perceptual features. The most intriguing of these come from Joseph Schillinger's *System of Musical Composition*, which, despite its grand title, is more a compendium of odd musico-mathematical tricks and techniques than it is a comprehensive system [17]. It does, however, provide a long list of interesting musical perceptual generalizations. These include tension, periodicity, symmetry, density, continuity, clarity, chroma, saturation, fragmentation, melodic figuration, attack continuity, rhythmic continuity, instrumental continuity, density, group continuity, dynamic continuity and harmonic continuity.

This phase represents the fruition of the musical application of GAs.

APPLICATIONS

The Artificial Evolution Studio is currently engaged in the application of artificial-life techniques to music. In the winter of 1998, the studio will present a concert of music created using MOE. The concert will include new works created for the occasion by a group of invited composers including Gustav Ciamaga, Bruno Degazio, David Keane, Karl Mohr and Gene Martinek.

References and Notes

1. A. Horner and D.E. Goldberg, "Genetic Algorithms and Computer-Assisted Music Composition," *Proceedings of the International Computer Music Conference* (Montreal: ICMA, 1991) p. 480.

2. J.H. Holland, *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: Univ. of Michigan Press, 1975).

3. D. E. Goldberg, "The Existential Pleasures of Genetic Algorithms," IlliGAL Report No. 94010 (Urbana, IL: Univ. of Illinois, Illinois Genetic Algorithm Laboratory, 1990).

4. Karl Sims, "Artificial Evolution for Computer Graphics," SIGGRAPH 91 Proceedings (ACM Computer Graphics) **25**, No. 3, (1991).

5. S. Levy, *Artificial Life: The Quest for a New Creation* (New York: Pantheon, 1992).

6. Tapio Takala, James Hahn, Larry Gritz, Joe Geigel and Jong Won Lee, "Using Physically Based Models and Genetic Algorithms for Functional Composition of Sound Signals, Synchronised to Animated Motion," *Proceedings of the International Computer Music Conference* (Tokyo: ICMA, 1993).

7. Andrew Horner, James Beauchamp and Norman Packard, "Timbre Breeding," *Proceedings of the International Computer Music Conference* [6].

8. Jarkko Vuori and Vesa Valimaki, "Parameter Estimation of Non-Linear Physical Models by Simulated Evolution—Application to the Flute Model," *Proceedings of the International Computer Music Conference* [6].

9. Horner and Goldberg [1].

10. John Biles, "GenJam Populi—Training an IGA through Audience Mediated Performance," *Proceedings of the International Computer Music Conference* (Banff, Canada: ICMA, 1995). See also Biles, "GenJam: A Genetic Algorithm for Generating Jazz Solos," *Proceedings of the International Computer Music Conference* (San Francisco: ICMA, 1994).

11. B. Degazio, "Musical Aspects of Fractal Geometry," *Proceedings of the International Computer Music Conference* (San Francisco: ICMA, 1986).

12. B. Degazio, "Context Sensitive Editing in the MIDIFORTH Computer Music System," *Proceedings of the International Computer Music Conference* (Cologne, Germany: Feedback Studios Verlag, 1988) and "New Software Composition Tools," *Fourth Biennial Arts and Technology Symposium* (New London, CT: The Center for Arts and Technology at Connecticut College, 1993).

13. H. Schenker, *Der Freie Satz* (Free Composition) (Vienna: Universal Edition A.G., 1935; New York and London: Longman, 1979).

14. John Koza, *Genetic Programming* (Cambridge, MA: MIT Press, 1992).

15. Fred Lerdahl and Ray Jackendoff, *A Generative Theory of Tonal Music* (Cambridge, MA: MIT Press, 1983).

16. Lerdahl and Jackendoff [15].

17. Joseph Schillinger, *The Schillinger System of Musical Composition* (New York: Da Capo, 1978; originally published by Carl Fischer Inc. in 1941).

Bibliography

Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley, 1989).

_____ . "Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking," IlliGAL Report No. 90001 (Urbana, IL: Univ. of Illinois, Illinois Genetic Algorithm Laboratory, 1990).

Glossary

additive synthesis—A sound synthesis system in which complex sounds are created by summing together simpler sounds (generally, sine waves).

control space—The virtual space defined by the set of all possible values to a solution.

local maxima—In mathematical problem solving, a solution that is better than any adjacent solution but not necessarily the best possible one overall.

Mandelbrot set—The set of complex numbers defined by operations of the form $X = X^2 + c$, well known because of its application to computer graphics. It was first described in detail by Benoit Mandelbrot in 1980.

recursive process—In MIDIFORTH, any of a number of software processes involving feedback.

schema theorem—A theorem developed by John Holland of the University of Michigan demonstrating that above-average chromosomes are treated preferentially by the GA process.

Strange Attractor—In MIDIFORTH, the result of the recursive process defined by $X = \lambda \times X \times (1 - X)$.

von Koch curve—A fractal object formed by the recursive application of midpoint displacement to a line, first described by H. von Koch in 1904.

Manuscript received 28 May 1996.