At the Crossroads of Evolutionary Computation and Music: Self-Programming Synthesizers, Swarm Orchestras and the Origins of Melody

Eduardo Reck Miranda eduardo.miranda@plymouth.ac.uk
Computer Music Research, School of Computing, Communications and Electronics,
University of Plymouth, Drake Circus, Plymouth, Devon PL4 8AA, UK

Abstract

This paper introduces three approaches to using Evolutionary Computation (EC) in Music (namely, engineering, creative and musicological approaches) and discusses examples of representative systems that have been developed within the last decade, with emphasis on more recent and innovative works. We begin by reviewing engineering applications of EC in Music Technology such as Genetic Algorithms and Cellular Automata sound synthesis, followed by an introduction to applications where EC has been used to generate musical compositions. Next, we introduce ongoing research into EC models to study the origins of music and detail our own research work on modelling the evolution of melody.

Keywords

Self-programming audio synthesizers, cellular automata music, generative music, evolution of music, adaptive behaviour, cultural evolution.

1 Introduction

Perhaps the most interesting application of Evolutionary Computation (EC) to Music is for the study of the circumstances and mechanisms whereby musical compositions might originate and evolve in artificially created worlds inhabited by virtual communities of musicians and listeners. Origins and evolution are studied here in the context of the cultural conventions that may emerge under a number of constraints, including psychological, physiological and ecological constraints. A better understanding of the fundamental mechanisms of musical origins and evolution is of great importance for musicians looking for hitherto unexplored ways to create new musical works. As with the fields of Acoustics, Psychoacoustics and Artificial Intelligence, which have greatly contributed to our understanding of Music, Evolutionary Computation has the potential to reveal new aspects that are just waiting to be unveiled.

We have identified three main approaches to using EC in Music: *engineering*, *creative* and *musicological* approaches. Whereas the engineering approach applies EC to solve specific engineering problems in Music Technology, the creative approach employs EC to generate musical compositions. The musicological approach seeks answers to musicological problems by means of models and simulations. The questions that have been addressed by the musicological approach overlap with those considered in Evolutionary Linguistics (Cangelosi and Parisi, 2001): What functional theories of its

evolutionary origins make sense? How do learning and evolved components interact to shape the musical culture that develops over time? What are the dynamics of the spread of musical memes through a population? The quest for the origins of music is only one of the musicological applications of EC. Other applications are certain to emerge as the research in this new field develops.

2 Approaches to Evolutionary Computation and Music

2.1 The Engineering Approach

Genetic Algorithms (GA) and Genetic Programming (GP) are popular amongst engineers as techniques for solving problems that are not well understood, or whose solutions involve searching a huge space of combinatorial possibilities. They have been used in engineering for addressing problems for which numerical analysis is either unavailable or too complex to handle.

Musical activities are obviously not engineering problems, but the development of technology for musicians involves a great deal of software engineering and electronics. GA and GP can and have been successfully employed in Music Technology, especially in the field of sound synthesis. Here GA and GP have been used to search either in the space of synthesis parameter values for a given synthesizer or in the space of synthesizer architectures.

GA can be used effectively in two kinds of synthesis tasks: a) to find optimal synthesis parameters to reproduce a given sound (Horner et al., 1993) and b) to aid the user to explore the space of synthesis possibilities (Johnson, 1999; Mandelis, 2001). In both cases, the algorithm manipulates the synthesis parameters for a specific synthesizer; that is, the genotypes correspond to synthesis parameter values. The fundamental difference between these two general cases lies in the fitness evaluation process: whereas in the former case the fitness of a synthesized sound is given by the degree of similarity to a target sound, the fitness of the latter case is given by an interacting user, who ranks the sounds according to their own judgment.

Promising preliminary results on evolving synthesizer architectures using GP were reported at the 1st Workshop on Artificial Life Models for Musical Applications (ALMMA) held in Prague (Garcia, 2001).

The architecture of a sound synthesizer consists of a network of interconnected signal generators and processors. The search space here spans all the possible combinations of these elements and their connections. The system presented at the ALMMA workshop is far from being able to evolve neat complex synthesizers effectively. Nevertheless, it was possible to evolve simple interesting architectures from scratch. This certainly is a first step towards self-assembling musical instrument systems.

A different use of EC in sound synthesis technology is found in Chaosynth, a software for granular synthesis (Miranda, 1995; Miranda, 2000). Granular synthesis works by generating a rapid stream of very short sound bursts called grains (e.g. 35 milliseconds long) that together form larger sound events (Figure 1).

The problem with this synthesis technique is that one needs to specify parameter

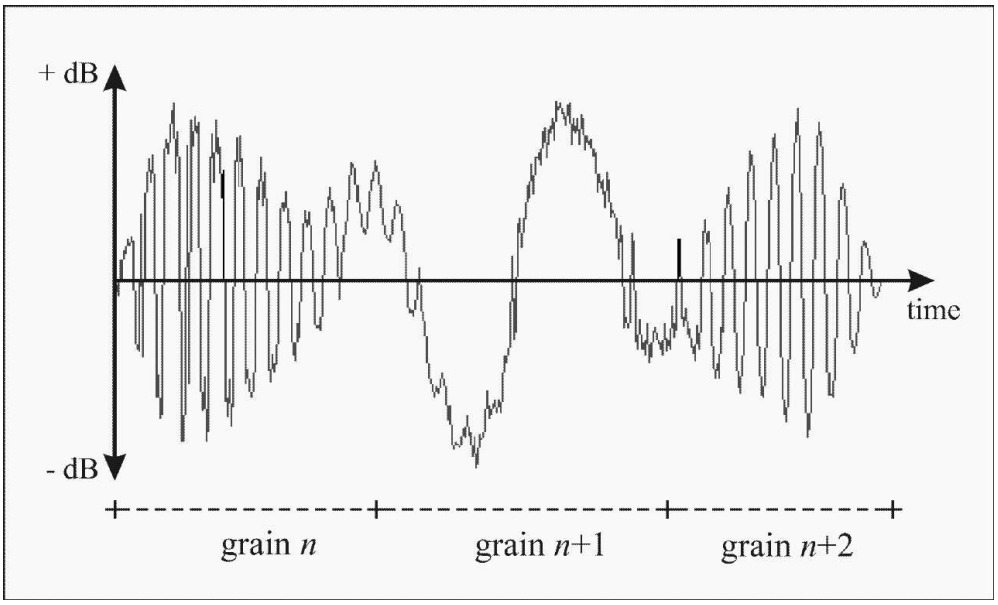

Figure 1: A sequence of three short sound grains. A rapid succession of thousands of such grains forms larger complex sounds.

values for the production of every single grain in the stream. Considering the simplest case, where each grain consists of a single sinusoid, one would still need to specify its frequency, amplitude and duration. It is very impractical to control the production of thousands of grains manually. The standard technique for controlling these grains automatically employs probability tables (Roads, 1996); for example, 10% of grains will have a frequency equal to 110 Hz, 30% equal to 300 Hz, and so forth. Unfortunately this technique does not produce satisfactory auditory results; the resulting sounds often lack the dynamic unfolding that the human ear would expect from a sound (Handel, 1993).

Chaosynth uses Cellular Automata (CA) to control the production of these grains. The technique of Chaosynth proved to be more efficient than probability tables and can produce granular sounds of very good quality. The CA-based grain controller uses an adapted version of a CA that has been used to model the behaviour of various oscillatory phenomena, such as Belouzow-Zhabotinsky chemical reactions (Dewdney, 1988). In general, this CA tends to evolve from an initial random distribution of cell values on the grid towards an oscillatory cycle of patterns. It can be thought of as a grid of cells representing identical simple electronic circuits. At a given moment, cells can be in any one of the following conditions: *quiescent*, *depolarised* or *burned*. A cell interacts with its neighbours (four or eight nearest cells) through the flow of electric current between them. There are minimum ($Vmin$) and maximum ($Vmax$) threshold values which characterize the condition of a cell. If its internal voltage (Vi) is under $Vmin$, then the cell is quiescent (or polarised). If it is between $Vmin$ (inclusive) and $Vmax$ values, then the cell is being depolarised. Each cell has a potential divider which is aimed at maintaining Vi below $Vmin$. But when it fails (that is, if Vi reaches $Vmin$) the cell becomes depolarised. There is also an electric capacitor, which regulates the

rate of depolarisation. The tendency, however, is to become increasingly depolarised with time. When Vi reaches $Vmax$, the cell fires and becomes burned. A burned cell at time t is automatically replaced by a new quiescent cell at time $t + 1$. In addition to the fundamental parameters mentioned earlier, the functioning of the automaton is also determined by the number n of possible cell values (or colours) such that $n \geq 3$, and the dimension (X, Y) of the grid. In practice, the state of the cells is represented by a number between 0 and $n - 1$, where n is the amount of different cell states. A cell in state $m[t] = 0$ corresponds to a quiescent state, whilst a cell in state $m[t] = n - 1$ corresponds to a burned state. All states in between represent a degree of depolarisation, according to their respective values. The closer a state value gets to $n - 1$, the more depolarised the cell becomes. One of the attractive features of this particular automaton is that it allows for a variable number of different cell states $n+2$.

The cells on the grid are updated by the application of following rules to all cells simultaneously:

- IF $m_{x,y}[t] = 0$ THEN $m_{x,y}[t + 1] = int(\frac{A}{r1}) + int(\frac{B}{r2})$
- IF $0 < m_{x,y}[t] < n - 1$ THEN $m_{x,y}[t + 1] = int(\frac{S}{A}) + k$
- IF $m_{x,y}[t] = n - 1$ THEN $m_{x,y}[t + 1] = 0$

where the state of a cell at a time t is denoted $m_{x,y}[t]$; x and y are the horizontal and vertical coordinates of a cell; A and B represent the number of collapsed and depolarised cells amongst the eight neighbours respectively; S stands for the sum of the neighbours' states; $r1$ and $r2$ represent the cell's resistance to becoming depolarised and k is the electrical capacitance of the cell. An example of the application of the rule to a certain cell (x, y) is given in Figure 2 and various snapshots taken from the cellular automaton in action are shown in Figure 3. This CA is interesting because of its dynamic self-organizing behaviour: it tends to evolve from an initial wide distribution of cell states in the grid towards oscillatory cycles of patterns (Figure 3). The behaviour of this CA therefore matches the type of dynamics we need for our synthesized granular sound: we need to convey signals that tend to evolve from a wide distribution of their spectrum at the onset, up to quasi-periodical oscillations. The parameters $r1$ and $r2$ and k allow for control of this transition.

The CA drives the production of the grains as follows: at each cycle t, the automaton produces one sound-grain (Figure 4). In this case, the values of the cells during the unfolding of the CA in time control the frequency, amplitude and length of each grain. The behaviour of a CA is normally visualized by associating different colours to the states of the cells. But in our case, the values of the cells are associated to frequencies and amplitudes, and oscillators are associated to a group of cells, or sub-grid. The frequency F and the amplitude A values for the oscillators are determined by the arithmetic mean over the frequency and the amplitude values associated to the states of the cells of their corresponding sub-grid (Figure 5):

$$F_i = \frac{\sum_{p=1}^{P} \phi_p}{P}$$

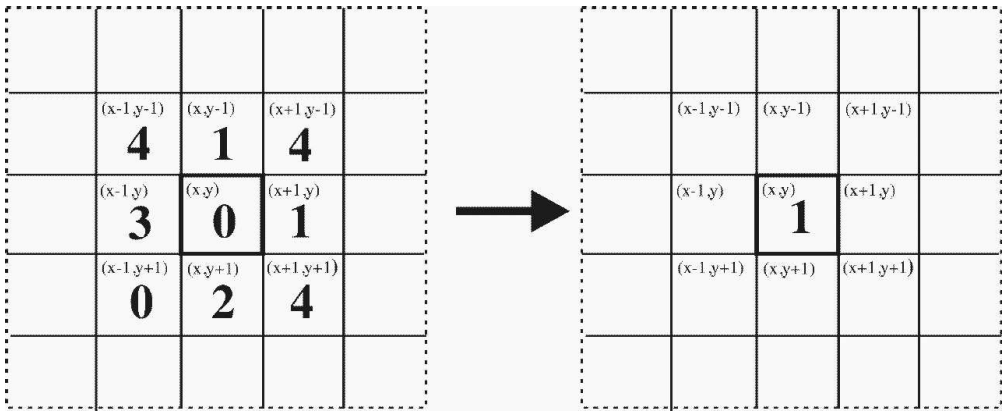

Figure 2: An example of the application of the transition rule to one cell. Assume that $m = \{0, 1, 2, 3, 4\}, r1 = 8.5, r2 = 5.2$ and $k = 3$. Considering the 8 neighbours of the cell (x, y), then $A = 3$ (number of burned cells) and $B = 4$ (number of depolarised cells). Since the cell (x, y) is quiescent, then the top condition of the rule applies and the value of this cell at the next tick of the clock will be equal to 1.

$$A_i = \frac{\sum_{p=1}^{P} \tau_p}{P}$$

where ϕ_p and τ_p are the frequency and amplitude of cell p, and P is the total amount of cells of the sub-grid. Suppose, for example, that each oscillator is associated to 9 cells and that at a certain time t, 3 cells correspond to 110 Hz, 2 to 220 Hz and the other 4 correspond to 880 Hz. In this case, the mean frequency value for this oscillator at time t will be 476.66 Hz. The duration D of a whole sound-stream is given by the total number of cycles of the cellular automaton and the duration of the individual grains (these are arbitrarily set beforehand); for example, 100 iterations of 35 millisecond particles result in a sound event of 3.5 seconds duration:

$$D = \sum_{g=1}^{G} d_g$$

where d_g is the duration of granule g and G is the amount of grains that form the whole stream. Considering that T is the total amount of cycles of a cellular automata run, then $G = T$. Note, however, this does not always necessarily need to be the case; for example, if we wish to skip some cycles during the synthesis process. An example of a grid of 400 cells allocated to 16 oscillators of 25 cells each is shown in Figure 5.

The resulting sounds are interesting because the synthesizer explores the behaviour of the CA in order to produce sounds in a way that resembles the evolution of natural sounds during their emission; their partials converge from a wide distribution to form oscillatory patterns. The random initialisation of states in the grid produces a wide initial distribution of frequency and amplitude values, which tend to settle into a periodic fluctuation.

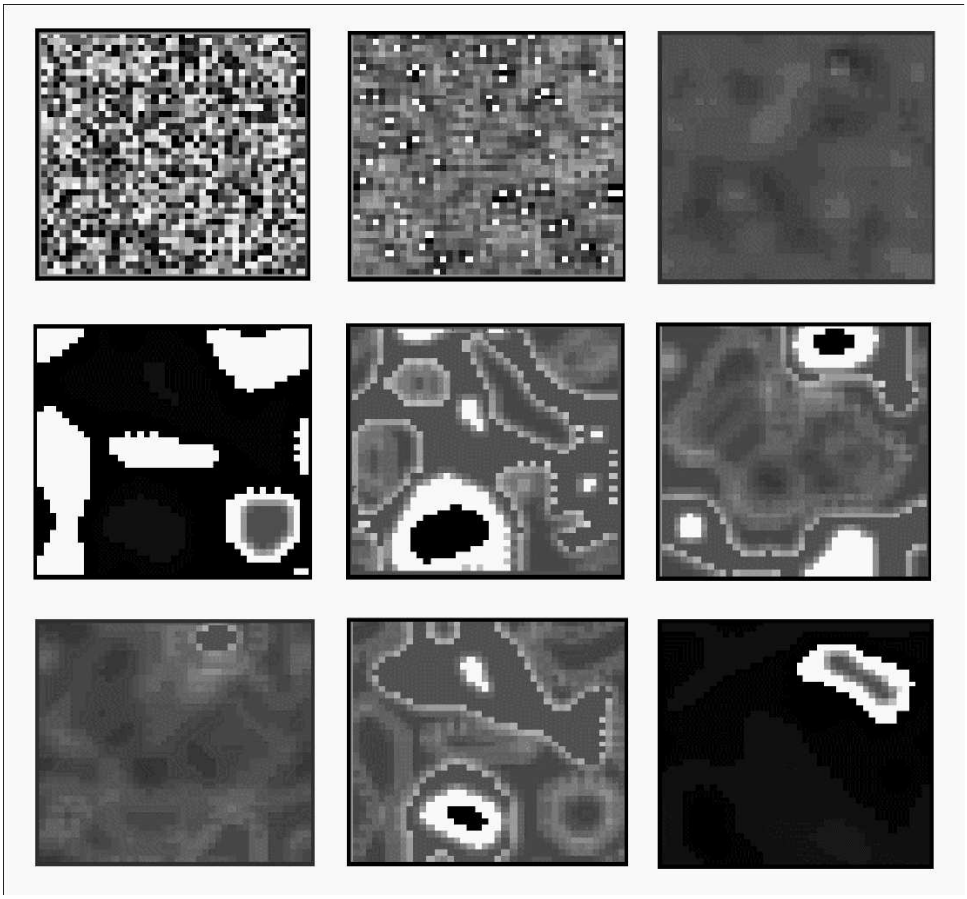

Figure 3: Various snapshots taken from our cellular automaton in action (from left to right, top to bottom).

EC methods can clearly provide plausible solutions to a number of engineering problems in the field of Music Technology. GA has great potential to be embedded in software synthesis interfaces in order to support the exploration of their synthesis capabilities. Also, GP combined with evolvable hardware technology (Zebulum et al., 1996) will probably be commonplace in future generations of musical synthesizers.

Music Technology can also benefit from EC to solve less orthodox problems, such as the granular synthesis control introduced above. Here the use of a model for an application for which it was not originally designed demonstrates the power of EC models for applications beyond the goals of standard EC research. Indeed, the next section will demonstrate how composers have been using EC for aiding musical creativity rather than engineering.

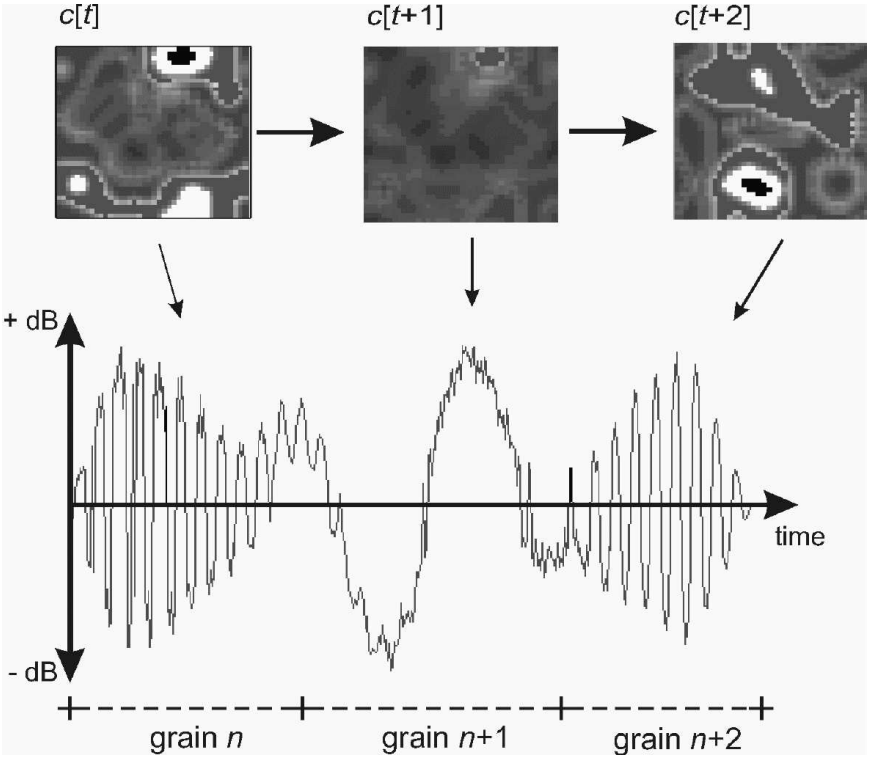

Figure 4: Each snapshot of the CA produces a sound-grain. (Note, however, that this is only a schematic representation, as the grains displayed here do not actually correspond to these particular snapshots.)

2.2 The Creative Approach

The great majority of Artificial Intelligence-based systems for generating musical compositions are either hard-wired to compose in a certain style or are able to learn how to imitate a style by looking at patterns in a bulk of training examples. Such systems are therefore good for imitating composers or well-established musical styles, such as medieval, baroque or even jazz. Conversely, issues such as whether computers can create new kinds of compositions are much harder to deal with, because in such cases the computer should neither be embedded with particular models at the outset nor learn from carefully selected examples. Since the invention of the computer, many composers have tried out mathematical models, which were thought to embody musical composition processes, such as combinatorial systems, grammars, stochastic models and fractals (Dodge and Jerse, 1985; Cope, 1991; Xenakis, 1971; Worral, 1996). Some of these trials have produced interesting pieces and much has been learned about using mathematical formalisms and computer models in musical composition. The use of EC models in generative musical systems is a natural progression to pushing this understanding even further.

There has been a number of experimental systems using EC models to generate musical compositions in the past decade or so, some of which were rather successful:

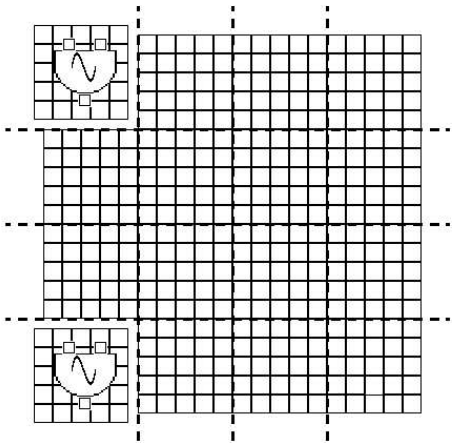

Figure 5: An example of a grid of 400 cells allocated to 16 digital oscillators.

Cellular Automata Music (Millen, 1990), a CA Music Workstation (Hunt et al., 1991), CAMUS (Miranda, 1993), MOE (Degazio, 1999), GenDash (Waschka, 1999), CAMUS 3D (McAlpine et al., 1999), Vox Populi (Manzolli et al., 1999), Synthetic Harmonies (Bilotta et al., 2000), Living Melodies (Dahlstedt and Nordahl, 2001) and Genophone (Mandelis, 2001), to cite but a few. Before GA became fashionable in musical applications, a number of people tried to explore the emergent behaviour of CA to generate musical compositions. The standard practice here was to associate the values (or states) of the CA cells to musical notes and then devise some strategy to generate the respective MIDI commands to play these notes on a synthesis device; see (Rumsey, 1994) for further information on MIDI. More sophisticated systems attempted to devise abstract representation schemas to mediate the mapping from CA behaviour onto actual musical structures rather than onto single notes (Figure 6).

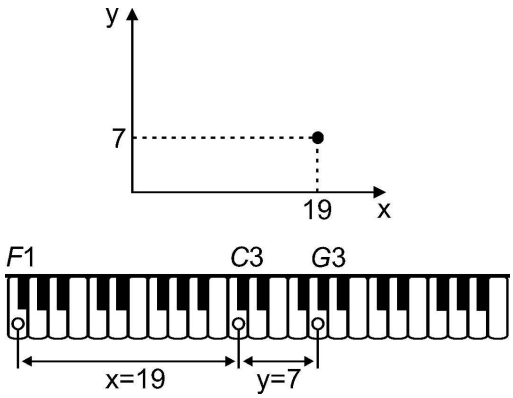

Figure 6: The CAMUS system goes beyond simplistic mapping of CA onto musical notes by adopting a two-dimension spatial representation whereby the co-ordinates of a cell in the space correspond to the distances between the notes of an ordered set of three musical notes.

GA-based generative musical systems generally follow the standard GA procedures for evolving musical materials such as melodies, rhythms, chords, and so on (Figure 7). For example, Vox Populi (Manzolli et al., 1999) evolves populations of chords of four notes, each of which is represented as a 7-bit string. The genotype of a chord therefore consists of a string of 28 bits (e.g., 1001011 0010011 0010110 00101010) and the genetic operations of crossover and mutation are applied to this code in order to produce new generations of the population. The fitness criteria takes three factors into account: melodic fitness, harmonic fitness and voice range fitness. The melodic fitness is evaluated by comparing the notes of the chord to a user-specified reference value. In short, this reference value determines a sort of tonal attractor: the closer the notes are to this reference value, the higher the fitness value. The harmonic fitness takes into account the consonance of the chord and the voice range fitness measures whether or not the notes of the chord are within a user-specified range (e.g., one octave). A straightforward user-interface provides sliders and other controls for the specification of fitness parameters and other musical attributes.

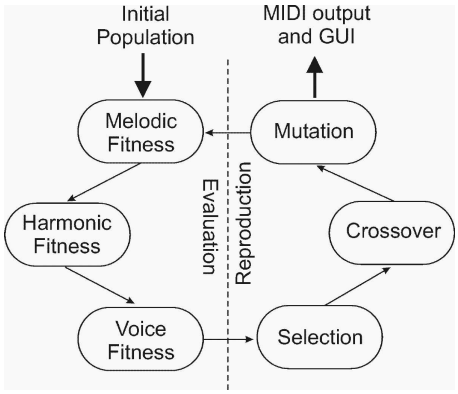

Figure 7: Vox Populi abides by the standard GA cycle but here the fitness criteria take three factors into account: melodic fitness, harmonic fitness and voice range fitness.

There has been increasing interest in the last decade for systems that compose interactively in real-time, as if the computer were acting as a fellow musician in a jam session (Rowe, 1993). A number of the GA-based systems referred to earlier allow for interactive use in real-time, whereby the user can, through various means, control GA operators and fitness values while the system is running. For example, composer Jonathan Impett has proposed an interesting swarm-like approach to interactive generative musical composition (Impett, 2001). Here, musical composition is modelled as a dynamic agent system consisting of interacting embodied behaviours. These behaviours can be physical or virtual and they can be emergent or composed beforehand. All behaviours co-exist and interact in the same world and are adaptive to the changing environment to which they belong. Such behaviours are autonomous and prone to aggregation and the generation of dynamically hierarchic structures.

This swarm orchestra is implemented using Santa Fe's Swarm simulation system (Minar et al., 1996). As Swarm was not originally intended for implementing real-time musical applications, Impett devised SwarmMIDI, a module that renders real-time

musical performance possible. By way of related work we cite Swarm Music, a system that also uses self-organization to produce interactive improvisations (Blackwell and Bentley, 2002).

2.3 The Musicological Approach

The musicological approach is represented by researchers in the quest for the origins of music by means of computer models and simulations. The pursuit of the origins of music is not new; philosophers throughout all ages have addressed this problem. As an example we cite the book *Music and the Origins of Language*, by Downing Thomas (1995) as an excellent review of the theories purported by philosophers of the French Enlightenment. And more recently, *The Origins of Music*, edited by Nils Wallin and colleagues (2000), collates a series of chapters written by top contemporary musicologists. With the exception of one chapter (Todd, 2000), however, none of these thinkers sought theoretical validation through computer modelling. Although we are aware that Musicology does not need such support to make sense, we do think, however, that computer simulation can be useful to develop and demonstrate specific musical theories.

Peter Todd and Gregory Werner (1999) proposed a system for studying the evolution of musical tunes in a community of virtual composers and critics. Inspired by the notion that some species of birds use tunes to attract a partner for mating, the model employs mating selective pressure to foster the evolution of fit composers of courting tunes. The model can co-evolve male composers who play tunes (i.e., sequences of notes) along with female critics who judge those songs and decide with whom to mate in order to produce the next generation of composers and critics.

Each composer holds a tune of 32 musical notes from a set of 24 different notes spanning two octaves. The critics encode a Markov-like chain that rates the transitions from one note to another in a heard tune. The chain is a 24-by-24 matrix, where each entry represents the female's expectation of the probability of one pitch following another in a song. Given these expectations, a critic can decide how well she likes a particular tune. When she listens to a composer, she considers the transition from the previous pitch to the current pitch for each note of the tune, gives each transition a score based on her transition table, and adds those scores to come up with her final evaluation of the tune. Each critic listens to the tunes of a certain number of composers who are randomly selected; all critics hear the same number of composers. After listening to all the composers in her courting-choir, the critic selects as her mate the composer who produces the tune with the highest score. This selective process ensures that all critics will have exactly one mate, but a composer can have a range of mates from none to many, depending on whether his tune is unpopular with everyone, or if he has a song that is universally liked by the critics. Each critic has one child per generation created via crossover and mutation with her chosen mate. This child will have a mix of the musical traits and preferences encoded in its mother and father. The sex of the child is randomly determined and a third of the population is removed at random after a mating session in order not to reach a population overflow.

From the many different scoring methods proposed to judge the tunes, the one that seems to produce the most interesting results is the method whereby critics enjoy

being surprised. Here the critic listens to each transition in the tune individually, computes how much she expected the transition, and subtracts this value from the probability that she attached to the transition she most expected to hear. For example, if a critic has a value 0.8 stored in her chain for the A-E transition, whenever she hears a note A in a tune, she would expect a note E to follow it 80% of the time. If she hears an A-C transition, then this transition will be taken as a surprise because it violates the A-E expectation. A score is calculated for all the transitions in the tune and the final sum registers how much surprise the critic experienced; that is, how much she likes the tune. What is interesting here is that this does not result in the composers generating random tunes all over the place. It turns out that in order to get a high surprise score, a tune must first build up expectations, by making transitions to notes that have highly anticipated notes following them, and then violate these expectations, by not using the highly anticipated note. Thus there is constant tension between doing what is expected and what is unexpected in each tune, but only highly surprising tunes are rewarded (Figure 8).

The composers are initiated with random tunes and the critics with tables set with probabilities calculated from a collection of folk-tune melodies. Overall, this model has shown that the selection of co-evolving male composers who generate attracting tunes, and female critics who assess these tunes according to their preferences, can lead to the evolution of tunes and the maintenance and continual turnover of tune diversity over time.

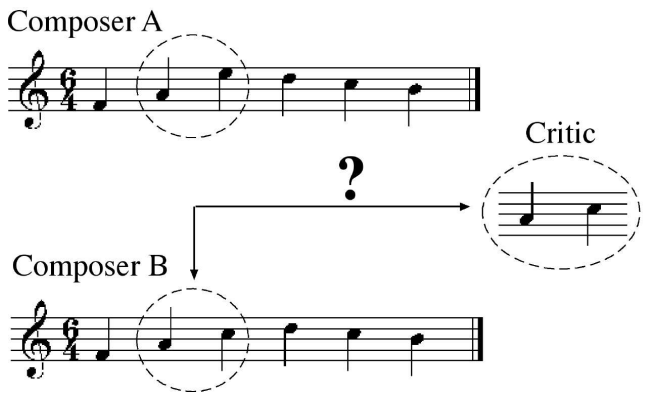

Figure 8: The critic selects composer B because it produces the most surprising tune.

This model is remarkable in the sense that it demonstrates how a Darwinian model with a pressure for survival mechanism can foster the evolution of coherent repertoires of melodies in a community of software agents. There is, however, a puzzling fundamental question that has not been addressed in this model: Where do the expectations of the female critics come from? Currently the model initialises their Markov chains with coefficients computed from samples of existing folk-tune melodies. Would it be possible to evolve such expectations from scratch? The following paragraphs introduce a model that may provide support for addressing this question.

We propose a *mimetic model* as an attempt to demonstrate that a small community

of interactive distributed agents furnished with appropriate motor, auditory and cognitive skills can evolve from scratch a shared repertoire of melodies, or tunes, after a period of spontaneous creation, adjustment and memory reinforcement (Miranda, 2002b). In this case, the tunes are not coded in the genes of the agents and the agents do not reproduce or die.

The motivation of the agents is to form a repertoire of tunes in their memories and foster social bonding. In order to be sociable, an agent must form a repertoire that is similar to the repertoire of its peers. Sociability is therefore assessed in terms of the similarity of the agents' repertoires. In addition to the ability to produce and hear sounds, the agents are born with a basic instinct: to imitate what they hear.

The agents are equipped with a voice synthesizer, a hearing apparatus, a memory device and an enacting script. The voice synthesizer is essentially implemented as a physical model of the human vocal mechanism (Miranda, 2002a). In short, the agents need to compute two parameters in order to produce a sound: $f(n)$ and $t(n)$, where $f(n)$ controls the pitch of the sound and $t(n)$ gives its triggering time; n corresponds to the ordering of these parameters in a sequence. As for the hearing apparatus, it employs short-term autocorrelation-based analysis to extract the pitch contour of a spoken signal. The algorithm features a parameter f that regulates the degree of attention of the hearing apparatus, by controlling the resolution of the short-term autocorrelation analysis (Miranda, 2001a). This resolution defines the sensitivity of the auditory perception of the agents.

The agent's memory stores its sound repertoire and other data such as probabilities, thresholds and reinforcement parameters. An agent processes and stores tunes in terms of synthesis and analysis parameters. They have a dual representation of tunes in their memories: a *motor map* (synthesis) and a *perceptual representation* (analysis). The motor representation is in terms of a function of motor (i.e. synthesis) parameters $f(n)$ and $t(n)$ and the perceptual representation is in terms of an abstract scheme we designed for representing melodic contour.

2.3.1 Abstract representation of melodic contour

A melodic unit (MU) is represented as a graph whose vertices stand for initial (or relative) pitch points and pitch movements, and the edges represent a directional path. Whilst the first vertex must have one outbound edge, the last one must have only one incoming edge. All vertices in between must have one incoming and one outbound edge each. Vertices can be of two types, initial pitch points (referred to as *p-ini) and pitch movements (referred to as* p-mov) as follows (Figure 9):

p-ini = {SM, SL, SH}

p-mov = {JUT, JU, SUT, SU, R, SD, SDB, JD, JDB}

where

SM = start MU in the middle register

SL = start MU in the lower register

SH = start MU in the higher register

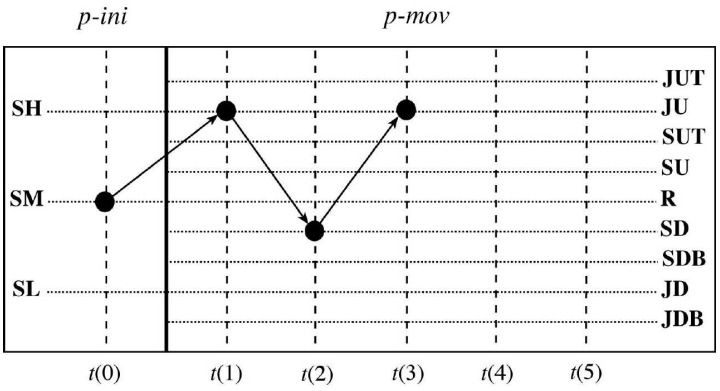

Figure 9: The representation of a melodic unit.

and

JUT = very large step up

JU = large step up

SUT = medium step up

SU = small step up

R = remain at the same band

SD = small step down

SDB = medium step down

JD = large step down

JDB = very large step down

An MU will invariably start with a *p-ini*, followed by one or more *p-mov*s. It is assumed that an MU can start at three different voice registers: low (SL), middle (SM) and high (SH). Then, from this initial point the next pitch might jump or step up or down, and so forth.

It is important to emphasize that musical labels or absolute pitch values are not relevant here because this scheme is intended to represent abstract melodic contours rather than a sequence of musical notes drawn from a specific tuning system.

2.3.2 The enacting script

Imitation is defined as the task of hearing a tune and activating the motor system to reproduce it. When we say that the agents should evolve a shared repertoire of tunes, we mean that the perceptual representation in the memory of the agents of the community should be identical, but the motor may be different. An important presupposition in this model is that the action of singing tunes involves the activation of certain vocal motor mechanisms in specific ways. The recognition of tunes here

thus requires knowledge of the activation of the right motor sequence (i.e. synthesis parameter values) in order to reproduce the tune in question. It does not matter, however, if one agent uses different values to its peers in order to produce identical tunes.

The enacting script provides the agent with knowledge of how to behave during the interactions: the agent must know what to do when another agent produces a tune, how to assess an imitation, when to remain quiet, and so forth. The enacting script does not evolve in the present model; all agents are alike in this respect. Also, all agents have identical synthesis and listening apparatus. At each round, each of the agents in a pair from the community plays one of two different roles: the *agent-player* and the *agent-imitator*; the main algorithm is given below. The agent-player starts the interaction by producing a tune p_1, randomly chosen from its repertoire. If its repertoire is empty, then it produces a random tune. The agent-imitator then analyses the tune p_1, searches for a similar tune in its repertoire (p_2) and produces it. The agent-player in turn analyses the tune p_2 and compares it with all other tunes in its own repertoire. If its repertoire holds no other tune p_n that is more perceptibly similar to p_2 than p_1 is, then the agent-player replays p_1 as a reassuring feedback for the agent-imitator; in this case the imitation would be acceptable (Figure 10). Conversely, if the agent-player finds another tune p_n that is more perceptibly similar to p_2 than p_1 is, then the imitation is unsatisfactory and in this case the agent-player would halt the interaction without emitting the reassuring feedback; the agent-imitator realizes that no feedback means imitation failure. If the agent-imitator hears the reassuring feedback, then it will reinforce the existence of p_2; in practice, the reinforcement is implemented as a counter that registers how many times a tune has successfully been used. Conversely, if the agent-imitator does not receive the feedback then it will infer that something went wrong with its imitation. In this case, the agent has to choose between two potential courses of action. If it finds out that p_2 is a weak tune (i.e., low past success rate) in its memory, because it has not received enough reinforcement in the past, then it will try to modify its representation of p_2 slightly, as an attempt to further approximate it to p_1. It is hoped that this approximation will give the tune a better chance of success if it is used again in another round. But if p_2 is a strong tune (i.e. good past success rate), then the agent will leave p_2 untouched (because it has been successfully used in previous imitations and a few other agents in the community also probably know it), will create a new tune that is similar to p_1, and will include it in its repertoire. The new tune is created as follows: the agent produces a number of random tunes and then it picks the one that is perceptually most similar p_1 to include in the repertoire. In case of failure, no reinforcement is applied to p_2. Before terminating the round, both agents perform final updates. Firstly they scan their repertoire and merge those tunes that are considered to be perceptibly identical to each other. Also, at the end of each round, both agents have a certain probability P_b of undertaking a spring-cleaning to get rid of weak tunes; those tunes that have not been sufficiently reinforced are forgotten. Finally, at the end of each round, the agent-imitator has a certain probability P_a of adding a new randomly created tune to its repertoire. The main algorithms of the enacting script are given as follows (the agent-player is represented by AP and the agent-imitator is represented by AI):

Algorithm 1: AP produces a tune
IF *repertoire(AP)* not empty

THEN
pick motor control for p_d
produce p_d
ELSE
generate random motor control for p_d
add p_d to *repertoire(AP)*
produce p_d

Algorithm 2: AI produces an imitation
analyse p_d
build perceptual representation
IF *repertoire(AI)* not empty
THEN
i_n = most perceptually similar to p_d
ELSE
generate random motor control for i_n
add i_n to *repertoire(AI)*
produce i_n

Algorithm 3: AP hears the imitation and gives a feedback
analyse i_n
build perceptual representation
p_n = most perceptually similar to i_n
IF $p_n = p_d$
THEN
send POSITIVE feedback to AI
reinforce p_d in *repertoire(AP)*
ELSE
send NEGATIVE feedback to AI

Algorithm 4: AI reacts to AP's feedback
IF feedback = POSITIVE
THEN
approximate i_n to p_d perceptually
generate appropriate motor control
reinforce i_n in *repertoire(AI)*
ELSE IF feedback = NEGATIVE
THEN IF i_n has good success history
THEN
execute *add-new-similar(tune)*
ELSE
modify motor representation of i_n towards p_d

Algorithm 5: Final updates
AP executes *final-updates(AP)*
AI executes *final-updates(AI)*

The *add-new-similar()* function works as follows: the agent produces a number of random imitations and then it picks the one that is perceptually most similar to p_d to

include in the repertoire.

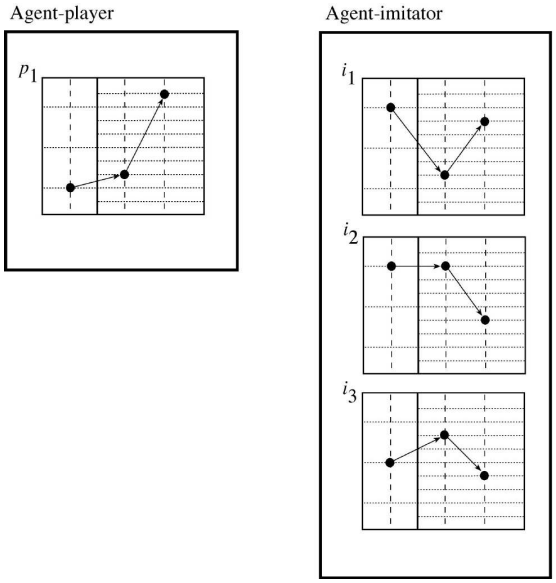

Figure 10: A case where the agent-player has only one tune in its repertoire whereas the agent-imitator has three. Since there is only one tune in the repertoire of the agent-player, any tune played by the agent-imitator will be considered to be an acceptable imitation, even though it might sound very different to an external observer. As far as this agent is concerned, both tunes are similar because it does not have yet the ability to distinguish between tunes.

2.3.3 Results and discussion

In order to evaluate and discuss the results of our simulations, let us consider a simple case study whereby a community of 5 agents performed 5000 interactions. The graph in Figure 11 shows the evolution of the average repertoire of the community, with snapshots taken after every 100 interactions. The agents quickly increase their repertoire to an average of between six and eight tunes per agent. At about 4000 interactions, more tunes appear, but at a lower rate. Identical behaviour has been observed in many such simulations with varied settings. The general tendency is to quickly settle into a repertoire of a certain size, which occasionally increases at lower rates. The pressure to increase the repertoire is mostly due to the creativity willingness parameter combined with the rate of new inclusions due to imitation failures. In this case the repertoire settled to 8 tunes between 1600 and 4000 interactions.

The graph in Figure 12 plots the imitation success rate of the community, measured at every 100 interactions. Notice that the success rate drops within the first 1000 interactions, which coincides with the steep rising curve of the graph in Figure 11. This is the period in which the agents are negotiating how their repertoires should look in order to foster communication; this period is characterized by inclusions of tunes due to imitation failure and by motor adjustments due to imitation successes.

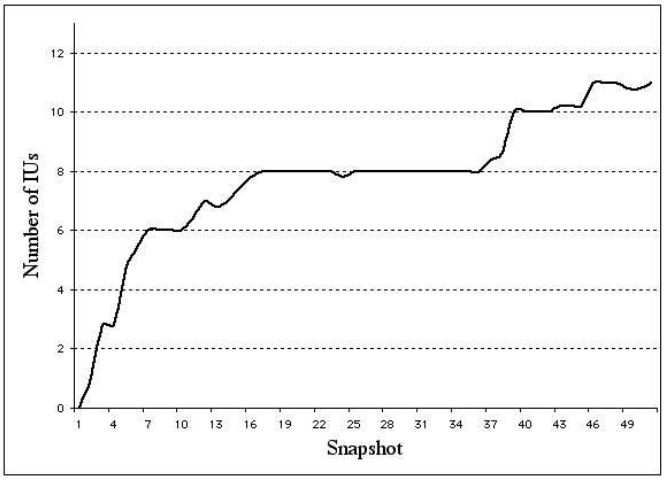

Figure 11: The evolution of the average size of the repertoire of the whole community (IU means intonation unit, or tune).

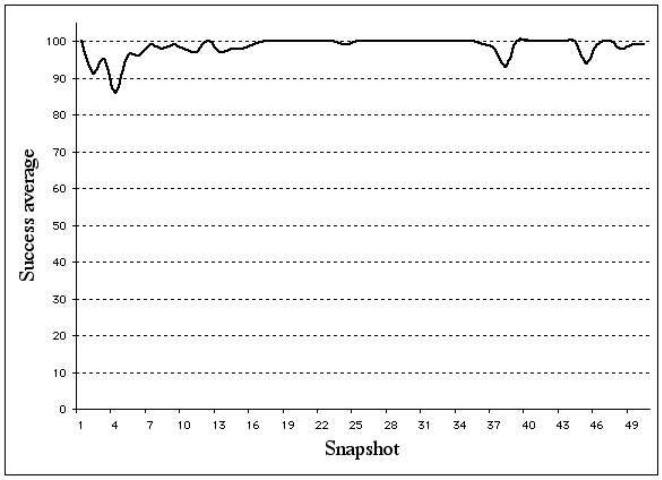

Figure 12: The imitation success rate over time.

At approximately 1800 interactions, the imitation rate goes back up to 100%. Then, occasional periods of lower success rate occur due to the appearance of new random tunes. Although the repertoire tends to increase with time, the success rate stays consistently high. This is good evidence that the community does manage to foster social bonding. How about the other goal of the agents? Did they evolve a shared repertoire of tunes?

The answer is yes. Figures 13(a) and 13(b) portray the perceptual memory of two agents randomly selected from the community, after 5000 interactions. The repertoire of all five agents plotted on top of each other is shown in Figure 13(c). Figure 13(c) demonstrates that at least two agents (the ones whose memories are plotted in Figures 13(a) and 13(b)) share identical tunes with the whole community.

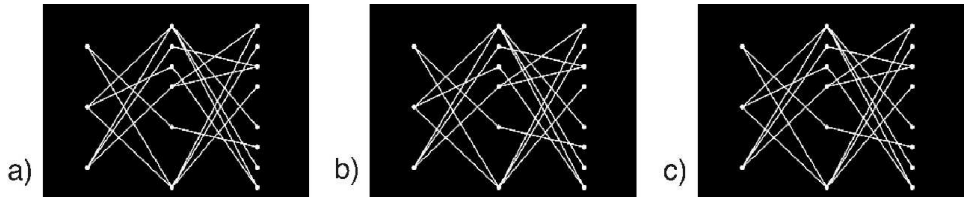

Figure 13: The perceptual memory of the agents. For the sake of clarity, the background metrics and labels of the graph are not shown.

Figures 14(a) and 14(b) plot the motor maps of two agents, also randomly selected from the community. Although both maps give an overall impression of similarity, the actual curves are slightly different. A better assessment can be made if both maps are plotted on the top of each other, as shown in Figure 14(c). This figure is a concrete example of a case where different motor maps yield the same perceptual representation.

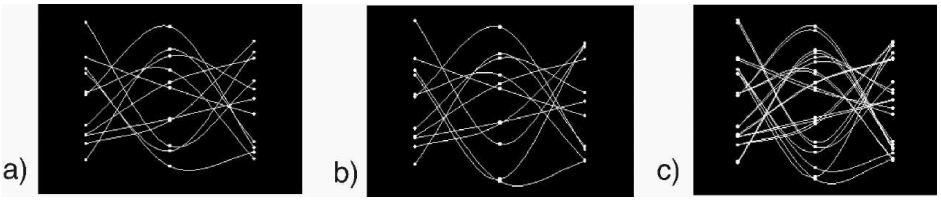

Figure 14: The motor memories of the agents. For the sake of clarity the figure omits background metrics and labelling; only curves of the motor functions are displayed. These are functions of synthesis parameters $f(n)$ and $t(n)$, for frequency and triggering time, respectively.

What can we learn from this model? We have demonstrated that imitation can be used as a mechanism for evolving from scratch a shared repertoire of musical tunes in a community of agents. What makes this model interesting is the fact that there is no global memory holding the repertoire of the community. The agents are autonomous individuals, each of them has its own memory and, most importantly, no other agent

can see what is stored in there. The only way to access the sound repertoire of a community is by considering the memories of all agents put together; e.g., Figure 13c. The repertoire of tunes emerges from the interactions of the agents and there is no global procedure supervising or regulating them; the actions of each agent are solely based upon their own local rules.

We do not assume the existence of a one-to-one mapping between perception and production. Although in this model we used one-to-one mapping, it is clear that the agents learn by themselves how to correlate perception parameters (analysis) with production (synthesis) ones. The agents do not necessarily need to build the same motor representations for what is considered to be perceptibly identical.

3 Conclusion

This paper has presented an overview of the major approaches to the interplay between EC and Music (engineering, creative and musicological approaches, respectively) and introduced a few representative systems that have been developed within the last decade. Particular attention has been given to developments that have potential to contribute to musicological research.

As far as the engineering approach is concerned, it is clear that EC provides solutions for problems that would otherwise have been very difficult to address. The creative approach has gained much popularity amongst composers interested in exploring the emergent behaviour of EC models for musical composition. Although there are not many professional EC-based compositional systems available, there are a great number of experimental ones that have been used to compose professionally. It seems that composers have managed to find good ways to use the emergent behaviour from EC models to compose musical pieces, and this is a trend that is here to stay (Miranda, 2001b). EC brings a different approach to building programs to aid music making in the sense that such systems are allowed to take an important part in the creative process. For example, EC-based interactive systems, such as those discussed in this paper, open a new dimension to interactive system design. Such EC-based systems can interact with musicians as if they were participating in a jam session. And more importantly, the musical behaviour of the system evolves during the interaction.

Unfortunately it is very difficult to evaluate EC-based systems for music because of the very nature of these systems. How can one evaluate a system for musical composition objectively? Engineering applications could be evaluated if we compared their results with those from non-EC solutions for the same problem. But in systems such as Chaosynth, for example, there is no quantitative measurement to evaluate the results; only our ears can tell if the sounds are of good quality or not. Evaluation does not apply in the musicological approach because EC is a research tool. Evaluation here takes place at the level of the theoretical framework of the specific musical theories in question.

As for the musicological approach, EC models have the potential to reveal new aspects of musical theory. The former two approaches are characterized by the fact that engineers and musicians often employ models that were developed for some purpose other than anything to do with Music; e.g., controlling a synthesizer with CA

or generating musical compositions with GA. These cases, however, offer very little potential for significant contribution to the advancement of Musicology. Conversely, the musicological approach forces researchers to devise EC models that are appropriate for addressing specific musicological questions. The mimetic model introduced earlier is an example of a model of a fundamental mechanism of musical learning. It demonstrates that imitation is a strong contender for any functional theory of musical evolution.

It is important to mention that all systems described in this paper were implemented on standard personal computers and/or Unix workstations and they can produced sounds and/or music in real-time. The only exception is the mimetic model discussed in section 2.3, which is slow due to the heavy computation required by the voice synthesizer and audio analysis modules; e.g., the mimetic simulations on the scale of the ones discussed in this paper can take up to 120 hours on a 300 MHz PowerPC.

References

Blackwell, T. M. and Bentley, P. (2002). Improvised Music with Swarms. In *Proceedings of the Congress of Evolutionary Computation*, pages 1462–1467, IEEE: New York, New York.

Bilotta, E., Pantano, P. and Talarico, V. (2000). Synthetic harmonies: an approach to musical semiosis by means of cellular automata. In Bedau, M. A., McCaskill, J. S., Packard, N. H. and Rasmussen, S., editors, *Proceedings of Artificial Life VII*, pages 537–546, The MIT Press, Cambridge, Massachusetts.

Cangelosi, A. and Parisi, D., editors, (2001). *Simulating the Evolution of Language*, Springer-Verlag, London, UK.

Cope, D. (1991). *Computers and Musical Style*, Oxford University Press, Oxford, UK.

Dahlstedt, P. and Nordhal, M. G. (2001). Living Melodies: Coevolution of Sonic Communication. *Leonardo*, 34(3):243–248.

Degazio, B. (1999). La evolucion de los organismos musicales. In Miranda, E. R., editor, *Musica y nuevas tecnologias: Perspectivas para el siglo XXI*, pages 137–148, L'Angelot, Barcelona, Spain.

Dewdney, A. K. (1988). The hodgepodge machine makes waves. *Scientific American*, August:86–89.

Dodge, C. and Jerse, T. (1985). *Computer Music*, Schirmer Books, London, UK.

Garcia, R. (2001). Growing Sound Synthesizers using Evolutionary Methods. In Bilotta, E., Miranda, E. R., Pantano, P. and Todd, P. M., editors, *Proceedings of ALMMA 2002 Workshop on Artificial Models for Musical Applications*, pages 99–107, Editoriale Bios, Cosenza, Italy.

Handel, S. (1993). *Listening: An Introduction to the Perception of Auditory Events*, The MIT Press, Cambridge, Massachusetts.

Horner, A., Beauchamp, J. and Haken, L. (1993). Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis. *Computer Music Journal*, 17(4):17–29.

Hunt, A., Kirk, R. and Orton, R. (1991). Musical Applications of a Cellular Automata Workstation. In *Proceedings of the International Computer Music Conference ICMC91*, pages 165–166, ICMA, San Francisco, California.

Impett, J. (2001). Interaction, Simulation and Invention: a Model for Interactive Music. In Bilotta, E., Miranda, E. R., Pantano, P. and Todd, P. M., editors, *Proceedings of ALMMA 2002 Workshop on Artificial Models for Musical Applications*, pages 108–119, Editoriale Bios, Cosenza, Italy.

Johnson, C. (1999). Exploring the sound-space of synthesis algorithms using interactive genetic algorithms. In *Proceedings of the AISB99 Symposium on Musical Creativity*, pages 20–27, AISB, Edinburgh, UK.

Mandelis, J. (2001). Genophone: An Evolutionary Approach to Sound Synthesis and Performance. In Bilotta, E., Miranda, E. R., Pantano, P. and Todd, P. M., editors, *Proceedings of ALMMA 2002 Workshop on Artificial Models for Musical Applications*, pages 108–119, Editoriale Bios, Cosenza, Italy.

Manzolli, J., Moroni, A., von Zuben, F. and Gudwin, R. (1999). An Evolutionary Approach Applied to Algorithmic Composition. In Miranda, E. R. and Ramalho, G. L., editors, *Proceedings of VI Brazilian Symposium on Computer Music*, pages 201–210, SBC/Entre Lugar, Rio de Janeiro, Brazil.

McAlpine, K., Miranda, E. R. and Hogar, S. (1999). Composing Music with Algorithms: A Case Study System. *Computer Music Journal*, 23(2):19–30.

Millen, D. (1990). Cellular Automata Music. In *Proceedings of the International Computer Music Conference ICMC90*, pages 314–316, ICMA, San Francisco, California.

Minar, N., Burckhart, R., Langton, C and Askenasi, V. (1996). The Swarm Simulation System: a Toolkit for Building Multi-Agent Systems. http://www.swarm.org/

Miranda, E. R. (1993). Cellular Automata Music: An Interdisciplinary Music Project. *Interface (Journal of New Music Research)*, 22(1):03–21.

Miranda, E. R. (1995). Granular Synthesis of Sounds by Means of Cellular Automata. *Leonardo*, 28(4):297–300.

Miranda, E. R. (2000). The Art of Rendering Sounds from Emergent Behaviour: Cellular Automata Granular Synthesis. In Vajda, F., editor, *Proceedings of the 25th EUROMICRO Conference*, pages 350–355 (volume 2), IEEE Computer Society, Los Alamitos, California.

Miranda, E. R. (2001a). Synthesising Prosody with Variable Resolution. *AES Convention Paper 5332*, Audio Engineering Society, New York, New York.

Miranda, E. R. (2001b). *Composing Music with Computers*, Elsevier - Focal Press, Oxford, UK.

Miranda, E. R. (2002a). *Computer Sound Design: Synthesis Techniques and Programming*, Elsevier - Focal Press, Oxford, UK.

Miranda, E. R. (2002b). Mimetic Development of Intonation. In *Proceedings of the 2nd International Conference on Music and Artificial Intelligence (ICMAI 2002) - Lecture Notes on Artificial Intelligence LNAI 2445*, pages 107–118, Springer-Verlag, Heidelberg, Germany.

Roads, C. (1996). *The Computer Music Tutorial*, The MIT Press, Cambridge, Massachusetts.

Rowe, R. (1993). *Interactive Music Systems*, The MIT Press, Cambridge, Massachusetts.

Rumsey, F. (1993). *Midi Systems and Control*, Elsevier - Focal Press, Oxford, UK.

Thomas, D. (1995). *Music and the Origins of Language*. Cambridge University Press, Cambridge, UK.

Todd, P. M. (2000). Simulating the evolution of musical behavior. In Wallin, N., Merker, B. and Brown, S., editors, *The origins of music*, pages 361–388, The MIT Press, Cambridge, Massachusetts.

Todd, P. M. and Werner, G. M. (2000). Frankensteinian Methods for Evolutionary Music Composition. In Griffith, N. and Todd, P. M., editors, *Musical networks: Parallel distributed perception and performance*, pages 313–339, The MIT Press - Bradford Books, Cambridge, Massachusetts.

Wallin, N. J., Merker, B. and Brown, S., editors, (2000). *The Origins of Music*, Cambridge University Press, Cambridge, UK.

Waschka II, R. (1999). Avoiding the Fitness Bottleneck: Using Genetic Algorithms to Compose Orchestral Music. In *Proceedings of the International Computer Music Conference ICMC99*, pages 201–203, ICMA, San Francisco, California.

Worral, D. (1996). Studies in metamusical methods for sound image and composition. *Organised Sound*, 1(3):183–194.

Xenakis, I. (1971). *Formalized Music*, Indiana University Press, Bloomington, Indiana.

Zebulum, R., Pacheco, P. and Vellasco, M. (1996). Evolvable Hardware Systems: Taxonomy, Survey and Applications. In *Proceedings of the First International Conference on Evolvable Systems*, pages 344–358, Tsukuba, Japan.