Voices of Artificial Life: On Making Music with Computer Models of Nature

Eduardo Reck Miranda
SONY Computer Science Laboratory Paris
6 rue Amyot, 75005 Paris, France
miranda@csl.sony.fr

Abstract
We are investigating the potential of artificial life (Alife) models for composition and evolutionary musicology. This paper begins with a brief introduction to our research scenario and then revisits two systems of our own design that use cellular automata to control a sound synthesiser and to generate musical passages: Chaosynth and CAMUS. Next, we introduce a discussion on the potential and limitations of these two systems. Then, we present a new paradigm for Alife in music inspired by the notion of adaptive distributed-agent systems. We demonstrate how a small community of agents furnished with a voice synthesiser, a hearing system and a memory mechanism can evolve a shared repertoire of melodic patterns from scratch, after a period of spontaneous creation, adjustment and memory reinforcement.

Keywords: Artificial life, cellular automata, evolutionary musicology, algorithmic composition, generative music, adaptive behaviour, collective machine learning.

1. INTRODUCTION
Perhaps one the greatest achievements of Artificial Intelligence (AI) to date lies in the construction of machines that can compose music of incredibly high quality; e.g. (Cope, 1991). These AI systems (Miranda, 2000) are only any good, however, at mimicking well-known musical styles. They are either hard-wired to compose in a certain style or able to learn how to imitate a style by looking at patterns in a batch of training examples. Conversely, issues as to whether computers can create new kinds of music are much harder to study, because in such cases the computer should neither be embedded with particular models at the outset, nor learn from carefully selected examples. One plausible approach to address this problem is to program the computer with abstract models that embody our understanding of the dynamics of some compositional processes. Indeed, many composers have tried out mathematical models, which were thought to embody musical composition processes, such as combinatorial systems, stochastic models and fractals (Dodge, 1985; Worrall, 1996; Xenakis, 1971). Some of these trials produced interesting music and much has been learned about using mathematical formalisms and computer models in composition. We believe Alife modelling techniques to be a natural progression to pushing this understanding even further. Also, as we shall see below, such models can be very useful for evolutionary musicology (Wallin et al., 2000). By ‘artificial life models’, we mean those computational models that display some form of emergent behaviour resembling natural phenomena; for example, cellular automata, genetic algorithms and adaptive games, to cite but a few (Kelemen and Sosik, 2001).

We have been investigating the potential of a class of Alife modelling techniques called cellular automata (CA) for music composition for almost a decade, but we have recently shifted our attention towards models inspired by the notion of adaptive distributed-agent systems. This paper begins with a brief review of two representative systems resulting from our research into cellular automata: a software synthesiser called Chaosynth and a generator of musical passages called CAMUS. Then, we discuss the potential and limitations of these two systems based upon the experience that we have gained from using them in professional composition. In the light of this short discussion, we introduce one of the new models that we have implemented to study the evolution of melody in an artificial society of distributed agents.

2. CELLULAR AUTOMATA INVESTIGATION: FROM SOUND SYNTHESIS TO MUSICAL FORM
Cellular automata (CA) are discrete dynamical systems often described as a counterpart to partial differential equations, which have the capability to
describe continuous dynamical systems. The meaning of discrete is that space, time and properties of the automata can have only a finite, countable number of states. The basic idea is not to try to describe a complex system using difficult equations, but rather to simulate this system by the interaction of its components following simple rules. CA are implemented as an array of identically programmed automata, or cells, that interact with one another. The arrays usually form either a 1-dimensional string of cells, a 2-D grid, or a 3-D solid. Most often the cells are arranged as a simple regular grid or matrix, but other arrangements, such as a honeycomb, are sometimes used. The essential features of CA are: the state of the cells and their neighbourhood. The state of a cell can be either a number or a property. For instance if each cell represents part of a landscape, then the state might represent the number of animals at each location or the type of forest cover growing there. The neighbourhood is the set of cells that a single cell interacts with. In a grid these are normally the cells physically closest to the cell in question.

Chaosynth is essentially a granular synthesiser (Miranda, 1998). Granular synthesis works by generating a rapid succession of very short sound bursts called grains (e.g. 35 milliseconds long) that together form larger sound events.

Whereas most granular synthesis systems so far have used stochasticity to control the production of the grains (for example, to control the waveform and the duration of the individual grains), Chaosynth uses CA.

The CA used in Chaosynth tend to evolve from an initial random distribution of cell values on the grid, towards an oscillatory cycle of patterns. These CA can be thought of as a grid of cells representing identical 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 (4 or 8 nearest cells) through the flow of electric current between them. There are minimum ($V_{min}$) and maximum ($V_{max}$) threshold values which characterise the condition of a cell. If its internal voltage ($V_i$) is under $V_{min}$, then the cell is quiescent (or polarised). If it is between $V_{min}$ (inclusive) and $V_{max}$ values, then the cell is being depolarised. Each cell has a potential divider which is aimed at maintaining $V_i$ below $V_{min}$. But when it fails (that is, if $V_i$ reaches $V_{min}$) 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 $V_i$ reaches $V_{max}$, the cell fires and becomes burned. A burned cell at time $t$ is automatically replaced by a new quiescent cell at time $t+1$.

Each configuration of cells $c$ of the CA evolution in time $t$ controls the frequency and duration values of each grain (Figure 1); the amplitude values are given via another procedure. The values (i.e. the colours) of the cells are associated to frequencies and oscillators are associated to a number of cells. The frequencies of the components of a grain at time $t$ are established by the arithmetic mean of the frequencies associated with the values of the cells associated with the respective oscillators. Suppose, for example, that each oscillator is associated with 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 user can also specify the dimension of the grid, the amount of oscillators, the allocation of cells to oscillators, the allocation of frequencies to CA values, and various other CA-related parameters. An example of a grid of 400 cells allocated to 16 oscillators of 25 cells each is shown in Figure 2. The duration of a whole sound event is determined by the number of CA generations and the duration of the individual grains; for example, 100 iterations of 35 millisecond particles result in a sound event of 3.5 seconds of duration. More details about Chaosynth and references to related work can be found in a paper that appeared in *Leonardo* (Miranda, 1995).

**Figure 1:** Each screen of the CA evolution controls the production of an individual granule.
Figure 2: An example of a grid of 400 cells allocated to 16 digital oscillators.

CAMUS uses two simultaneous CA to generate musical passages in MIDI format: the Game of Life and Demon Cyclic Space. Due to limitations of space, we will briefly introduce only the role of the Game of Life in the generative process. More information on CAMUS and references to related work can be obtained in a paper that appeared in the Computer Music Journal (McAlpine et al., 1999) or in this author’s book Composing Music with Computers (Miranda, 2001a). The Game of Life is an attempt to model a colony of simple virtual organisms. The CA are defined as a matrix of cells, each of which can be in one of two possible states: alive represented by the number one (coloured in black), or dead represented by the number zero (coloured in white) (Figure 3). The state of the cells as time progresses is determined by the state of the eight nearest neighbouring cells. There are essentially four rules that determine the fate of the cells of the Game of Life CA: (a) Birth: A cell that is dead at time $t$ becomes alive at time $t+1$ if exactly three of its neighbours are alive at time $t$; (b) Death by overcrowding: A cell that is alive at time $t$ will die at time $t+1$ if four or more of its neighbours are alive at time $t$; (c) Death by exposure: A cell that is alive at time $t$ will die at time $t+1$ only if it has one or none live neighbours at time $t$; (d) Survival: A cell that is alive at time $t$ will remain alive at time $t+1$ only if it has either two or three live neighbours at time $t$. A number of alternative rules can set, but not all of them produce interesting emergent behaviour.

CAMUS uses a Cartesian model in order to represent a triple of notes; that is, an ordered set of three notes that may or may not sound simultaneously. These three notes are defined in terms of the intervals between them. Given a starting note, the horizontal co-ordinate of the model represents the first interval of the triple and the vertical co-ordinate represents its second interval (Figure 4).

To begin the generative music process, the CA is set up with an initial random configuration of cell values and let to run. When the algorithm arrives at a live cell, its co-ordinates are taken to estimate the triple from a given lowest reference note. For example, if a cell at the position (19, 7) is alive, the co-ordinates (19, 7) describe the intervals of a triple of notes: a fundamental pitch is given, then the next note will be at 19 semitones above the fundamental and the last note 26 semitones above the fundamental (Figure 4). Although the cell updates occur at each time step in parallel, CAMUS plays the live cells column by column, from top to bottom. Each of these musical cells has its own timing, but the notes within a cell can be of different lengths and can be triggered at different times. Once the triple of notes for each cell has been determined, the states of the neighbouring cells are used to calculate a timing template, according to a set of temporal codes. More information about this temporal representation can be found in a paper that appeared in Interface (now called Journal of New Music Research) (Miranda, 1993). As a brief example, if we assume that Figure 5 portrays the temporal template for a certain live cell (5, 5), then a musical passage that could be generated by this cell is given in Figure 6.

Figure 3: Game of Life in action.

Figure 4: CAMUS uses a Cartesian model in order to represent a triple of notes.

Figure 5: An example of a template for the organisation of a cell’s triplet. The horizontal axis represents time and the vertical axis pitch.
2.1 Brief discussion of the results

CAMUS is available on the CD-ROM of the book *Composing Music with Computers* cited earlier and Chaosynth is commercially available from Nyr Sound in the UK. [http://www.ynysound.com]. A number of professional pieces were composed using CAMUS-generated material, such as *Entre o Absurdo e o Mistério* (Miranda, 2001b), for chamber orchestra, the second movement of the string quartet *Wee Batucada Scotia* (Miranda, 1998b) and more recently *Grain Streams*, for (acoustic) piano, sampler and live electronics. Chaosynth has also proved to be a successful system in the sense that it is able to synthesise a large amount of unusual sounds that are not normally found in the real acoustic world, but nonetheless sound pleasing to the ear. As an example of an electroacoustic piece composed using the sounds of Chaosynth we cite *Oliveine Trees*, which was awarded a bronze medal at the International Luigi Russolo Electroacoustic Music Competition in Italy. The results of the CA experiments are very encouraging, as they are good evidence that both musical sounds and abstract musical forms might indeed share similar organisational principles with cellular automata, most notably the principles of *pattern propagation* and *variation of musical forms*. In general, we found that Chaosynth produced more interesting results than CAMUS. We think that this might be due to the very nature of the phenomena in question. The inner structures of sounds seem more susceptible to CA modelling than large musical structures. As music is primarily a cultural phenomenon, we have come to the conclusion that it is necessary to take into account the dynamics of social formation and cultural evolution when designing generative music systems. We are currently devising a number of experiments aimed at models that take these aspects into account. The remainder of this paper introduces an experiment whereby the entities of a small virtual community evolve a common repertoire of melodic patterns from scratch by interacting with one another.

3. A NEW BREEDING GROUND FOR EVOLVING MUSICAL FORMS

This model is an attempt to demonstrate that a small community of interactive distributed agents furnished with appropriate motor, auditory and cognitive skills can evolve a shared repertoire of melodic patterns from scratch, after a period of spontaneous creation, adjustment and memory reinforcement. The motivation of the agents is to form a repertoire of melodic patterns 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.

Why imitation? The mimetic instinct is one of our most fundamental instincts. Life is animated with mutual interest between individuals who interact with one another to express reciprocity of intentions, co-ordination of skilful tasks, and so on. There is strong evidence that mimesis, or the ability to imitate the actions of other people, animals and nature, is a defining feature of the first evolutionary steps to human intelligence (Nadel and Butterworth, 1999; Tzervanthen et al., 1999).

The agents are equipped with a voice synthesiser, a hearing apparatus, a memory device and an enacting script. The voice synthesiser is essentially implemented as a physical model of the human vocal mechanism (Miranda, 2002). 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 \( m \) that regulates the degree of attention of the hearing apparatus, by controlling the resolution of the short-term autocorrelation analysis. This resolution defines the sensibility 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. In the present simulation, the memory holds values for the following parameters (their meaning will be clarified as we go along): \( P_f \) = creative willingness; \( P_b \) = forgetfulness disposition; \( H_b \) = bad history threshold; \( H^* \) = success history threshold; \( E_T \) = erase threshold; \( R^* \) = reinforcement threshold; \( J_T \) = jump threshold; \( D \) = motor deviation coefficient; \( a \) = motor approximation factor; \( m \) = degree of attention; \( L_{\text{min}} \) = minimum length and \( L_{\text{max}} \) = maximum length. The agents process and store melodies in terms of synthesis and analysis parameters. In fact they have a dual
representation of melodies in their memories: a motor map (synthesis) and a perceptual representation (analysis). The perceptual representation is in terms of the CARMEC representation scheme (refer to the Appendix) and the motor representation is in terms of a function $\phi$ of ‘motor’ (i.e. synthesis) parameters $\mathbf{f}(n)$ and $\mathbf{f}(n)$.

Imitation is defined as the task of hearing a sound and activating the motor system to reproduce it. When we say that the agents should evolve a shared repertoire of sounds, we mean that the perceptual representation in the memory of the agents of the community should be identical, but the motor may, and indeed should be different. Figure 7 shows an example where two motor maps yield an identical perceptual representation.

As for the enacting script, it provides the agent with knowledge of how to behave during the interactions: the agent must know what to do when another agent produces a melodic pattern, how to assess an imitation, when to remain quiet, and so forth. By and large, it is the enacting script that determines the dynamics of the interactions. 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 apparatuses. 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 agent-player starts the interaction by producing a melody $p_1$ randomly chosen from its repertoire. If its repertoire is empty, then it produces a random melody. The agent-imitator then analyses the melody $p_1$, searches for a similar melody in its repertoire, $p_2$, and produces it. The agent-player in turn analyses the melody $p_2$ and compares it with all other melodies in its own repertoire. If its repertoire holds no other melody $p_n$ that is more perceptibly similar to $p_1$ 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. Conversely, if the agent-player finds another melody $p_n$ that is more perceptibly similar to $p_1$ 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. In all these cases, the agents always add a small deviation to the motor realization of the melodies they know; the amount of deviation is determined by the coefficient $n$. If the agent-imitator hears the reassuring feedback, then it will reinforce the existence of $p_1$ in its repertoire and will change its motor parameters slightly as an attempt to make the melody even more similar to $p_1$. In practice, the reinforcement is implemented as a counter that registers how many times a melody 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 $p_1$ is a ‘weak melody’ (i.e. bad past success rate) in its memory, because it has not received enough reinforcement in the past, then it will try to modify its perceptual representation of $p_1$ slightly, as an attempt to further approximate it to $p_1$. It is hoped that this perceptual approximation will give the melody a better chance of success if it is used again in another round. But if $p_1$ is a ‘strong melody’ (i.e. good past success rate), then the agent will leave $p_1$ 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 melody that is similar to $n$ and will include it in its repertoire. The new sound is created as follows: the agent produces a number of random sounds and then it picks the one that is perceptually most similar to $p_1$ to include in the repertoire. In case of failure, no reinforcement is applied to $p_1$. Before terminating the round, both agents perform final updates. Firstly they scan their repertoire and merge those melodies that are considered to be perceptibly identical to each other. Also, at the end of each round, both agents have a certain probability $P_n$ of undertaking a spring-cleaning to get rid of weak melodies; those melodies that have not been sufficiently reinforced are deleted. Finally, at the end of each round, the agent-imitator has a certain probability $P_n$ of adding a new randomly created melody unit to its repertoire.

Figure 7: An example where two motor maps (top) yield the same perceptual representation (bottom). In this case, the degree of attention of the hearing apparatus was set to a medium resolution.

Bear in mind that an important presupposition in our research is that the action of singing involves the activation of certain vocal motor mechanisms in specific ways. The recognition of melodies in our simulations thus requires knowledge of the activation of the right sequence of motor (i.e. synthesis) parameter values in order to reproduce the melody in
question. It does not matter, however, if one agent uses different values to its peers in order to produce identical melodies.

3.1 Case study and discussion

In order to evaluate and discuss the results of our simulations, let us consider a case study whereby a community of 5 agents performed 5000 interactions, with the following settings: \( P_a = \) probability of 0.5%; \( P_b = \) probability of 20%; \( H_f = 0.1\% \) of total amount of interactions of the round; \( H = 90\% \) of interactions so far; \( E_f = 1\% \) of total amount of interactions of the round; \( R = 95\% \) of interactions so far; \( J_p = 0.25 \) in a scale from 0.0 to 1.0; \( D = +/- 1\% \); \( a = 2\% \) of the distance; \( m = 0.06 \) (middle resolution); \( L_{\text{min}} = 3 \) and \( L_{\text{max}} = 3.\)

![Figure 8: The evolution of the average size of the repertoire of the whole community.](image)

The graph in Figure 8 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 melodies per agent. At about 4000 interactions, more melodies 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 increase 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 melodic patterns between 1600 and 4000 interactions.

![Figure 9: The imitation success rate over time.](image)

The graph in Figure 9 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 8. 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 melodies due to imitation failure and by motor adjustments due to imitation successes. 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 patterns or eventual motor-perceptual inconsistencies that might be caused by pattern approximations and/or deviations. Although the repertoire tends to increase with time, the success rate is maintained consistently high. This is a 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 melodic patterns?

![Figure 10: The perceptual memory of the agents. For the sake of clarity, the background metrics and labels of the graph are not shown; refer to Figure 12 in the Appendix if necessary.](image)

The answer is yes. Figures 10(a) and 10(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 10(c). Figure 10(c) demonstrated that at least two agents (the ones whose memories are plotted in Figures 10(a) and 10(b)) share identical patterns with the whole community. Figures 11(a) and 11(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 11(c). Figures 11(c) is a concrete example of a case where different motor maps yield the same perceptual representation.

4. DISCUSSIONS AND CONCLUSION

What can we learn from this model? We have demonstrated that imitation is important for the evolution of music, more specifically melodic patterns, in the context of our research. But 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 10c. Moreover, the repertoire of melodies emerges from the interactions of the agents, but there is no global procedure supervising or regulating them. The actions of each agent are solely based upon its own internal rules.

We do not assume the existence of a one-to-one mapping between perception and production. Although in this model we forged 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 perceptually identical.

But why are such features important for generative music and how can such a model contribute to evolutionary musicology? Perhaps the breakthrough of this work is that we introduced the notion of active collective machine learning to generative music systems. Instead of modelling specific musical styles or compositional processes, this model is a first step towards the implementation of collective music-making systems whereby virtual musicians learn how to compose and play by themselves, and eventually with people interacting with them. The novelty here is that these systems should not need to learn passively, as is the case with most compositional tools that have been designed so far. Our hypothesis is that if we furnish these virtual musicians with the proper cognitive and physical abilities, combined with appropriate interaction dynamics and adequate environmental conditions, then the virtual society should be able to evolve its own realistic musical cultures. Such models can therefore be very useful for evolutionary musicology, in addition to composition.

From a composer’s point of view, although our agents are capable of evolving repertoires of simple melodies, we can hardly say that they can compose music at this stage. We will only be able to maintain that such a system can actually create music when more complex models for evolving multi-levelled generative procedures have successfully been implemented. There are still, however, a number of questions that need to be addressed before we set out to design such complex models. For example, we do not know what would happen if we gave the agents the choice of a sound-making device other than the voice, and/or created agents with diverse cognitive skills. Also, we need to consider that the agents should also be able to evolve (e.g., grow up, get smarter, get older, etc.) and therefore the enacting script should evolve too.

5. REFERENCES


6. APPENDIX: Common Abstract Representation of Melodic Contour (CARMEC)

Note labels or absolute pitch values are not relevant this investigation because we are dealing with generic abstract melodic contours, as opposed to a sequence of notes drawn from a specific tuning system. To this end we designed CARMEC, short for Common Abstract Representation of Melodic Contour. CARMEC represents a melodic pattern 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: p-ini = \{SM, SL, SH, SR\} and p-mov = \{JDB, JD, SDB, SD, R, SU, SUU, JU, JUU\} where: SM = start MU in the middle register; SL = start MU in the lower register; SH = start MU in the higher register; SR = start MU in a relative register; and JDB = jump down to the bottom; JD = simply jump down; SDB = step down to the bottom; SD = simply step down; R = remain at the same level; SU = simply step up; SUT = step up to the top; JU = simply jump up and JUT = jump up to the top.

![Figure 12: The representation of a melodic pattern.](image)

A melodic pattern will invariably start with a p-ini, followed by one or more p-movs. It is assumed that a 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. An example of a melodic contour of four ‘notes’ is illustrated in Figure 12.