

Sheridan College

SOURCE: Sheridan Scholarly Output, Research, and Creative Excellence

Publications and Scholarship

Faculty of Animation, Arts & Design (FAAD)


1996

Evolution of Musical Organisms

Bruno Degazio

Sheridan College, bruno.degazio@sheridancollege.ca

Follow this and additional works at: https://source.sheridancollege.ca/faad_publications

 Part of the [Composition Commons](#), and the [Software Engineering Commons](#)

SOURCE Citation

Degazio, Bruno, "Evolution of Musical Organisms" (1996). *Publications and Scholarship*. 8.
https://source.sheridancollege.ca/faad_publications/8



This work is licensed under a [Creative Commons Attribution-NonCommercial-No Derivative Works 4.0 License](#).

This Conference Proceeding is brought to you for free and open access by the Faculty of Animation, Arts & Design (FAAD) at SOURCE: Sheridan Scholarly Output, Research, and Creative Excellence. It has been accepted for inclusion in Publications and Scholarship by an authorized administrator of SOURCE: Sheridan Scholarly Output, Research, and Creative Excellence. For more information, please contact source@sheridancollege.ca.

EVOLUTION OF MUSICAL ORGANISMS

Bruno Degazio

The Artificial Evolution Studio (artevo@interlog.com)

192 Spadina Ave • Suite 512 • Toronto • Ontario • Canada • M5T 2C2

ABSTRACT: The development of software for musical applications has led to a proliferation of elaborate control paradigms with extremely large parameter spaces. These spaces can be daunting to explore interactively because of their vastness. Furthermore, parameters often interact in ways not made explicit by the control panel, effectively increasing the complexity of the space even further. Application of genetic algorithms (GAs) can be used to search through these vast spaces in a highly efficient manner. Coordinated control of interacting parameters is handled automatically by this system. Even for control paradigms that are well understood, the genetic algorithm can efficiently search out control settings that would have been otherwise unlikely to arise. The author has developed a software system that employs genetic algorithms to evolve 'musical organisms'. The system is built on MidiForth, the author's computer-assisted composition software [Degazio 1988, 1993] and employs many unique functions developed in previous research. This paper describes the second phase of research; future work will extend the GA searching technique to abstract, subjective musical concepts such as *density* and *smoothness*.

1.0 Background

There are a number of points to bear in mind regarding GAs: (1) They are *not* a random search. Though beginning from a random origin, the optimization search engine is highly directed. (2) They are very *general* because all knowledge of the specific problem to be solved is 'hidden' in the fitness function. (3) By evaluating many potential solutions in parallel, they avoid becoming stuck on local maxima. (4) GAs rest on a solid theoretical foundation (the *schema theorem*) supplied by John Holland. In addition, David Goldberg, a prominent current 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. Thus, in a limited and mechanical fashion, genetic algorithms provide a means of automating creativity. Or to put it in a more human centred way: they may allow us to understand *our own* creativity."

In 1989, Karl Sims of Thinking Machines Incorporated used the 65,536 processor Connection Machine to 'grow' images with the complexity of natural objects. Sims included, as genes, a number of *ad hoc* factors that derived from graphics functions available on his supercomputer. These included such parameters as: branching factor, growth rate, twistiness, and budding behaviour. In Sims' system the 'fitness function' evaluation of a standard genetic algorithm is replaced by the 'unnatural selection' of the user - the user simply chose whichever descendant looked best. A similar selection procedure is used by the system under discussion. Likewise, the 'genes' of the musical organism relate to high level musical abstractions: (1) *ad hoc* parameters derived from existing MIDIFORTH functions such as: embellishment, activation, contrapuntal 'correctness', tonal continuity, modulation, and repetitiveness, (2) melodic recursion/von Koch curves, Mandelbrot mapping, and strange attractors.

1.1 Previous Work in the Field

To date, genetic algorithms have not been heavily used in musical applications. [Takala et al 1993] briefly describe the use of a GA to search through 'timbre trees' for desirable sounds in a hybrid physical model/additive synthesis system. This appears to be a classic application of a GA for search and optimization, and also employs selection by hand rather than through automated evaluation of a 'fitness function'. [Horner, Beauchamp, Packard 1993] present a similar application that they call 'timbre breeding'. Additive synthesis is the paradigm under control in their system. [Vuori and Valimaki 1993] likewise discuss the application of a GA to determine physical modelling synthesis parameters. [Horner and Goldberg 1991] describe the use of GA generating melodies that 'evolve' from a specified origin to a specified destination. 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 a series of these operations that turns an originating melody into a 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 - they are simply compared note for note against the melodic goal. Except for [Horner and Goldberg 1991], all of these papers describe applications to audio synthesis, which is a conventional application of GAs to parameter search and optimization. This project differs radically in that it intends to apply GAs to much higher level musical structures. Of recent interest is John Biles' program *GenJam*, which applies genetic algorithm techniques to generation of music in a conventional jazz style [Biles 1994, 1995].

2.0 Software Architecture

The *evolver* itself consists of two components: the *engine* and the *renderer*. The *engine* 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* (figure 1), which allows the interactive selection of a small number of parents, typically three. Chromosomes, rendered as MIDI data, can be viewed graphically, and performed on a MIDI synthesizer. The graphic view is a conventional pitch-time representation, with time running from left to right in each of the sixteen chromosome graphs, and pitch represented as discrete MIDI notes from low to high along the vertical axis. In future versions of the system, the selector will also present some statistical information about each rendered chromosome, and will perform an application specific 'fitness' evaluation. Types of evaluation may include:

- (1) calculation of percentage of notes that are 'contrapuntally correct' (i.e. that obey counterpoint rules *vis a vis* a given *cantus firmus* pre-existing melody)
- (2) calculation of percentage of notes that are 'harmonically correct' (i.e. for a given chord progression)
- (3) calculation of percentage of notes that meet statistical criteria for durations, melodic leaps, etc.

The *renderer* turns an evolved chromosome into a sensible data structure, for example a MIDI file, for playback on a standard MIDI synthesizer. The *renderer* must examine every gene in the evolved chromosome and apply a selected process to the degree specified by that gene's content. In computing terms, the *renderer* demands most of the CPU resources of the host system. In the system described here, a simplified version of the MIDI renderer has been implemented.

3.0 Working Procedure

The basic procedure for breeding musical organisms is illustrated with computer screens from the system. First, a small population of sixteen musical organisms is generated randomly (figure 2), labelled CH00 to CH15. Note that in this initial population individuals are quite distinct from one another. From this group, three parents are selected for further breeding. The choice is made both 'by eye' and 'by ear', since the graphic view in the chromosome selector allows a quick grasp of music structural features, and musical details can be noted by performing the chromosomes on a MIDI synthesizer simply by clicking on them with the mouse. These individuals are bred with the remaining organisms at a rate proportional to their "fitness", according to their ranking as parent 1, 2 or 3. It is important, 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, the offspring appear as in figure 3. Note that features in the selected parents now begin to appear in several offspring, causing them to group as 'species' or 'families'. For example, the individuals numbered 02, 10, 11 and 13 in figure 3 form one family of similar (but not identical) individuals clearly deriving from the parent numbered 15 in the preceding generation. Other families apparent in this graph consist of individuals 00, 04 (but note the interesting inversion in overall shape), 08 and 09 and a third grouping of numbers 03, 05, 06 and 15. This cycle of selection, breeding and rendering is repeated until an individual is generated which is acceptable as a musical composition.

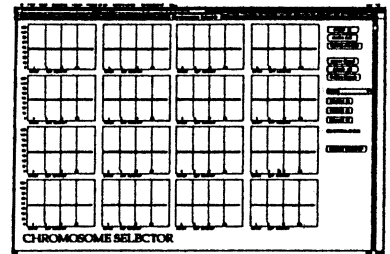


fig. 1 - chromosome selector with 16 'blank' chromosomes

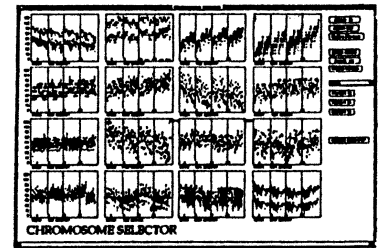


fig. 2 - Initial population of random musical organisms

4.0 Implementation Plan

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

4.1 Phase I: Implementation of GA's as control algorithms for existing individual MIDIFORTH processes. As a relatively simple example, consider the control panel for the MIDIFORTH function called the *arbitrary pattern generator* (figure 4). The dialog box is dominated by the 48 fields for the pattern elements.

There are also fields for the number of elements in the pattern, and for a small number of relevant flags that control the operating 'mode' of the function. Not shown in the control panel itself is an additional parameter -

the MIDI data type on which the function will operate. This is interactively selected from a separate menu. The 48 element fields are typical MIDI parameters and have a seven bit dynamic range (i.e. they can take on values from 0 to 127). These fields comprise the largest part of the control space, taking up $48 \times 7 = 336$ bits.

The *number of elements* field can take on a value of 0 to 48 and therefore requires 6 bits to define, while the five 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 them. Finally, the MIDI data type pointer must choose from a list of 22 items and therefore requires 5 bits.

This results in a total control space of 383 bits. The number of different settings for this simple function is therefore 2^{383} , which is an inconceivably large number. Admittedly, most of these settings are useless (for example, all of the settings with zero throughout the 48 element fields). However, it is still possible that a GA search through this space will arrive at unique applications of patterns.

The chromosome for this function consists of a binary string, 383 bits long, of which the individual 'genes' control the parameters listed above.

The standard genetic operators of crossover, inversion, and mutation work on populations of these strings. The net result after some number of generations is the *arbitrary pattern generator* setting that produced a unique and desirable result.

The functions tested so far include the *arbitrary pattern generator* and the *recursive pattern generator*. The remaining MIDIFORTH functions are currently being implemented.

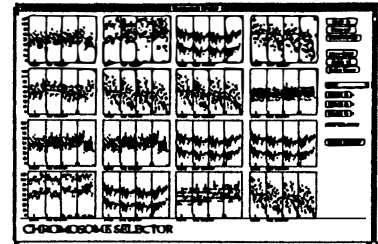


fig. 3 - population after one generation of interbreeding

4.2 Phase II: Implementation of GAs as a macro language for strings of MIDIFORTH processes. While interesting as a test bed for the processes involved, the musical use of GAs does not come into its own until applied to longer sequences of operations. In this sense, the GA becomes a control structure for a macro language, controlled through *genetic programming* techniques [Koza 1992]. The necessary gene data structure is:

1 byte	operation_code,
1 byte	grouping_structure,
126 bytes	operation dependent parameter fields.

The chromosome structure is then a sequence of these genes rather than a sequence of bits. This sequence may be of variable rather than fixed length, another point which marks this as a genetic programming application rather than a simple genetic algorithm. The chromosome structure is then simply:

gene0 (128 bytes), gene1 (128 bytes), gene2 (128 bytes) ... geneN (128 bytes)

In the gene structure, the *operation_code* is a single byte character representing one of MIDIFORTH's many built-in functions, which include: ramp, transpose, compand, randomize, invert, crab, modalize, quantize, scale-time, set to value, correct intervals, activate, harmonize, arbitrary pattern generator, strange generator, recursive pattern generator, Mandelbrot generator, change field, select notes, and ornamentation (20 different types including trill, turn, mordent). The following 126 bytes of data possess a meaning dependent on the particular operation code. For example, an operation code of 1 indicates the transpose function. For this function, the bytes have the following meaning: byte 0 - transposition amount (-128 to +127), byte 1 - transpose only

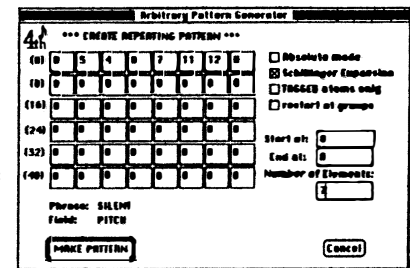


fig. 4 - arbitrary pattern generator control screen

tagged notes, bytes 2 to 127 - undefined. These correspond to the controls available in the standard MIDIFORTH interactive control panel for this function (figure 5).

4.3 Future Work: Implementation of processes to control 'high-level' musical parameters (e.g. activity, density, clarity). With the successful implementation of the preceding phases, combinations of MIDIFORTH functions could 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 the grand title, is more of a compendium of odd musico-mathematical tricks and techniques. He does, however, provide a long list of interesting musical perceptual generalizations. Some of these include:

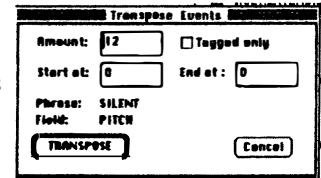


fig. 5 - MIDIFORTH transpose control panel.

<i>tension</i>	<i>periodicity</i>	<i>fragmentation</i>	<i>density group continuity</i>
<i>chroma</i>	<i>clarity</i>	<i>continuity</i>	<i>harmonic continuity</i>
<i>saturation</i>	<i>melodic figuration</i>	<i>attack continuity</i>	<i>rhythmic continuity</i>
<i>symmetry</i>	<i>density</i>	<i>dynamic continuity</i>	<i>instrumental continuity</i>

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

5.0 Applications

The Artificial Evolution Studio is currently engaged in the application of artificial life techniques to music and will be presenting, in the spring of 1997, a concert of music created using this software. 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.

Acknowledgements

This research was carried out with the financial assistance of the Canada Council, Media Arts Section. The Artificial Evolution Studio wishes to thank Karl Mohr for his help in the preparation of this paper.

References

- [Biles, John 1995] *GenJam Populi - Training an IGA through Audience Mediated Performance*. Proceedings of the International Computer Music Conference, Banff, Canada, 1995
- [Degazio, Bruno 1988] *Context Sensitive Editing in the MIDIFORTH Computer Music System*. Proceedings of the International Computer Music Conference, Cologne, Germany, 1988
- [Degazio, Bruno 1993] *New Software Composition Tools*. Fourth Biennial Arts and Technology Symposium. New London, Connecticut, 1993
- [Goldberg, 1989] *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley
- [Goldberg, 1990] *Real-coded Genetic Algorithms, virtual alphabets, and Blocking*. Illinois Genetic Algorithm Laboratory Report No.90001. Urbana: University of Illinois, Illinois Genetic Algorithms Laboratory
- [Goldberg, D.E. 1991/1994] *The Existential Pleasures of Genetic Algorithms*. IlliGAL Report No.94010. Urbana: University of Illinois, Illinois Genetic Algorithms Laboratory
- [Homer and Goldberg, 1991] *Genetic Algorithms and Computer-Assisted Music Composition*. Proc. ICMC-1991.
- [Horner, Beauchamp, Packard 1993] *Timbre Breeding*. Proc. ICMC-1993
- [Koza 1992] J.Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992
- [Lerdhal, F, Jackendoff, R 1983] *A Generative Theory of Tonal Music* MIT PRESS, Cambridge Massachusetts 1983
- [Levy 1992] *Artificial Life: The Quest for a New Creation*. Pantheon, New York, 1992
- [Sims 1991] *Artificial Evolution for Computer Graphics*. SIGGRAPH'91, ACM Computer Graphics, Vol.25, No.3
- [Takala et al 1993] *Using Physically-Based Models and Genetic Algorithms for Functional Composition of Sound Signals, Synchronised to Animated Motion*. Proc. ICMC-1993
- [Vuori and Valimaki 1993] *Parameter Estimation of Non-Linear Physical Models by Simulated Evolution - Application to the Flute Model*. Proc. ICMC-1993