ARTIFICIAL EMBRYOLOGY

The Genetic Programming of an Artificial Embryo

Hugo de Garis

CADEPS Artificial Intelligence
and Artificial Life Research Unit,
Universite Libre de Bruxelles (U.L.B.),
Ave F.D. Roosevelt 50, C.P. 194/7,
B-1050, Brussels, Belgium, Europe.
tel + 32 2 650 2783,
fax + 32 2 650 2785,
email CADEPS@BBRNSF11.BITNET

Center for Artificial Intelligence,
Computer Science Department,
George Mason University,
4400 University Drive, Fairfax,
Virginia, VA 22030, USA.
tel + 1 703 764 6328,
fax + 1 703 323 2630,
email HUGODEG@AIC.GMU.EDU

Keywords :

Genetic Programming, Genetic Algorithm, Hyper-Complex Systems, Artificial Life, Cellular Automata, Connection Machine, Artificial Embryology.

Abstract :

This chapter introduces the concept of Genetic Programming (GP) and its application to the "growth" of an artificial embryo. Genetic Programming (GP) is the application of the Genetic Algorithm [5,6] to building (evolving) functional systems which are too complex in their dynamics or their interactions to be pre-specified or analyzed in detail. Such systems can be built, but (probably) not understood.

1. Introduction

This paper introduces the concept of Genetic Programming in a more general way than in earlier papers [1,2,3,4]. Genetic Programming (GP) is the application of the Genetic Algorithm (GA) to the creation of systems which are too complex in their dynamics or interactions to be pre-specified or analyzed in detail. Such systems can be built (evolved), but (probably) not understood. This paper shows how the GP approach can be applied to the evolution of an artificial embryo. Section 2 presents the "Embryo Algorithm" and some early results in 2D. Section 3 presents results of evolving shapes in 2D and 3D on a Connection Machine at Boston University. Section 4 presents some remarks on the above sections and a suggestion as to how truly arbitrary (non convex) shapes might be grown without ad hoc interventions.

This chapter is concerned with the idea that it is possible to grow artificial embryos in two and three dimensions even though one does not understand fully how this occurs. This idea, that one can build something successfully without having a full understanding of how the building takes place, is one of the fundamental ideas behind Genetic Programming (GP). This chapter begins by justifying the need for something like Genetic Programming, and then proceeds to explain what Genetic Programming is. Finally a detailed description follows on

how GP can be applied to building (simulated) embryos in two and three dimensions.

2. Future Computing Technologies

One of the interests of the author is studying the characteristics of future computing technologies and then thinking about the conceptual problems these technolgies will pose. A brief summary of 5 of these technologies will be given and then a discussion of some of the problems and challenges they will raise. These 5 technologies are WSI (Wafer Scale Integration), Optical Computing, Molecular Electronics, Nanotechnology, and Quantum Computing. The feature which all these technologies have in common is that they will allow computers to be built with an enormous number of components. Several of them will allow an Avodagro number of components. Quantum Computing could even produce ten to the power 30 components.

Wafer Scale Integration (WSI) is the art of putting one huge integrated circuit on a single silicon wafer or slice. The state of the art at the time of writing is a surface area of several inches square. Estimates are that by the middle of the 90s, it will be possible using ULSI (ultra large scale integration i.e. submicron line widths) to put roughly 10 million artificial neurons on a WSI wafer. Optical Computing offers the possibility of performing computation entirely optically, using optical nonlinear effects to perform light amplification (thus mimicking the transistor). Three dimensional optical holographic memories offer the possibility of storing bits at a far higher density than possible today with silicon techniques. Molecular Electronics (ME) is the rapidly developing field which recognizes that there are limits to the downscaling of microelectronics which when met will introduce quantum effects which will interfere with the conventional action of the transistor. The alternative offered by Molecular Electronics is to explore the possibility that molecules can perform computation. Reading ME papers gives one the impression of studying organic chemistry. If ME succeeds, computers based on such devices will obviously contain an Avagadro number of components. Nanotechnology is even more ambitious than ME. Its goal is nothing short of "mechanical chemistry", i.e. building molecular scale assemblers or robots which can build molecular structures an atom at a time. These assemblers will be able to build virtually any substance and will revolutionise the 21st century in all respects. Assemblers will be capable of building copies of themselves and hence create emponential numbers needed to build macro-scale substances. We know that "nanotech" is possible because nature has already done it with its molecular biological robosome assemblers and their coded instructions to build virtually any substance as found in DNA. Finally, the theoretical physicists the last few years have been studying the possibility of Quantum Computing, i.e. using quantum level entities to perform computation in a characteristic quantum mechanical way.

All these wonderful technologies, most of which humanity will possess within a human generation, pose at least two major issues as far as Genetic Programming is concerned. They are the need for self assembly, and the need for evolutionary techniques to be applied to what is called here, the "complexity problem". When one is considering technologies with an Avogadro number of components, it will obviously be impossible to hand craft them all and put them together into an integrated whole. They will somehow have to build themselves, the way biological creatures do. Hence some form of artificial embryology becomes essential. When one is confronted with the challenge of building an "Avogadro"

machine, i.e. one with an Avogadro number of components, one has at least two choices in terms of how complex the machine is to be. One can construct it with a huge number of identical copies of a limited number of component designs which are simple enough for human beings to understand how they are built, or one can let the machine self assemble in a manner too complex for human beings to understand how the resulting whole functions. At first sight it seems that the second possibility would not be possible nor interesting, because what is the point of attempting to build such a machine.

These future technologies will allow the possibility of both approaches. What is interesting is that it may still be worthwhile to build hypercomplex systems even though we do not know how they function because their dynamics or interactions are so massively complicated and well beyond what the human brain can cope with in theoretical terms. Traditionally speaking, to build something, engineers first make blueprints of their devices. This implies they understand (more or less) how their planned devices will function. In other words, what is buildable and what is (more or less) understandable are often (but not always) identical. An alternative is the empirical hit or miss approach, which uses trial and error, e.g. drug testing, which takes a large number of possible drugs and tests whether they work. Those which do are usually marketed.

However, if one is to allow hypercomplex systems to self assemble, how can one be sure that the system which has self assembled has done so into something which is functional or useful in some sense? What does useful mean in this context? As will become clear in this chapter, using GP applied to the "growth" of artificial embryos is both self assembling and satisfies demands that the self assembled system is useful i.e. that it performs well according to some performance criterion. Genetic Programming is similar to the trial and error approach. Essentiallly it is a form of simulated or artificial evolution. It is a way of building hypercomplex systems such as (simulated) artificial nervous systems and (as will be described in this chapter), artificial embryos. GP allows one to "extend the barrier of the buildable" by placing a tool in ones hands which enables one to create useful devices which function, but nevertheless are (usually) too complex to be analysable. The application of GP to the construction of artificial nervous systems has already been described in another chapter similar to this one, also published by Wiley [4]. This chapter extends the discussion of this earlier chapter by showing how the same principles can be used to build artificial embryos. In other words, GP can be used as a first step to creating self assembling systems of great complexity yet with a desired functionality.

Before beginning a description of how this is done, and to give this chapter a reasonable level of self containment, a brief description of the basic principles of GP is presented next.

3. Genetic Programming

Genetic Programming is "applied evolution" i.e. using a form of artificial or simulated evolution called the Genetic Algorithm (GA) [5,6] to build/evolve hyper-complex systems. This chapter will give a clear example of this process when it is applied to growing artificial desired shapes in two and three dimensions. In order to understand GP then, one needs to understand what the Genetic Algorithm is. This section is largely devoted to giving a description of the essential features of the Genetic Algorithm.

The Genetic Algorithm (GA), as has already been mentioned, is a form of artificial evolution. But evolution of what? There are a few key concepts which need to be introduced to be able to understand how the GA can be employed to evolve such things as shapes. Probably the most important idea is that of a linear mapping of the structure of the system one is dealing with, into a string of symbols, usually a binary string. For example, imagine you were trying to find the numerical values of 100 parameters used to specify some complicated nonlinear control process. You could convert these values into binary form and concatenate the results into a long binary string. Alternatively, you could do the reverse and take an "arbitrary" binary string (of suitable length) and extract (or map) the 100 numerical values from it.

Imagine forming a population of 50 of these binary strings, simply by using a random number generator to decide the binary value at each bit position. Each of these bit strings (of appropriate length to agree with the numerical representation chosen) will specify the values of the 100 parameters. Imagine now that you have a means of measuring the quality or performance level (or "fitness", in Darwinian terms) of your (control) process. The next step in the discussion is to actually measure the performance levels of each process as specified by its corresponding bit string. These bit strings are called "chromosomes" by GA specialists, for reasons which will soon become apparent. Let us now reproduce these chromosomes in proportion to their fitnesses, i.e. chromosomes with high scores relative to the other members of the population will reproduce themselves more.

If we now assume that the initial population of chromosomes is to be replaced by their offspring, and that the population size is fixed, then we will inevitably have a competition for survival of the chromosomes amongst themselves into the next generation. Poor fitness chromosomes may not even survive. We have a Darwinian "Survival of the Fittest" situation. To increase the closeness of the analogy between biological chromosomes and our binary chromosomes, let us introduce what the GA specialists call "genetic operators" such as mutation, crossover etc. and apply them with appropriate probabilities to the offspring.

Mutation in this case, is simply flipping a bit, with a given (usually very small) probability. Crossover is (usually) taking two chromosomes, cutting them both at (the same) two positions along the length of the bit string, and then exchanging the two portions thus cut out, as shown in FIG. 1

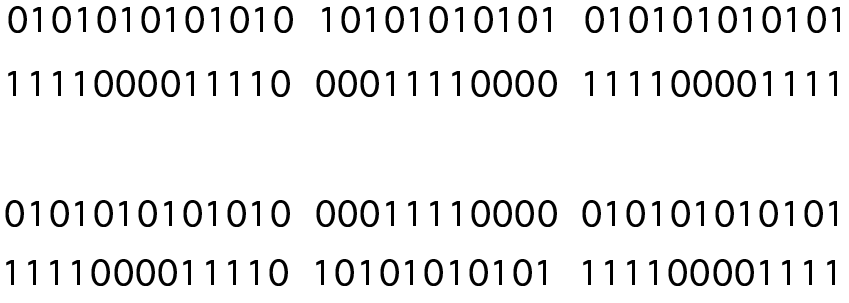


FIG.1 CROSSOVER OF CHROMOSOMAL PAIRS

If by chance, the application of one or more of these genetic operators causes the fitness (a scalar) of the resulting chromosome to be slightly superior to the average fitness of the other members of the population, it will have more offspring in the following generation. Over many generations, the average fitness of the population increases. This increase in fitness is due to favourable mutations and the possible combination of more than one favourable mutation into a single offspring due to the effect of crossover. Crossover is synonymous with sex. Biological evolution hit on the efficacy of sex very early in its history. With crossover, two separate favourable mutations, one in each of two parents, can be combined in the same chromosome of the offspring, thus accelerating significantly the adaptation rate (which is vital for species survival when environments change rapidly).

In order to understand some of the details mentioned in later sections, some further GA concepts need to be introduced. One such concept concerns the techniques used to select the number of offspring for each member of the current generation. One of the simplest techniques is called "roulette wheel", which slices up a pie (or roulette wheel) with as many sectors as members of the population (assumed fixed over the whole evolution). The angle subtended by the sector (slice of pie) is proportional to the fitness value. High fitness chromosomes will have slices of pie with large sector angles. To select the next generation (before the genetic operators are applied), one simply "spins" the roulette wheel a number of times equal to the number of chromosomes in the population. If the roulette "ball" lands in a given sector (as simulated with a random number generator), the corresponding chromosome is selected. Since the probability of landing in a given sector is proportional to its angle (and hence fitness), the fitter chromosomes are more likely to survive into the next generation, hence the Darwinian "Survival of the Fittest", and hence the very title of the algorithm, the Genetic Algorithm.

Another concept is called scaling, which is used to linearly transform the fitness scores before selection of the next generation. This is often done for two reasons. One is to avoid premature convergence. Imagine that one chromosome had a fitness value well above the others in the early generations. It would quickly squeeze out the other members of the population if the fitness scores were not transformed. So choose a scaling such that the largest (scaled) fitness is some small factor (e.g. 2) times the average (scaled) fitness. Similarly, if there were no scaling, towards the end of the evolution the average (unscaled) fitness value often approaches the largest (unscaled) fitness, because the GA is hitting up against the natural (local) "optimum behaviour" of the system being evolved. If the fitness scores differ by too little, no real selection pressure remains, and the search for superior chromosomes becomes increasingly random. To continue the selective pressure, the increasingly small differences between the unscaled average fitness and the largest fitness are magnified by using the same linear scaling employed to prevent premature convergence. This technique is used when evolving GenNets.

There are many forms of crossover in the GA literature. One fairly simple form is as follows. Randomly pair off the chromosomes selected for the next generation and apply crossover to each pair with a user specified probability (i.e. the "crossprob" value, usually about 0.6). If (so-called "two point") crossover is to be applied, each chromosome is cut at two random positions (as in FIG. 1) and the inner portions of the two chromosomes swapped.

Another form of crossover is called "Uniform Crossover", which works as follows. Randomly pair off the chromosomes in the population. Apply uniform

crossover to these pairs (with a user specified probability, typically about 60%), by creating an offspring pair to replace the parent pair in the next generation. To do this, for each bit position in the chromosome, select one of the two parents (with a user specified probability, typically 50%), and put the selected parent's bit in the corresponding bit position of offspring 1. In offspring 2 put the bit of the parent not selected. Empirical studies have shown that uniform crossover often works well with small populations of chromosomes and for simpler problems [17].

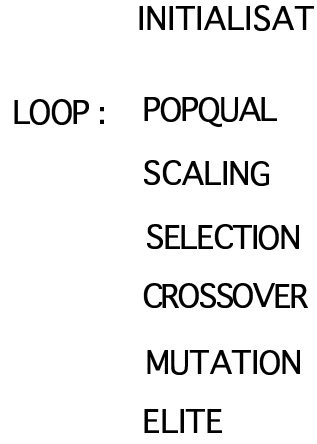


FIG. 2 TYPICAL CYCLE OF THE GENETIC ALGORITHM

A fairly typical cycle of the GA algorithm is shown in FIG. 2, where "Initialization" usually means the creation of random binary strings. "Popqual" translates the binary strings into the structures whose performances are to be measured, and measures the fitnesses of these performances. Often the chromosome with the highest fitness is saved at this point, to be inserted directly into the next generation once the genetic operators have been applied. "Scaling" linearly scales the fitness values before selection occurs. "Selection" is using a technique such as "roulette wheel" to select the next generation from the present generation of chromosomes. "Crossover" swaps over portions (or bits, in the case of uniform crossover) of chromosome pairs. "Mutation" flips the bits of each chromosome with a low probability (typically 0.001 or so) per bit. "Elite" simply inserts the best chromosome met so far (as stored in the "Popqual" phase) directly into the next generation. This can be useful, because the stochastic nature of the GA can sometimes eliminate high fitness chromosomes. The "elitist" strategy prevents this.

The above sketch of the GA is far from complete. However it contains the essence of the subject. For a more detailed analysis, particularly a mathematical justification as to why crossover works so well, see the several texts available [7], especially [8]. The GA is a flourishing research domain in its own right, and is as much a mix of empirical exploration and mathematical analysis as are neural networks.

Now that the essential principles of the GA have been presented, a description as to how the GA can be used to evolve shapes (embryos) can now follow.

4. Genetic Programming (GP) of 2D Shapes

Earlier papers [1,2,3,4] defined Genetic Programming (GP) as a programming methodology which used the Genetic Algorithm to evolve neural network modules. However, the ideas of this section show that the GP approach is quite general. For example, it can be applied to the evolution of artificial embryos.

This section is concerned with how one might create a new speciality called "Artificial Embryology". Later, as this subject develops, it is likely that techniques similar to those proposed below, will be used to "grow" (neuro-) electronic circuits and to test their functional properties. Such a possibility will be important as was explained in section 2. As a first step towards building something which both self assembles and which can have its fitness measured will be the growth of arbitrary 2D and later 3D shapes. The idea is to begin with a small number of "cells" which reproduce until some final shape is formed. The reproduction is obviously the self assembly and the fitness can be some function of the final shape. These ideas will now be described in greater detail. In this section, we begin by describing an algorithm to “grow” (evolve) arbitrary 2D shapes. The following section will do the same for 3D shapes (embryos).

The principal idea used in the experiments described in this chapter (when evolving 2D or 3D shapes) was to use the GA to evolve the so-called "neighbourhood interaction rules" of cellular automata (CA). Remember that the state of a CA cell in the next iteration of some cyclical CA algorithm depends upon the states of the neighbouring cells in the present iteration (e.g. the famous game of "Life"). Cells in a 2D (square grid) matrix may remain intact, reproduce, or die. Rules (which apply to all cells) are provided for these behaviours. Imagine a grid, partially filled with a contiguous colony of cells (i.e. the colony has no holes) and that a particular cell is to reproduce. Where will this cell put its offspring if there is no (immediate) space for it in the colony? Let us assume that the parent cell pushes all the cells in a given column or row of the grid in a given direction (i.e. one of N,E,W,S) and that the internal state of the parent cell decides in which direction. We assume that there is a finite number of reproductive iterations of the colony of cells. Hence with such rules, there will be a final shape whose fitness can be measured.

By introducing a large number of degrees of freedom for the GA to play with, one can imagine that a huge variety of possible final shapes can be made. The aim of the exercise then is to "grow" a final shape as similar as possible to some predefined form, e.g. a diamond, a heart, or irregular shapes such as amoebas. One could define the fitness to be simply how closely the final shape resembles the desired shape. Expressed mathematically, this could be as trivial as the formula below.

$$\text{fitness} = (\#in - 0.5*\#out)/(\#des)$$

where :

#in = the number of filled cells inside the desired shape
#out = the number of filled cells outside the desired shape

#des = the number of cells inside the desired shape

If one can grow one shape, one can grow many. If one can grow a body shape (e.g. an ellipsoid), why not limbs, a head, then fingers and toes. It would be nice if new shapes could be grown from the surface cells of old shapes in a time sequential manner. The GA "chromosome" which codes all this growth could become "differentiable", i.e. portions of the chromosome could become active and grow a certain kind of shape. Some of the cells in these shapes could transfer "markers" which if present in a cell with an appropriate state, could switch on a new portion of the chromosome (which was previously inactive) and become the germ cell for a new shape. *It was thought that using these ideas it would be possible to grow an embryo in a 3D grid.*

The above few paragraphs give a quick overview of the essential ideas for evolving shapes. The next few paragraphs provide more detail. The algorithm implemented was as follows. Look at FIG. 3 which shows an array of contiguous cells. Each cell contains 4 numbers, the so-called "Cellular NEWS Numbers (CNNs)", from which one derives the "NEWS Product", which is the product of these four numbers. The N number (for example) of a cell is its position in the column of cells from the top i.e. N(orth). A cell with N = 5, would have 4 cells above it. If its W number was 5, it would have 4 cells to the left of it, etc. These NEWS numbers tell the cell something about its position relative to other cells in the "colony".

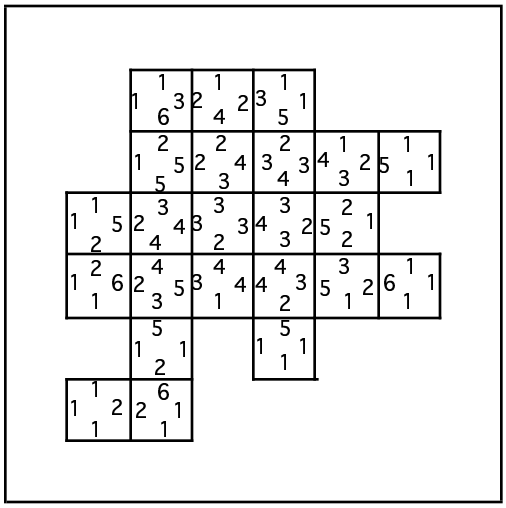
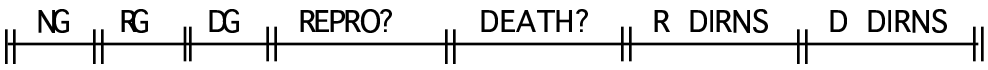


FIG. 3 AN ARRAY OF CELLS AND THEIR "NEWS" NUMBERS

Consider now the rankings of these NEWS numbers, rather than their absolute values. If all 4 numbers of a cell are different, there are 4! = 24 ways these four rankings (1 2 3 4) can be placed around the four NEWS positions. If two (or twice two) of these numbers are the same, there are 42 ways their rankings can be positioned. If three are the same, there are 8 ways. If four are the same, there is only 1 way. Hence there are 75 ways altogether. We can use these 75 possibilities and define them to be the states of the cells. Each cell will have one of these 75 states. For each of these 75 different positioned rankings, there are corresponding

bits in the chromosome (used to control the growth of the colony), which specify whether a cell with a given ranking pattern (and hence state) will reproduce and/or die.

FIG.4 shows the various fields of the bit-string chromosome used to evolve the rules of reproduction and death of cells in the colony. We will discuss these systematically from left to right. NG is the number of generations that the colony is to evolve over (i.e. the number of loops in the algorithm shown in FIG. 5). The maximum number of generations (Gmax) is a user given system parameter, and determines the number of bit positions in this field. The RG field is used to specify whether reproduction can occur in a given round (loop) of the embryo algorithm. The length of the RG (and the DG) field is usually several times Gmax bits long. The "ith" bit in the RG field is set if reproduction is allowed in the "ith" loop. The "jth" bit in the DG field is set if death is allowed in the "jth" loop. It is possible that in a given round, both reproduction and death occur, or neither, or one or the other. These four possibilities give the embryo algorithm more flexibility.



NG = NUMBER OF GENERATIONS
RG = WHICH GENERATIONS CAN REPRODUCE
DG = WHICH GENERATIONS CAN DIE
REPRO? = WHICH RANKING PATTERNS CAN REPRODUCE
DEATH? = WHICH RANKING PATTERNS CAN DIE
R DIRNS = REPRODUCTION DIRECTION FOR EACH RANKING PATTERN
D DIRNS = DEATH DIRECTION FOR EACH RANKING PATTERN

FIG. 4 FORMAT OF THE "EMBRYO" CHROMOSOME

The REPRO? and DEATH? fields both consist of 75 bits. The 75 ranking patterns are ordered from 1 to 75. For a cell to reproduce, not only must reproduction be allowed for that round, it also has to satisfy the condition that if its ranking pattern number is "R", then the "Rth" bit in the REPRO? field has to be set to 1 (i.e. cells with ranking pattern "R" are to reproduce, so long as reproduction is allowed for the round concerned). A similar story occurs for the DEATH? field, (i.e. cells with ranking patterns "D" are to die, so long as death is allowed for the round concerned). These two fields allow reproduction and death to counteract each other in their influence on the shape of the colony.

If a cell can reproduce, it has to create a space for the offspring cell. Since a cell colony is (mostly) contiguous, a space is created by pushing all the cells lying to the (N,E,W,S) of the parent cell in one of four directions (N,E,W,S) by one square on the grid. The fields R DIRNS and D DIRNS both contain 75 pairs of bits. If a cell with ranking pattern "R" is to reproduce, then the direction (one of N,E,W,S) is specified by the "Rth" pair of bits in the R DIRNS field. A similar story holds for a cell with ranking pattern "R" which is to die, using the D DIRNS field. When a cell dies, all the cells to (one of the) N,E,W,S of it move one square to fill the vacancy.

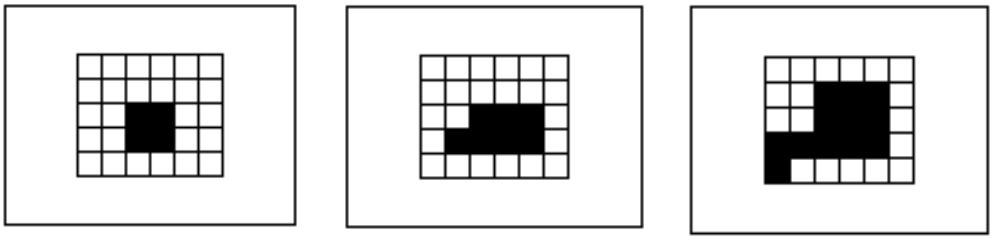
Since the final shape of the colony depends on the order in which cells reproduce and die, this order has to be specified. In each loop (or round) of the algorithm, all the reproductions occur first, then the deaths. At the beginning of each round, after the NEWS numbers of all the cells have been found, and before there are any cell modifications, each cell records its NEWS numbers, its NEWS product and whether it is to reproduce and/or die. The cells reproduce in decreasing order of their NEWS product. If NEWS products are sometimes equal for two or more cells, then the average of the NEWS products of the cell and its 8 nearest neighbours can be used in decreasing order. Offspring cells cannot die in the same round (i.e. the "I am a cell to die in this round" marker is not passed on to the offspring cell). FIG. 5 summarizes the embryo algorithm. Since the number of loops is finite, there will be a final shape of the colony. By mutating (flipping) bits in the chromosome and using (uniform) crossover, the final shapes evolve to being very close to the desired shapes.

EMBRYO LOOP :

* CALCULATE CELLULAR NEWS NUMBERS (CNNs) OF EACH CELL
* FROM CNNs CALCULATE RANKING PATTERN NUMBER (RPN) OF EACH CELL
* USE RANKING PATTERN NUMBER TO SEE IF EACH CELL REPRODUCES OR DIES
* FIND REPRODUCTION AND DEATH DIRECTIONS FOR SELECTED CELLS
* PERFORM REPRODUCTIONS, UPDATING GENERATION COUNTS
* PERFORM DEATHS

FIG. 5 PRIMARY LOOP OF EMBRYO ALGORITHM

FIGs. 6, 7, 8, 9 present results of the 2D Embryo Algorithm.The target shapes are indicated by the grid lines. Each frame shows the results at the end of each loop. The quality or fitness in shown in the figures. These initial experiments showed that the basic idea of being able to grow shapes was feasible, at least for convex shapes such as the rectangle, square, triangle shown below. The algorithm failed however for non convex shapes such as an arch shown in FIG. 9 In the next section will be described more ambitious experiments performed on a larger scale and in both 2 and 3 dimensions.



Start 3 Births 5 Births

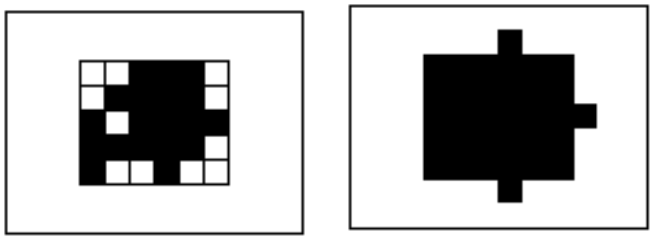
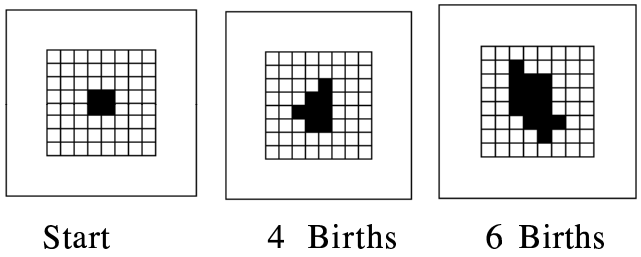

7 Births
14 Births (Q = 95%)

FIG. 6


Start
4 Births
6 Births

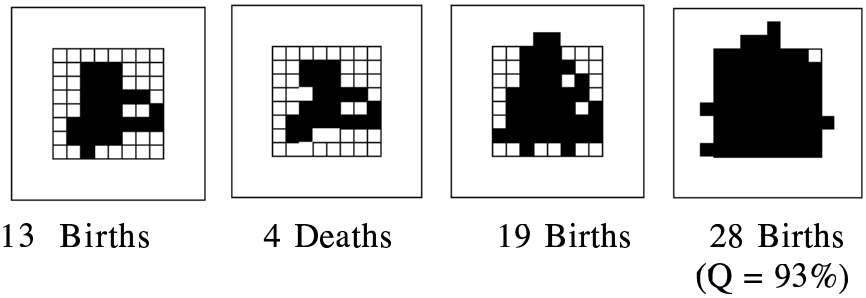
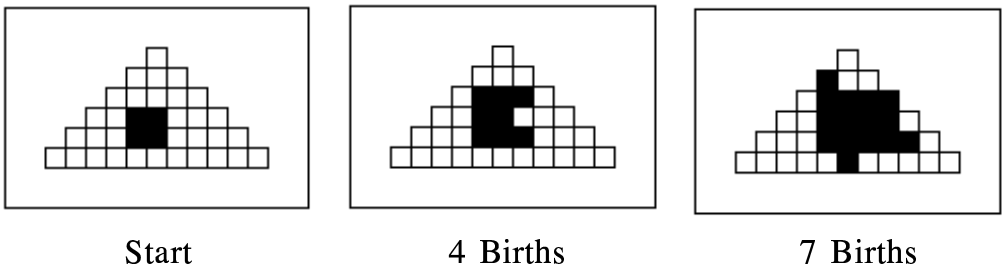
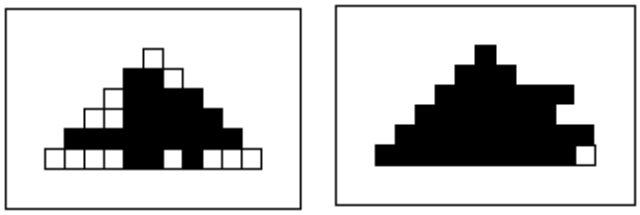

13 Births
4 Deaths
19 Births
28 Births
(Q = 93%)

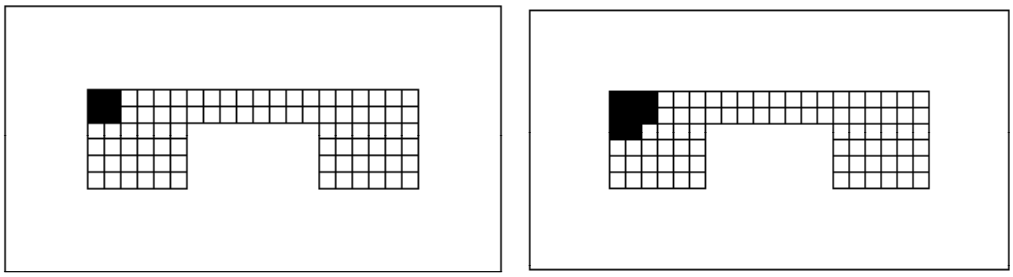
FIG. 7


Start
4 Births
7 Births

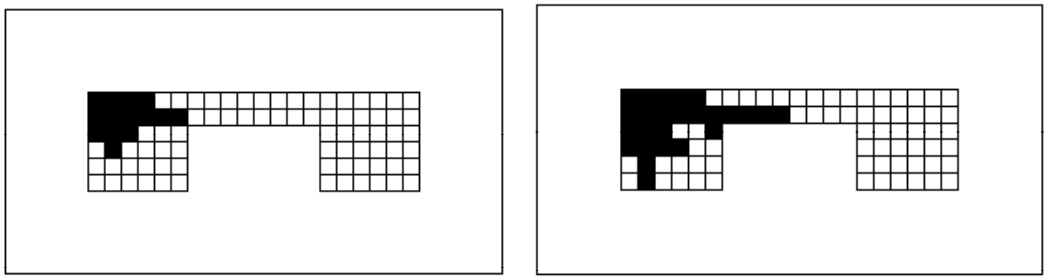


8 Births 15 Births (Q = 93%)

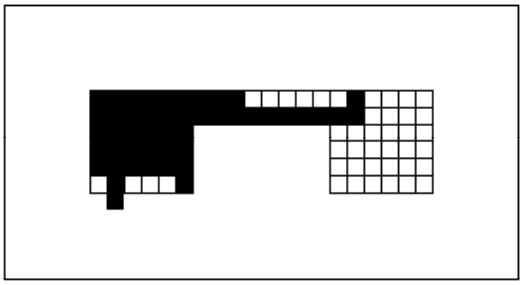
FIG. 8



Start 4 Births



6 Births 11 Births



22 Births (Q = bad)

FIG. 9

5. 2D and 3D Shapes on the Connection Machine

After the encouraging results of the above experiments on a MAC II, it was considered interesting by the author to see whether more ambitious shapes could be "grown" in 2D and 3D on a substantial computer such as a Connection Machine (a massively parallel machine with up to 65000 little processors all executing the same instruction simultaneously but each with its own data - a "data parallel" machine). With this ambition in mind, the author went to Boston to put a modifed version of the above embryo algorithm onto a Connection Machine. This section presents some of the initial results (initial because work continues).

If one observes FIGs 6,7, and 8 above, it is clear that any deaths which occurred, seemed to have played a rather minor role, if at all. To simplify matters, it was decided that deaths could be eliminated from the algorithm without undue loss. To simplify matters further, it was also decided that only edge or surface cells would reproduce, thus avoiding the need to displace whole rows or columns of cells to make room for offspring cells. Having decided that only edge (in 2D) or surface (in 3D) cells will reproduce allows one to simplify the definition of the state of a cell.

The state of a cell in the new algorithm was defined from the positions of its neighbourless sides. For example, for a 2D cell, only 4 different states are possible if 3 sides are neighbourless (i.e. equivalently, there are only 4 ways of positioning the neighboured side). There were 14 such states (4C1+4C2+4C3 = 4+6+4 = 14) as shown in FIG. 10. Neighbourless sides are indicated with a 0, neighboured sides with a 1. Any 2D edge cell must be in one of these 14 states. Its is assumed that cells are always have at least one neighbour, i.e. there are no island cells. With these simplifications, the number of possible states for the 2D cells has been reduced from 75 to 14. At first sight it appears as though the second algorithm is less flexible, that there is less scope for the GA to play with. That this need not be so, is explained below.

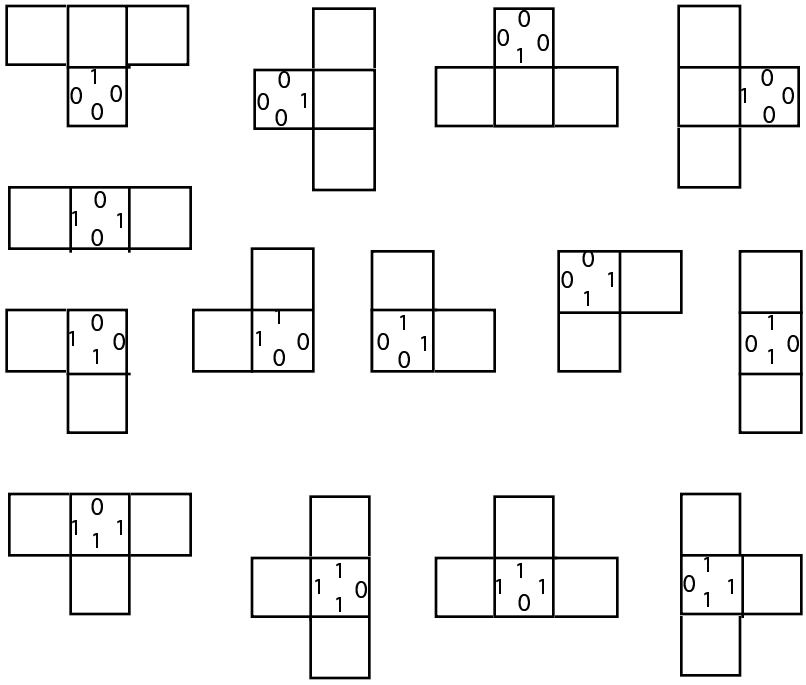


FIG. 10 The 14 Neighbourless Patterns for 2D Cells

The above assumptions thus allow the "embryo" chromosome of FIG. 4 to be simplified as shown in FIG. 11 Since there are no deaths, the fields DG, DEATH? and D DIRNS can be dropped. With no DG field, it makes no sense to have an RG field. Instead we have reproduction allowed for every iteration. All that remains therefore are the NG, REPRO? and R DIRNS fields. The REPRO? and R DIRNS fields are redefined as follows. Since there are now 14 states, 14 bits are needed in the REPRO? field to specify which edge cells will reproduce. The R DIRNS field happens also to be 14 bits long. This figure was derived from the following considerations. If an edge cell has only one neighbourless side, then it can only reproduce in that direction, so no information is needed to specify the choice of direction. For a cell with two neighbourless sides, one bit is needed to specify the choice. There are 6 states with 2 neighbourless sides, hence 6 bits are needed for those states. For a cell with three neighbourless sides, 2 bits are needed to make the choice of direction of reproduction. However 2 bits can specify four choices, but only three are needed. In practice, if the fourth choice is specified, i.e. a forbidden direction (towards a neighboured side) is chosen, then the reproduction is simply ignored. There are 4 states with three neighbourless sides, hence 8 bits are needed for them. The total length of the R DIRNS field is thus 6 + 8 = 14 bits. The total length of the REPRO? and R DIRNS fields is only 28 bits, a very short chromosome.

NI = NUMBER OF ITERATIONS
REPRO1? = WHICH STATES CAN REPRODUCE IN ITERATION 1
R DIRNS1 = REPRODUCTION DIRECTIONS IN ITERATION 1
REPRO2? = WHICH STATES CAN REPRODUCE IN ITERATION 2
R DIRNS2 = REPRODUCTION DIRECTIONS IN ITERATION 2

FIG. 11 FORMAT OF THE SIMPLIFIED "EMBRYO" CHROMOSOME

This economy suggested the possibility that one might have a REPRO? and R DIRNS field pair for each iteration. The ith iteration would thus take its reproduction instructions from the ith pair of fields in the chromosome. This independence of instructions for each iteration allows several possibilities. One is that intermediate target shapes become possible. A specific shape S1 can be evolved in the first I1 iterations. The resulting chromosome can then be frozen and used as an initial component in a longer chromosome which codes for I2 (> I1) iterations which expands the shape S1 into S2. The same process can be continued from S2 to S3 etc. Another obvious possibility is that with a larger chromosome, more information is specified, hence a greater range of possible shapes should be evolvable.

FIGs. 12, 13 and 14 show the results of using the "simplified" embryo algorithm to evolve the convex shapes of a circle, ellipse and a rectangle. The fitness definition was unchanged. The fitness qualities were 99% for the circle, and 96% for the ellipse and rectangle. FIG. 15 shows an attempt at evolving a triangle

which failed, showing that sometimes an evolution can get stuck in an intermediate shape that it cannot get out of. For example, consider the sawtooth NW-SE edges at the upper right and lower left of the figure. Once such a sawtooth pattern emerges, the evolution only repeats it. FIG. 16 shows a more successful attempt at triangle evolution where only the mutation rate was changed. However in this case, there were not enough prespecified iterations in the chromosome to cover the target shape. The triangular pattern was evolving well enough, but ran out steam. Its fitness value was 92%. FIG. 17 shows the result of an attempt to evolve a non convex shape in the form of the letter L. At the bottom and left of the figure, one can see the evolution was successful, but that it failed (again) to "turn corners". A "webbed foot" phenomenon occurred, filling the space contained between the three corners of the L. The evolution of nonconvex shapes thus remains a challenge requiring additional effort.

We turn now to the 3D case. For 3D cells (cubes), with 6 faces (east, north, west, south, up and down), there were 62 states (6C1 + 6C2 + 6C3 + 6C4 + 6C5) = (6 + 15 + 20 + 15 + 6) = 62. Thus the REPRO? field for the 3D case contained 62 bits. The length of the R DIRNS field was derived from the following considerations. If there is only one neighbourless face, there is only one direction of reproduction, hence no information is needed to specify any choice. If there are two neighbourless faces, 1 bit is needed per state, and there are 6C2 = 15 such states, hence 15 bits are needed. If there are three neighbourless faces, 2 bits are needed per state and there are 6C3 = 20 such states, hence 40 bits are needed. If the chromosome specified the fourth possible reproduction direction, it was ignored. If there are 4 neighbourless faces, again 2 bits are needed per state, and there are 6C4 = 15 such states, hence 30 bits. If there are 5 neighbourless faces, 3 bits are needed per state, and there are 6C5 = 6 such states. Three of the eight possible "directions" specifiable by the chromosome were simply ignored. The total length of the R DIRNS field was thus (1*15 + 2*20 + 2*15 + 3*6) = (15 + 40 + 30 + 18) = 103 bits.

When the author set out for Boston to work on a Connection Machine, family constraints imposed a two month deadline. Initially it was thought that this would be enough time to evolve the 3D embryo shape as was originally planned. As will become evident, this objective was only partially achieved. The realities of having to learn a new computer language (called C*, pronounced "C star" - a parallised version of C for the Connection Machine), a new editor, a new debugger, a whole new system and the delays of minutes-long compilations on time shared front-end work stations which were also used to feed parallel instructions to the Connection Machine, meant that much time was lost. In practice, all that was possible in the mad rush of the last few days was to evolve a sort of tadpole shape, where one shape was "grown" out of another. It was reasoned that if this could be done once, it could be done several times. For example, one could grow an ellipsoid body, and then from it grow a head, and four limbs, thus creating an embryo shape. At least the tadpole shape showed that the principle of growing one shape out of another was successful.

For the 3D "embryo", the target shape was a sphere (the head) which was grown first and then its cells were frozen (i.e. they could no longer reproduce). FIGs 18 and 19 show its "embryological" development at 300 cells (half way) and 600 cells. Contiguous and overlapping slightly with the sphere target shape was an ellipsoid target shape, which was used to grow a “limb bud” or beginnings of a tail, thus producing a sort of tadpole shape. Those actual cells which were common to both target shapes (the "neck") were activated artificially and allowed to reproduce

into the second target shape. With more time it would have been possible to grow limbs this way out of a body shape and produce a human baby like embryo.

4. Remarks and Future Ideas :

The above experiments show that a lot of work still needs to be done in order to be able to grow truly arbitrary shapes. The two algorithms presented above show that non convex shapes remain a problem. The growth of the 3D embryo shape still has certain ad-hoc elements to it. For example, to get the tail to grow, it was first necessary to freeze the cells of the head, and then to unfreeze those cells at the neck which could grow into the tail. It would be nice if whole embryos could be grown in "one hit" so to speak, without such arbitrary intervention. A tentative suggestion as to how this might be done, could ber as follows. Imagine one wanted to grow the L shape. This means that cells must somehow be capable of "turning corners". One way of doing this might be to use a "growth substance", assumed to be emitted by some cells at a certain moment (i.e. at a given iteration). This substance could diffuse to its neighbours allowing them to reproduce. Those cells not within range of such a growth substance cannot reproduce. Hence if the growth substance source cell were near the bottom of the vertical portion of the L, the cells in the upper portion would not reproduce. Only those cells near the bottom could turn. The Genetic Algorithm could be used to decide which cells emit the growth substance and when.

Acknowledgments

The author would like to express his heartfelt gratitude to the following people at Boston University - to Professor Richard Brower (of BU's Center for Computational Science) for giving the author permission to work on BU's Connection Machine - to Robert Putnam, (Thinking Machines Corporation's Application Engineer for BU) for tolerating the author's constant pestering with technical questions to accelerate the learning curve, and to Tim Hall and Laura Giannitrapani (of BU's Graphics Lab) who made the video and stills.

References

[1] "Genetic Programming : Modular Neural Evolution for Darwin Machines", H. de Garis, *Proceedings IJCNN90 WASH DC, International Joint Conference on Neural Networks*, January 1990, Washington DC.

[2] "Genetic Programming : Building Nanobrains with Genetically Programmed Neural Network Modules", H. de Garis, *Proceedings IJCNN90 SanDiego, International Joint Conference on Neural Networks*, June1990, SanDiego.

[3] "Genetic Programming : Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules", H. de Garis, *Proceedings 7th. Int. Conf. on Machine Learning*, Austin Texas, June 1990, Morgan Kaufmann, 1990.

[4] "Genetic Programming", H. de Garis, Chapter 15, in book, "*Neural and Intelligent Systems Integration*", ed. Prof. Branko Soucek, Wiley Interscience, 1991.

[5] "*Genetic Algorithms in Search, Optimization, and Machine Learning*", D.E. Goldberg, Addison-Wesley, 1989.

[6] "*Adaptation in Natural and Artificial Systems*", J. H. Holland, Ann Arbor, Univ. of Michigan Press, 1975.