Evolution of Neural Structures Based on Cellular Automata

Felix Gers **Hugo de Garis**
Brain Builder Group, Evolutionary Systems Department,
ATR Human Information Processing Laboratories,
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan.
Phone: +81-774-95-1079, Fax : +81-774-95-1008,
email: flx@hip.atr.co.jp & degaris@hip.atr.co.jp
URL : http://www.hip.atr.co.jp/~flx
& http://www.hip.atr.co.jp/~degaris

Since modern electronics allows, or will soon allow, brain-like systems to be built with millions and even billions of artificial neurons, it is necessary to add 'evolutionary engineering' techniques to traditional top-down, 'blue-print', engineering. This chapter shows how this can be done, by stressing the importance of using Cellular Automata as a dominant computational paradigm. We also report on progress in the new field of 'Evolvable Hardware' (EHW), the brain child of the second author, which evolves systems directly in hardware, using such devices as FPGA's (Field Programmable Gate Arrays). In order to use this new technology, we will have to develop purely parallel computing models with highly local interactions. It is precisely these criteria, namely local connections, pure parallelism, plus the prospect of evolvability, that are satisfied by using Cellular Automata in parallel hardware. This chapter introduces a new model for ATR's 'CAM-Brain Project', which is far more efficient and simpler than the older model. The CAM-Brain Project aims at building a billion neuron artificial brain using 'evolutionary engineering' technologies. This arificial brain will consist of large numbers (millions) of cellular automata based, neural network modules, which are evolved at electronic speeds in special FPGA based hardware, and then downloaded into gigabytes of RAM and run on CA accelerator hardware, such as MIT's 'CAM-8' machine.

Keywords

Artificial Brains, Evolutionary Engineering, Neural Networks, Genetic Algorithms, Genetic Encoding, Cellular Automata, Cellular Automata Machine (CAM-8), Evolvable Hardware.

1 Introduction

The aim of ATR's CAM-Brain Project is to build/grow/evolve brain-like-systems, into which human knowledge can be inserted as initial conditions. The brain-like structures are based on cellular automata (CA) and grow inside cellular automata machines (CAMs). Evolutionary engineering techniques, combined with huge computing power, may then be able to evolve such brain-

like systems with a level of functionality simular to that of biological brains. The expectation is that it will be much easier to investigate the behavior of engineering-based brains than biological brains, and that the engineering solutions will reveal insights into the working mechanisms of the evolved systems, which might then be rediscovered in biological brains.

In the CAM-Brain Project, the CAs act as the medium in which to conduct the growth, and in a second step, the neural signaling, of brain-like structures.

To apply evolutionary engineering techniques to the creation of huge neural systems, e.g. an artificial insect brain, we need a simulation tool that has to satisfy two conditions. First of all, it must be possible to simulate most of the important characteristics of biological neural networks. Secondly, the evolution times must be manageably short. The bottle-neck in evolutionary algorithms is the fitness measurement time, i.e. the time needed to simulate the systems. Hence it must be possible to simulate these systems on massively parallel hardware.

With CAs as a simulating tool, both demands can be met. Artificial neurons and neural networks can be modeled at any level of detail, because CA cells can symbolize a molecule in a biological cell membrane, a whole neuron, or any structure in between. Since CAs are only locally connected, they are ideal for implementation on purely parallel hardware. CAs are both structurally flexibile and inherently parallel. We use MIT's Cellular Automata Machine CAM-8 [1] for our CAM-Brain modeling.

2 Knowledge Combined with Evolution

In recent years, neuro-scientists have put a lot of effort into trying to understand and to reconstruct the basic functions of the brain. Almost all this research has followed a bottom up approach. For example, slices of the visual and auditory cortex have been extracted and investigated for their possible functionality in a larger context. The idea is that the many little insights will be amalgamated to reveal the functionality of larger and more complex parts, until finally the brain is reasonably well understood. The success of this approach depends on the functional composition of the brain and its major components. If it is possible to divide the brain into more or less independent compartments which are small enough for human researchers to comprehend their function, then this approach has, considering the large number of researchers working in the field, a good chance of succeeding.

But it is possible that many regions of the brain behave differently. Their behaviors may result from a sophisticated interaction between many components. In this case, it is not possible to extract a component and assign a

certain function to it. The behavior of a region has to be considered and understood as a whole.

To cover this latter case, we add evolutionary engineering to the first approach. Our aim is to create a brain-like-system into which human knowledge can be inserted as initial conditions. Evolutionary algorithms combined with huge computing power may then be able to evolve artificial brains with the demanded all-over functionality of biological brains. The expectation is that it will be much easier to investigate engineering-based brains than biological brains, and that the engineering solutions will reveal insights into the working mechanisms of the evolved system as a whole.

3 Evolvable Hardware (EHW or E-Hard)

This section gives a brief tutorial overview of the new field of Evolvable Hardware (EHW, or in conversation 'E-Hard'). E-Hard is a new technology which is of fundamental importance to the CAM-Brain Project, because electronic evolutionary speeds are critical, if one is to evolve millions of neural net modules in a reasonable time (e.g. 0.1 second per module, which our Brain Builder Group will be achieving by the summer of 1997 with our new 'CAM-Brain Machine'.

The original idea of EHW occured to de Garis in the summer of 1992. At the time of writing (November 1996), at least 5 groups in the world are working on EHW (2 in Europe, 2 in Japan and 1 in the US). The first workshop on EHW was held in Lausanne, Switzerland in the fall of 1995. The first international conference on EHW (and biologically inspired electronics) was held in Tsukuba, Japan, in October 1996.

What now follows is a introduction to the basic idea of E-Hard, and a short summary of the work of the major players. We begin with the basic idea. Now that some FPGAs are S-RAM (static RAM) based, they can be rewritten to an infinite number of times (like ordinary RAM). The essential idea of EHW, is to consider the software bit string instruction which configures (wires up) the circuit, as a genetic algorithm chromosome. By taking the output of the configured circuit and measuring its "fitness" (i.e. performance quality on some control task), over a population of circuits, one can evolve hardware. In other words, if the same hardware can be infinitely reprogrammed, it can be evolved. Today's FPGAs were not built with EHW in mind. Hemmi at ATR & NNT, and Higuchi at ETL, both in Japan, are designing evolvable chips which should be ready in 1997. Arguably the most appropriate FPGA family of chips out now for EHW is probably the Xilinx company's XC6200 chips. Thompson, at Sussex University, is working with the XC6216 chip. We are

working with the XC6216 and XC6264 chips.

There are two broad approaches to EHW:

1. Extrinsic
    - a.k.a. (= also known as) Offline
    - a.k.a. Simulated
    - a.k.a. Offchip
2. Intrinsic
    - a.k.a. Online
    - a.k.a. Real
    - a.k.a. Onchip

Extrinsic EHW simulates the evolution of a hardware circuit, obtains an elite solution, which is then downloaded ONCE into the real programmable circuit. Intrinsic EHW uses the programmable circuit for the evolution of each chromosome in the genetic algorithm. All the evolution occurs in the circuit. At first, EHW researchers did only extrinsic EHW. The first person to do intrinsic EHW (using the Xilinx XC6216 chip) was Thompson in early 1996. [14,15,16]. This family of chips has certain features which make it more suitable for EHW research. Thompson, was able to intrinsically evolve a simple frequency detector circuit, using the XC6216 chip. The evolution takes advantage of the analogue characteristics of the chip.

John Koza is evolving (extrinsically) analogue electronic circuits, using a modifed form of Gruau's "cellular encoding" technique, and measuring the fitnesses of the evolved circuits by simulating their behaviors using the giant electronic circuit simulation software called SPICE. [6,7]

Dr. T. Higuchi, ETL (Electrotechnical Lab), Tsukuba, Japan, was the first person in the world to begin implementing EHW [12,13](after having heard de Garis give a talk on the E-Hard idea to Higuchi's group in the summer of 1992). At first he worked with GAL chips, (extrinsically) evolving simple combinatorial and sequential digital circuits. In 1995 he has obtained a million dollar grant from MITI to develop an evolvable chip.

Hemmi, a colleague at ATR, has been doing extrinsic EHW by applying GP (Koza style genetic programming) to tree-like HDLs (hardware description languages). He evolved a sequential circuit to solve the "Muir Trail" ant-food problem [17,18,19]. His next project is to design an evolvable chip, probably at NTT, his research company, in 1997.

4 Using CAs as a Tool to Evolve Systems

To apply evolutionary technologies to a huge neural system we need a simulation tool that has to satisfy two conditions. First of all it must be possible to simulate most of the important characteristics of biological neural networks. Secondly, the evolution times must be manageably short. The bottle-neck of evolutionary algorithms is the fitness measurement time, i.e. the time needed to simulate the system. Hence it must be possible to simulate the system on massively parallel hardware.

With CA as a simulating tool, both demands can be met. Artificial neurons and neural networks can be modeled at any level of detail, because CA cells can symbolize a molecule in a biological cell membrane, a whole neuron or any structure in between. In our model shown in figure 1, a neuron body consists of a square $(11 * 11)$ of cells.

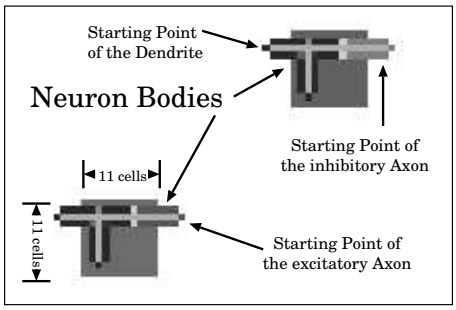

Figure 1: Our Neuron Model.

Changes to the model can be made easily, by adding a new feature or adding CA-states and additional rules. Since these model changes are hardware independent, they can be evolved by an evolutionary algorithm.

Since CAs are only locally connected, (our CAs for example use only the four nearest neighbors), they are ideal for implementation on pure parallel hardware. We use MIT's Cellular Automata Machine CAM-8[1] for our CAM-Brain modeling.

5 A New Model for ATR's Cellular Automata Based Artificial Brain Project

With the new CAM-Brain model that is introduced in this chapter, we have overcome many of the deficiencies of the old model. Nevertheless, the old

model did at least demonstrate the realizability of evolvable neural structures based on CAs [10].

The improvements of the new model are based on several ideas, plus the use of a higher level language to generate our CAs and transform them into an executable rule-base. The new model is implemented on MIT's CAM-8 machine, using a kind of CA that we call a "partitioned" CA. Partitioned CAs use a signal-based cell interaction, instead of exchanging complete cell-states. (See section 7.1 for details.) We introduce a new technique for the genetic encoding of neural structures, based on the idea of initially distributing the network's chromosome homogeneously over the whole CA-space. We have reduced the number of CA cell interactions to two when modeling the signaling of artificial neurons.

These features of the new CAM-Brain model can be consistently combined, so that the complete model, including growth, signaling and evolution, can be implemented in 2 bytes of memory per cell, and with a small number of logic gates. The simplifiction means that it may be possible to implement the new model directly in hardware.

After describing the old model, (the first work in the field of CA-based brain-like structures [10]), the following three sections explain the improvements of the new model in more detail.

6 The Old CAM-Brain Model

The old model was based on Codd [3], i.e. the CA trails representing the axons and dendrites were 3 cells wide, with 2(4) outer "sheath" cells (in the 2D(3D) version), and an inner cell which was used to transmit the growth signals in the growth phase, and the neural signals in the signaling phase. As will become clear from the perspective of the new model, the essential feature of the old model was that the state of each cell contained NO information concerning the direction of flow of signals. It was the insertion of such directional information into the new model which enabled a revolution in the simplicity and efficiency of its implementation. In the old model, the sheath cells guided the signal flow. At junctions in the signal path, gating cells (turn left, turn right) were employed to direct neural signals to their receiving neurons. Collisions between CA trails became complex, too complex for writing high level general rules to govern their dynamics. Hence new CA state transition rules had to be found empirically, i.e CAM-Brain would run until a missing rule was found. It would then prompt the human operator for the desired next state, store the new rule automatically into the rule-base, and continue running until the next missing rule. The old model of 3D CAM-Brain required nearly 4 million rules. This

large number of rules resulted from the 24 symmetry rotations of each 3D rule generated empirically. This empirical approach worked, but was slow. It took 2 years to generate all the 3D rules in the old model.

We now give a brief overview of some of the details of the old model. We do not spend a lot of time on this, because such details have already been presented in many publications, e.g. [10].

The CA based network of the old model consists of two types of neuron, excitatory and inhibitory. At first, neuron bodies are grown from an initial pattern, as shown in figure 2. The starting positions of the seed cells can be either user specified or placed under genetic control.

To each grown neuron a "chromosome" block of random growth instruc-

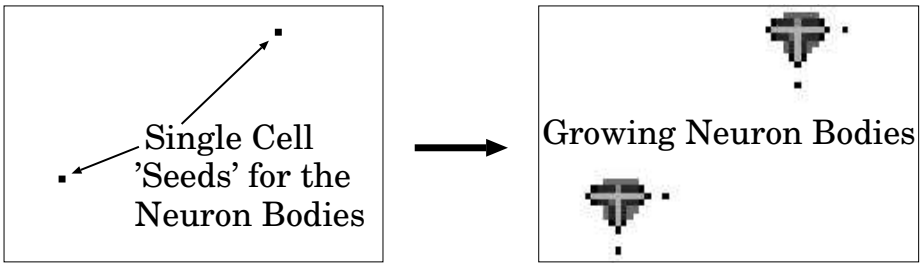

Figure 2: The growth of the neuron bodies (right) from an initial pattern (left).

tions is attached. These chromosome blocks are stored in a subspace "under" the actual CA-space as shown in figure 3.

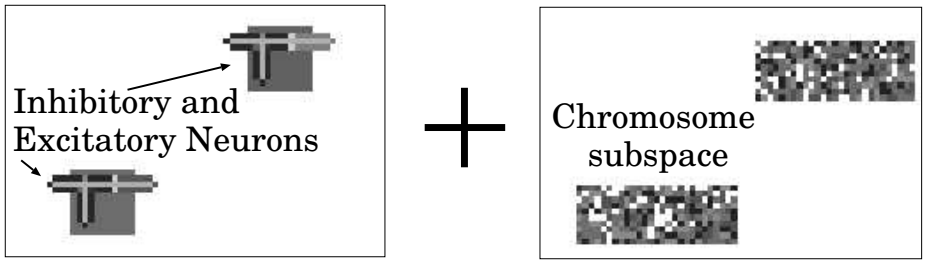

Figure 3: The neurons (left) with their individual chromosomes (right). A chromosome contains 6 different kinds of growth signals : grow straight, turn (left, right), split (left, right) and T-split. Every CA cell of a chromosome represents one growth signal.

With these growth instructions, each neuron grows an axonic and a dendritic tree, which later will be used to conduct neural signals. A collision of

a dendritic and an axonic branch results in the formation of a synapse. The synapses are not weighted. The weighting is expressed indirectly via the length of the dendritic trail that a signal has to pass through. A signal decreases linearly in strength while traveling along a dendrite, so the weighting can be adapted by evolving the length of the dendritic trail.

Each neuron grows its dendrite and axon with its individual chromosome information, until either all the chromosome information is used, or there is no more growth possible in the given CA-space. Alternatively, the chromosome can be wrapped around, so that the available space determines when growth stops. Figure 4 shows examples of early and saturated growths.

To measure the fitness of a grown circuit, signal values are extracted from user chosen points, and are used to control some process. The quality of the control becomes the fitness of the circuit. A conventional genetic algorithm is then applied to a population of circuits. de Garis has shown, the evolvability of the model (for details, see[10]).

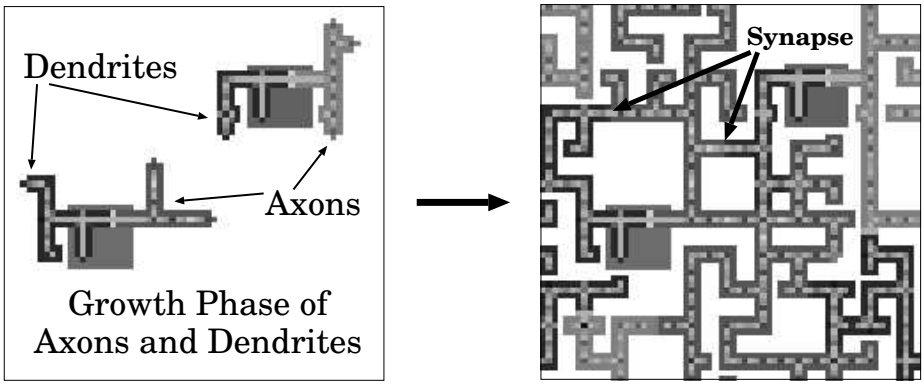

Figure 4: (Left) The growth phase of the network continues until all CA trails are connected through synapses or are blocked. (Right) A network, fully grown with the old CAM-Brain model.

7 The New CAM-Brain Model

The old CAM-Brain model is a pioneering work. It shows the realizability of evolvable neural networks based on CA and the possibility of combining evolved neural net modules to build artificial brains. As with most pioneering works, the old model opens new directions for research, but still contains deficiencies to be overcome. Some of the deficiencies of the old CAM-Brain model are

- inefficient use of CA-space, (based on Codd's model[3], and i.e. the use of sheath (and gating) cells),
- restrictions on growable structures of the axonic and dendritic trees, due to its "fractal" growth model,
- the large number of CA-states and CA-rules,
- low (cellular) level of CA rule generation rather than using high level "abstract" rules, (empirical, rather than automatic CA rule generation, which caused the slow rate of rule generation (i.e. 2 years instead of 20 seconds) and inflexibility with respect to changing the model),
- the necessity of symmetry rotations for each rule (4 rotations in 2D and 24 in 3D).

In the new model we overcome these problems and add ideas from the old model to the new. The next subsection (7.1) gives a general description of the CA-type we use. We introduce a CA type which differs somewhat from the more usual form of CA. We call this variant "partitioned CA". In the following three subsections (7.2 - 7.3) we describe how the deficiencies of the old model are overcome, and how these improvements are incorporated into the new model. We call this new CAM-Brain model the "CoDi-model" for neural signaling. We also show how it is possible to distribute the chromosome initially over the CA-space, an idea which has several advantages over the old model.

7.1 General Description of our CAs

When designing our CA-based neural networks for the new model, our objective is to implement them directly in evolvable hardware, e.g. FPGAs, content addressable memory chips, etc. Therefore, we try to keep the CAs as simple as possible, by having a small number of bits to specify the state, keeping the CA rules few in number, and having few cellular neighbors.

We use a von Neumann neighborhood[2], i.e. a cell in a 2D space looks at its four nearest neighbors and its own state[a]. At present our CA is implemented on a 2D space. We plan to extend it to 3D CA-spaces soon, since the general approach taken in our model is independent of the dimensionality of the CA-space. We use two different CAs for the growth phase and the signaling phase in our neural networks. The cell interactions are similar in both CAs. The main difference between the two phases, and hence the two CAs, is that the

[a] In general this situation is refered to as a "neighborhood" with 5 neighbors.

type of a cell can only change during the growth phase, e.g. a blank cell can change to an axon cell. (See later sections for details.) Cell types do not change during the signaling phase, although cells do change their activity values and exchange neural signals. Since the cell-to-cell interactions are similar for the 2 phases, the same CA can be used in both. This might be useful, when a future learning algorithm needs to grow new connections, depending on the activity in already existing axonic or dendritic connections.

Partitioned CAs

The states of our CAs have two parts, which are treated in different ways. The first part of the cell-state contains the cell's type. The second part serves as an interface to the cell's neighborhood. It contains the cell's activity and the input signals from the neighbors. Characteristic of our CA is that only part of the state of a cell is passed to its neighbors, namely the signal and then only to those neighbors specified in the fixed part of the cell state. We call our CAs "partitioned", because the state is partitioned into two parts, the first being fixed and the second is variable for each cell (see figure 5).

The advantage of this partitioning-technique is that the amount of infor-

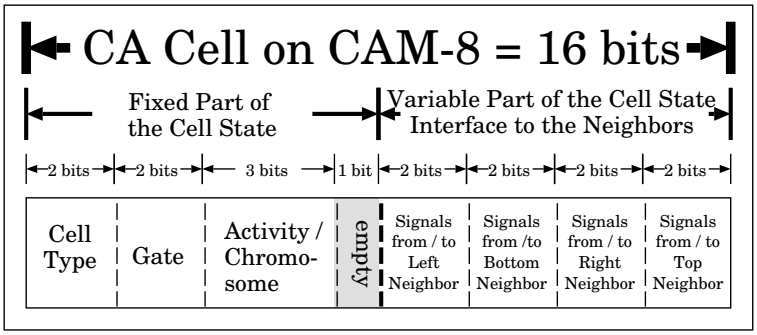

Figure 5: State representation in the new CAM-Brain model, which uses 15 of the 16 bits in the CAM-8 CA-cell. During the growth phase 3 of the bits are used to store the chromosome's growth instructions. The same 3 bits are later used to store the activity of a neuron cell during the signaling phase.

mation that defines the new state of a CA cell is kept to a minimum, due to its avoidance of unnecessary information-exchange. However CAs with many complex behaviors are usually designed with complete exchange of cell states, thus ensuring redundant information exchange. The old CAM-Brain model used such a complex CA with full exchange of state [10].

The old CAM-Brain CA needs 8 bits and 32 K rules to grow its neural networks. Even with the small von Neumann neighborhood, i.e. 5 neighbors, a 40 bit string of neighborhood information is necessary to determine the new cell state. If this CA is not compressed and transformed into a number of smaller CAs (to fit into a 16 bit look-up-table (LUT)), it is not possible to implement it on MIT's CAM-8 machine [22]. This compression and transformation makes the execution of the CA very slow and inefficient, especially if the down-loading time for new CA rule-sets is large compared to the execution time. For a CA implementation in FPGAs, down-loading a new rule-set means reconfiguring the FPGA completely. This technique is unacceptably slow in the case of FPGAs and other programmable hardware devices, especially when 3D circuits become realizable [23].

Another possibility to reduce the number of states in CAs is to use non-uniform CAs [8]. In non-uniform CAs, different rule-sets are used simultaneously, depending on the position of a cell in the CA-space. This non-uniformity can reduce the number of CA-states, and hence the amount of information exchanged between CA cells. The reduction of CA-state size causes a reduction in the total number of CA-rules, which, for a non-uniform CA, is the sum of the numbers of rules in all rule-sets. Non-uniform CAs can always be transformed into uniform CAs by adding CA-states and corresponding CA-rules. The distinct smaller rule-sets of non-uniform CAs can be stored sequentially in the larger rule-table of the corresponding uniform CAs. To distinguish the rule-sets of the non-uniform CA, in this sequence, new states are added to the state-set. This extension of the state size allows cells to use the part of the rule-table which corresponds to their "local" rule-set in non-uniform CAs.

The transformation from non-uniform CAs to uniform CAs removes the advantages of non-uniform CAs for the sake of uniformity. On the other hand, uniformity might be necessary to run a model on prospective parallel hardware [20].

Using partitioned CAs offers the advantages of non-uniform CAs by reducing information transfer, even though it is uniform. The cells pass the same reduced amount of data to their neighbors as the cells of non-uniform CAs. They do not transmit information about their own cell type. This corresponds to the fact that cells in non-uniform CAs do not know which rule-set their neighbors use. In this way the partitioning technique reduces the amount of information transfer that is necessary for a cell to determine its new state, e.g. by looking it up in a look-up-table (LUT). As a consequence, the necessary LUT gets significantly shorter than the corresponding LUT of traditional CAs. The same reasoning is valid for the storage of CA cell rules in programmable hardware. Partitioned CAs simplify the hardware programming.

This is a crucial advantage for the implementation of CAs into parallel hardware. Large LUTs or complex programs cannot be stored locally, and hence are non-parallel. Using the partitioning-technique, it becomes possible to use CAs with complex interactions for our new CAM-Brain model, and to simulate neural networks directly in parallel hardware.

The next three subsections describe the new CAM-Brain model and its implementation as a partitioned-CA in detail.

7.2 Codi-Model for the Signaling of Neural Structures in CA

As mentioned in section 6, in the 2D version of the old model, transportation of information through the CA was inspired by Codd [3]. Neural signals pass through 3-cell wide trails, which are gridded on a 6-cell grid (see figure 4). The neuron bodies consist of $11 \cdot 11 = 121$ cells (see figure 3). The result is a CA-space with at least 25% of the CA-cells unused. Two thirds of the occupied cells are sheath or blocker cells, and remain constant during the whole signaling phase of the neural network. This is inefficient use of the CA-space. An efficient CA neuron model should satisfy the following criteria.

- Its CA trails should be only one cell wide. For example, it could use signaling techniques similar to the model proposed by von Neumann [2].
- The grid should be dispensable, so that signal trails can be dense without disturbing each other.
- Contiguous signaling should be possible.
- Neuron bodies should be as small as possible, e.g. one cell.

We succeeded in satisfying all these criteria and incorporated them into a single model for the signaling of neural structures. We named the model "CoDi" based on the essential features of its axonic and dendritic cell behavior, i.e. to "collect" or to "distribute" the cellular values from/to the CA cells' neighbors. The CoDi-model works with four basic types of CA cells, neuron body, axon, dendrite and blank. Blank cells represent empty space. They do not participate in any cell interaction during the signaling of the neural network.

Neuron bodies consist of one CA cell. The neuron body cells collect neural signals from the surrounding dendritic cells and apply an internally defined function to the collected data. In the CoDi model the neurons sum the incoming signal values and fire after a threshold is reached. This behavior of the neuron bodies can be modified easily to suit a given problem. The output of the neuron bodies is passed on to its surrounding axon cells.

Axonic cells distribute data originating from the neuron body. Dendritic cells collect data and eventually pass it to the neuron body. These two types of cell-to-cell interaction cover all kinds of cell encounters. Axonic and dendritic cells do not need to know anything about the cell type of their neighbors. They behave in the same way towards any type of neighbor. Therefore it is not necessary to distinguish between the different cases of cell-to-cell interactions, e.g. axon-neuron, axon-dendrite, dendrite-dendrite etc, so that designing separate rules for each case is unnecessary. The two basic interactions cover every case, and they can be expressed simply, using a small number of rules. (This simplicity is essential when we come to implement CAM-Brain directly in hardware.) To make axonic and dendritic signal trails (which can be dense in the CA-space) without blockers or sheathes, the four basic cell types need further specification. To the two bits which specify the basic cell type, we added two more bits to set the "gate" of a cell. (See figure 5.) A neuron cell uses this gate to store its orientation, i.e. the direction in which the axon is pointing. In an axon cell, the gate points to the neighbor from which the neural signals are received. An axon cell accepts input only from this neighbor, but makes its own output available to all its neighbors. In this way axon cells distribute information. The source of information is always a neuron cell. Dendritic cells collect information by accepting information from any neighbor. They give their output, (which is the sum of their inputs) only to the neighbor specified by their own gate. In this way, dendrite cells collect and sum neural signals, until the final sum of collected neural signals reaches the neuron cell.

Each axon and dendrite cell "belongs" to exactly one neuron cell. This configuration of the CA-space is guaranteed by the preceding growth phase (section 7.3). Figure 6 shows the signaling phase with the CoDi-model. Each neuron is given two dendritic trees and two axonic trees. One of the axonic trees distributes inhibitory signals of the neuron, the other distributes excitatory signals. A neuron has two dendritic trees to maximize the amount of information that can be passed to the neuron body in each time step.

We use neural signals with signal-values in the range from -1 to 2 (= 2 bits). Signals can be contiguous. This is not possible in the old model, where neural signals travel in groups of three cells (triplets). Two of these three cells determine the direction of the signal movement. In the new model, the direction of motion is stored in the underlying signal trails. The signals are one cell wide and can fill a trail completely, as is the case in the growth phase.

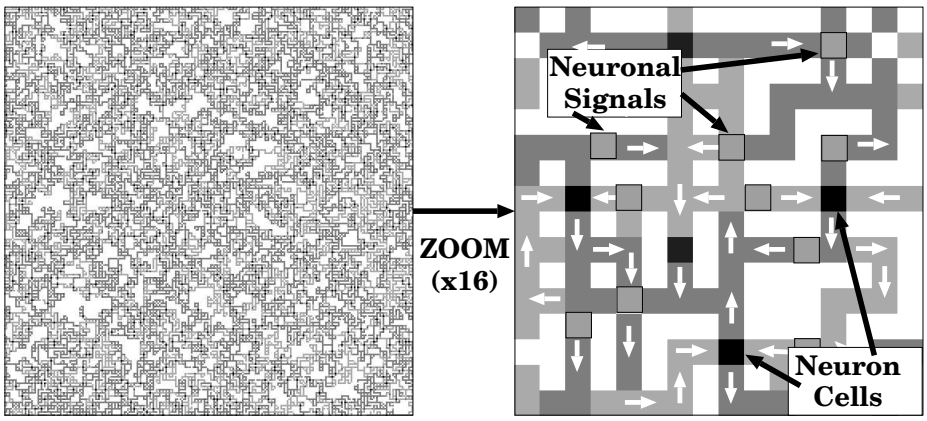

Figure 6: (Left) A CA-space with $256 \cdot 256$ cells in the signaling phase with the CoDi-model. (Right) A zoom (x16) with five neuron cells (black) with two dendrites and two axons each. The arrows inside the axonic (dark grey) signal trails and dendritic (light grey) signal trails indicate the direction of information flow during the signaling phase. In the growth phase the signals in the dendritic trails travel the other way, because they are emitted by the neuron. To improve the visualization of neural trails, the underlying chromosome is gridded with a checker-board pattern of "grow-straight" instructions. This causes the neural trails to grow on a 2-cell grid.

Synapses

The old CAM-Brain model uses synapses to transfer neural signals from axons to dendrites. The new model does not need synapses, because dendrite cells that are in contact with an axonic trail (i.e. have an axon cell as neighbor) collect the neural signals directly from the axonic trail. This results from the behavior of axon cells, which distribute to every neighbor, and from the behavior of the dendrite cells, which collect from any neighbor.

However axon-axon and dendrite-dendrite interactions of cells that do not belong to the same neuron are not possible. This is because axon cells take their information only from the neighboring axon cell that is closer to the "source" neuron on their axonic trail. Dendritic cells behave in the opposite way, by passing signals to the dendritic neighbor that is closer to the neuron or to the neuron cell itself. This configuration of the CA-space results from the neural growth phase.

These two simple behaviors of axon and dendrite cells (i.e. collect and distribute), which are exact opposites, allow axonic and dendritic trails to be dense in the CA-space, without interacting in an unintended way. The CoDi-model remains functional even if every cell in the CA-space is used.

Biological Systems have to self-assemble. The genetic encoding of biological brains is indirect, sophisticated and incomplete. There is not enough genetic information in vertebrates to encode the detailed connectivity of their brain circuits.

However, these characteristics of biological brain growth need not apply to electronic brains. Electronic circuits do not have to grow. They can be copied into hardware as initial settings. The genetic encoding can be direct. For example, a neural circuit can be stored as a spatial pattern in bitmap format. The genetic information can be complete, so that a given chromosome always produces exactly the same neural circuit and vice versa.

For our CAM-Brain models, we have kept the biological feature of growing a system. We choose a genetic encoding that is indirect, but very different from biological genetic encoding. We wanted our encoding to be one to one, so that a given chromosome leads to exactly one neural network and vice versa. In this way we ensure that a chromosome has a unique fitness value and hence do not have to spend time calculating an average fitness for each chromosome.

The advantage of a grown system is that the structural integrity of the basic components (in our case the neurons) is preserved in the presents of mutations (or alternations in the chromosome). All non-blank cells in the CA-space can be assigned to exactly one neuron, (the neuron from which they were grown). This is the case for any chromosome. Though the connectivity of the neural network may be unsuitable for a given problem after the growth according to a arbitrary chromosome, a consistent structure of the neural network is always guaranteed. The importance of this feature becomes apparent when we compare a) the effect of genetic operators on a neural system that grows and b) neural systems that are mapped directly to their chromosomes. In the first case a mutation leads to a small, local change in the direction of a signal trail. In the case of direct mapping, a mutation and the resulting local change of a cell, which may belong to a signal trail of a complex axonic tree, can lead to disconnection of a large amount of circuitry. This circuitry would then be disconnected from its source neuron, and loose the integration into the basic structure of the neural network model. The situation gets worse, if we look at the way new neural connections come into being. The probability that a new and useful signal trail evolves if all cells in the CA-space are mutated independently (as is the case for direct mapping), are very small. However a mutation in a grown neural network (for example from "turn right" to "split right") leads to the growth of a new neural signal trail that is consistently

integrated into the structure of the source neuron and hence the whole network. This feature of a grown system reduces the search space for possible CA-patterns to the patterns that represent consistent neural structures. The selection of an indirect genetic encoding is a logical consequence of the decision to grow our networks. The principle of growth and indirect genetic encoding has already been used in the old CAM-Brain model. But the way the growth of the neural networks is genetically encoded in the old model (see section OldCAM-BrainModel) has several disadvantages.

- When the spatial distance between neuron bodies can vary, one has to ensure that chromosomes do not overlap, by establishing buffer zones in the chromosome subspace (see figure 3). This forces a tradeoff between wasting memory and the freedom to position the neurons. Additionally, in a neural network, it is always the case that some neurons have more frequent and longer connections than others. These "bigger" neurons have to "wrap around" their chromosomes, i.e they run short of chromosome-growth-signals. While the "smaller" neurons, in an area where the neuron density is high, may not need all their chromosome information.

- A growth signal that comes to a junction has to split into two identical signals in order to supply all branches of its axonic and dendritic trees. This produces a kind of "fractal growth" and hence restricts the range of possible growth patterns.

- The position of the chromosome information in the CA-subspace has no connection with the position of the circuitry that grows from it. So, it is not possible to use a local learning algorithm that can change a local connection and rewrite the chromosome according to that change.

The new CAM-Brain model solves these problems by distributing the chromosome initially throughout the CA-space.

Each neuron no longer has its own chromosome. Instead, every cell in the CA-space contains one instruction of the chromosome, i.e. one growth instruction, so that the chromosome belongs to the network as a whole.

The neuron bodies can be positioned arbitrarily in the CA-space. (See figure 7.) Neuron cells send two kinds of growth signals to their neighbors, either "grow a dendrite", or "grow an axon". These signals are passed to the direct neighbors of the neuron cell according to its chromosome information. The blank neighbors, which receive a neural growth signal, turn into either an axon cell or a dendrite cell. (See figure 7 (right).) The just grown cells, then receive the next growth signal from the neuron body cell, which is sending them

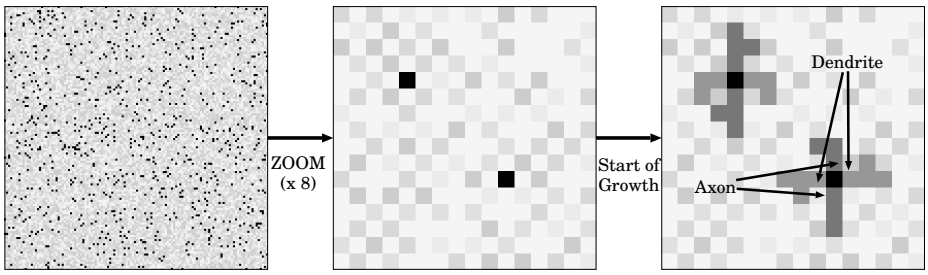

Figure 7: (Left) An initial pattern of $128 \cdot 128$ CA cells with a neuron-density of 5%. The (black) neuron cells are randomly positioned. (Middle) A zoom (x8) with two neuron cells on a random (2-cell gridded) chromosome. (Right) The beginning of the growth phase, after three CA-steps.

continuously. To decide in which directions axonic or dendritic trails should grow, the grown cells look at their chromosome information, that encodes the growth instructions, of which there are 6 types (grow straight, turn left or right, split left forward or right forward and block growth, see figure 8). Using its chromosome instructions and knowing from which direction the growth signals come, a cell can determine the neighbors to which it transmits the incoming growth signals. (See RHS of figure 8.) If, for example, a growth signal comes from the left neighbor to a cell whose chromosome instruction is "split right", it will pass the growth signal to its right and bottom neighbors. The model can easily be expanded to other types of growth signal, e.g. a T-split.

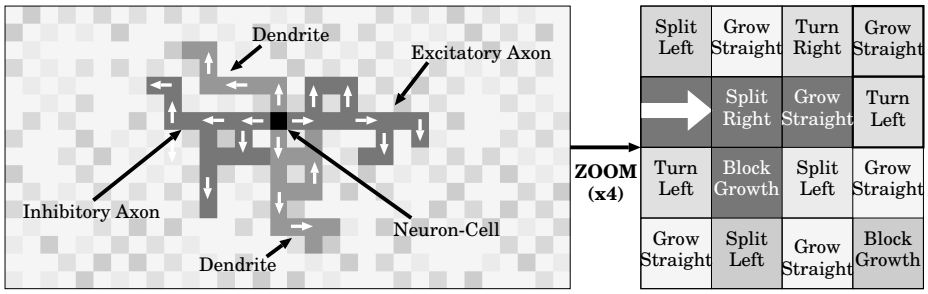

Figure 8: (Left) A neuron in the CoDi-model with two dendrites and two axons. The arrows inside the axonic and dendritic signal trails indicate the direction of information flow during the growth phase. The neuron grows on a random (2-cell gridded) chromosome in the CA-space. (Right) A signal trail grows according to the underlying chromosome instructions. The trail continues to grow over the two striped cells in the next two time steps. The branch of the trail that leads downward is blocked by the blocker instruction in the chromosome.

After a cell is grown, it accepts growth signals only from the direction from which it received its first signal. This "reception direction" information is stored in the "gate" position of each cell's state.

Figure 9 shows two cut-outs of neural networks at the end of a growth phase. The growth phase stops when growth is saturated, i.e. when no more trails (as specified by the instructions of the network's chromosome) can be formed.

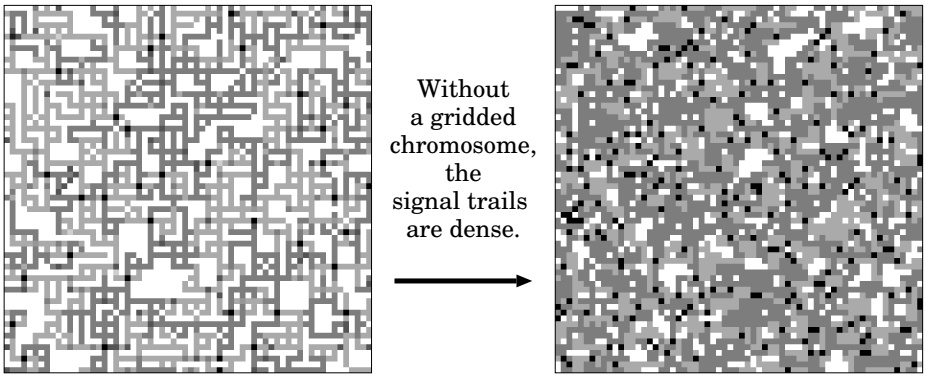

Figure 9: Two cut-outs of neural networks at the end of the growth phase. (Left) A network grown from a gridded chromosome. (Right) A network grown without gridded chromosome instructions.

In most of our figures, we have used gridded neural structures, in order to render the axonic and dendritic trails more visible. To force neuron trails to grow on a 2-cell grid, the underlying chromosome is initialized in a "checker-board" pattern. The "black" squares of the checker-board are filled with "grow-straight" instructions and the white squares are filled with arbitrary growth instructions. Since gridding is only a visualization trick, an unconstrained evolutionary algorithm can grow trails which are dense (the RHS of figure 9 shows an example of an ungridded neural network).

Beside the "grow straight", "turn left" etc. instructions, there are "block-growth" instructions in the chromosome. They cause a cell in an axonic or dendritic trail to block the passage of the incoming growth signals to its neighbors, so the branch of the trail stops growing. With a high density of blocker instructions in a chromosome, an area in the CA-space can be transformed into a more or less impenetrable "wall". Figure 10 shows two "walls" with different densities of blocker instructions in the two regions. A combination of walls can partition a neural network into more or less independent modules. This

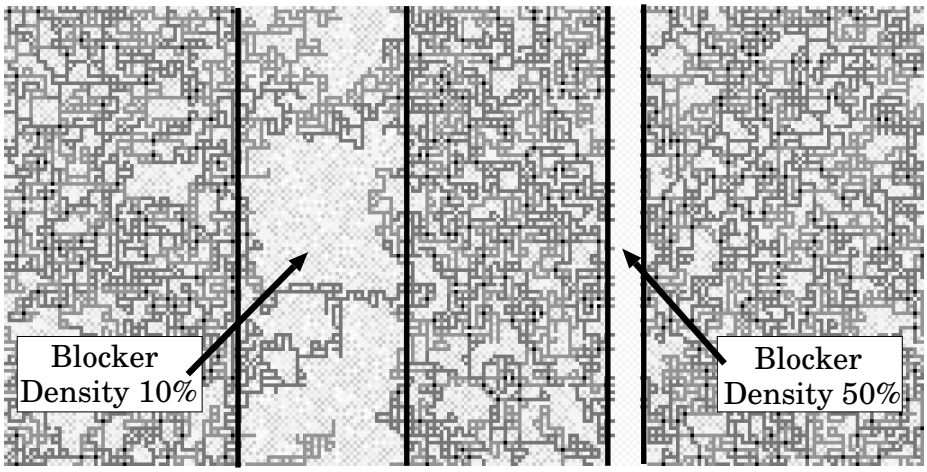

Figure 10: The use of "walls" to modularize neural circuits in the new CAM-Brain model. The density of blocker instructions on the chromosome determines the probability of trail-growth through a wall and hence the penetrability of the wall.

modular partitioning will probably play a major role when the time comes to assemble evolved modules to make an artificial brain. On the CAM-8 with 128 M cells, it is possible to evolve several tens of thousands of modules with 100 2D neurons each.

The grown circuitry and the chromosome are connected locally, so that a local learning algorithm can make local changes in the phenotype and the genotype of the network. This enables the use of Lamarckian evolution (with local learning algorithms) in addition to Darwinian evolution.

The distributed chromosome technique of the new CAM-Brain model makes maximum use of the available CA-space and enables the growth of any type of network connectivity. It also allows local learning to be combined with the evolution of grown neural networks.

7.4 A CA-Model Directly Evolvable in Hardware Must be Simple

Why is the simplicity of our CA-model important? We built CAs to model brain-like structures. Biological brains, the highly optimized results of millions of years of evolution, are obvious sources of inspiration for our work.

Even the tiny brains of insects consist of about a million neurons with complex connectivity. In order to model a 3D neuron with CAs in a way that preserves the essential functionality of real neurons, we estimate that we need

at least a cube of 10^3 CA cells. So the minimum size of CA-space to model an insect brain is 10^9 cells. To update a billion CA cells at acceptable speeds will require the use of parallel hardware. The ideal case for CA updating is to give each CA cell its own processor, so that the whole CA-space is updated, in parallel, in one clock cycle.

To build parallel hardware with 10^9 independent CA processors (for a small mammalian brain it would be 10^{12} CA processors), the components must be very simple in their internal logic, with very little memory. Large look-up-tables (LUTs), as are common in traditional CA implementations, will be impossible. Hence, for a CA to be executable on parallel hardware, the rules must be few enough to be expressible within the limited number of logic gates per CA processor inside the parallel hardware. For this reason we tried to keep our new CAM-brain model as simple as possible.

The CA state format of the new CAM-Brain model has 2 bits for the basic cell type, 2 bits for the gate, and 3 bits for the chromosome information. The chromosome is only used in the growth phase. In the signaling phase, the bits of the chromosome information position of the state are used to store the cells' activities. Together there are 7 bits of internal cell state. (See figure 5.) The growth signals and activity signals are encoded in 2 bits. Thus every CA cell receives 2 bits $\cdot$ 4 neighbors = 8 bits of signal information. So altogether, a cell in our model needs 15 bits of memory. As the CAM-8 hardware works with 16 bits per CA cell, it is possible to update our new CAM-Brain CA-space in one machine cycle of the CAM-8. Since the new model does not need sub-cells (a CAM-8 feature allowing more than 16 bits of state per CA cell), it can use the full CAM-8 CA-space of 128 M cells with an update speed of 200 M cells per second.

Considering the fact that each CA cell uses 2 bytes of memory per cell, and that the underlying principles of our CA-rules are expressible with a small number of logic gates, we are hoping to be able to implement the new CAM-Brain model on a large FPGA, so that the neural structures evolve directly in hardware, at hardware speeds.

8 Conclusions

This chapter introduced a new CAM-Brain model for the evolution of brain-like structures, based on cellular automata.

The new model, as the old, separates the growth and signaling phases. The underlying CAs for both phases are based on partitioned-CA, which will simplify an implementation in parallel hardware. For the growth phase, we introduced a distributed-chromosome technique, which allows us to grow arbi-

trary neural structures, and enables the combined use of local learning and evolutionary engineering. The neural signaling is realized with the CoDi-model, which allows maximum use of the given CA-space and reduces drastically the number of necessary CA-states and CA-rules. All presented techniques can be extended to 3D CA-spaces without difficulty.

The new CAM-Brain model can be executed on MIT's "CAM-8" hardware with several million 2D neurons in a 128 M cell CA-space, at a speed of 200 M cell-updates per second. In a 3D CA-space it will be possible to grow more then 100.000 3D neurons on the CAM-8.

The simplicity of the new model, and the use of 2 bytes of memory per CA cell allow an implementation in programmable hardware. Hence, with the new CAM-Brain model, it will be possible to evolve brain-like structures directly in hardware, at hardware speeds (a necessity, if we are to evolve neural structures with millions of neurons[24]).

Readers who are interested in following progress in the CAM-Brain Project should consult de Garis's web site, at http://www.hip.atr.co.jp/ degaris This site is voluminous (containg all de Garis's papers, book chapters, talks, essays, images, movies, future plans, etc) and is regularly updated.

1. Toffoli, T. & Margolous, N. *'Cellular Automata Machines'*, MIT Press, Cambridge, MA , 1987.
2. von Neumann, J. *'Theory of Self-Reproducing Automata'*, ed. Burks A.W.University of Illinois Press, Urbana , 1966
3. Codd, E.F. *'Cellular Automata'*, Academic Press, NY 1968.
4. Dexler, K.E. *'Nanosystems: Moleclular Machinery, Manufactoring and Computation'*, Wiley, NY , 1992.
5. Lloyd, S. *'A Potentially Realizable Quantum Computer'*, Science **261**, 1569-1571 1993.
6. Koza, J.R. *'Genetic Programming: On the Programming of Computers by the Means of Natural Selection'*, Cambrige, MA, MIT Press , 1992
7. Koza, J.R. & Bennet, F.H. & Andre, D. & Keane, M.M. *'Toward Evolution of Electronic Animals Using Genetic Programming'*, ALife V Conference Proceedings, MIT Press , 1996.
8. Sipper, M. *'Co-evolving Non-Uniform Cellular Automata to Perform Computations'*, Physica D **92**, 193-208 1996.
9. de Garis, H., *'Evolvable Hardware : Genetic Programming of a Darwin Machine'*, in Artificial Neural Nets and Genetic Algorithms - Proc. of the Int. Conf. in Innsbruck, Austria, Springer Verlag, 1993.
10. de Garis, H. *'CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speed*

Inside a Cellular Automata Machine (CAM)', in 'Towards Evolvable Hardware', Springer, Berlin, Heidelberg, NY , 1996.
11. de Hon A. *'DPGA Utilization and Application'*, in Proc. of the 1996 Int. Symp. on Field Programmable Gate Arrays, ACM/SIGDA, Feb. 1996.
12. Higuchi, T., Niwa T., Tanaka T., Iba H., de Garis H., Furuya T., *'Evolving Hardware with Genetic Learning : A First Step Towards Building a Darwin Machine'*, Proc.of 2nd. Int. Conf. on the Simulation of Adaptive Behavior, MIT Press, 1993.
13. Higuchi, T. *'Evolvable Hardware and its Applications to Pattern Recognition and Fault-Tolerant Systems'*, in Sanchez and Tomassini (eds.), "Towards Evolvable Hardware", Springer Verlag, 1996.
14. Thompson A. *'Evolving Electronic Robot Controllers that Exploit Hardware Resources'*, Proc. of the 3rd. European Conf. on Artificial Life, Springer Verlag, 1995.
15. Thompson A., Harvey I.,Husbands P., *'Unconstrained Evolution and Hard Consequences'*, in Sanchez and Tomassini (eds.), "Towards Evolvable Hardware", Springer Verlag, 1996.
16. Thompson A. *'Silicon Evolution'*, Proc. of Genetic Programming Conf. 1996, MIT Press.
17. Hemmi H., Mizoguchi J., Shimohara K. *'Evolving Large Scale Digital Circuits'*, Proc. of Artificial Life V, MIT Press, 1996.
18. Hemmi H., Mizoguchi J., Shimohara K. *'Development and Evolution of Hardware Behaviors'* in Sanchez and Tomassini (eds.), "Towards Evolvable Hardware", Springer Verlag, 1996.
19. Hemmi H., Mizoguchi J., Shimohara K. *'Development and Evolution of Hardware Behaviors'* Proc. of Artificial Life IV, MIT Press, 1994.
20. Carter, F. L. *'Molecular Electronic Devices'*, North-Holland, Amsterdam, NY, Oxford, Tokyo , 1986.
21. Gers, F. A. & de Garis, H. *'Porting a Cellular Automata Based Artificial Brain to MIT's Cellular Automata Machine 'CAM-8"*, SEAL 96 Conference S7-3 1996.
22. Gers, F. A. & de Garis, H. *'CAM-Brain: A New Model for ATR's Cellular Automata based Artificial Brain Project'*, ICES 96 Conference S7-5 1996.
23. Margolus, N. *'Crystalline Computation'*, (Preprint)1996.
24. de Garis, H. *'One Chip Evolvable Hardware: 1C-EHW'*, (submitted).