

Neuro-Cellular Automata: Connecting Cellular Automata, Neural Networks and Evolution

Jorge Tavares¹, Cornelia Kreutzer², and Anna Fedor³

¹ Department of Computer Science,
University of Coimbra, Portugal

² Department of Electronic and Computer Engineering,
University of Limerick, Ireland

³ Department of Plant Taxonomy and Ecology,
Eötvös Loránd University of Sciences, Hungary
jast@dei.uc.pt, cornelia.kreutzer@ul.ie, fedoranna@gmail.com

Abstract. A new type of Cellular Automata is presented and studied: the Neuro-Cellular Automata. The standard way of representing the rules of an automaton is replaced by an Artificial Neural Network. The motivation of this change is two-fold: to have a more biologically inspired automata and to explore the connections between Cellular Automata, Neural Networks, Evolution and Complexity. The process of the development of this kind of automata is also described, as well as some considerations about important aspects of the work. We use simple one-dimensional Cellular Automata, in order to better understand the complexity of the different components of the Neuro-Cellular Automata and to evolve it, because evolution is required to design these type of automata. A study regarding ways to measure complexity is also performed, since these measures play an important role in evaluating the population of automata in the evolutionary process.

1 Introduction

Cellular Automata (CAs) are simple type of computing machines: dynamical systems that are discrete in both space and time where different types of behavior can be observed, for example limit cycles and chaos.

These kind of models were developed in the 1940s by John von Neumann, who was interested in the problem of self-replicating systems, and Stanislaw Ulam, who studied the growth of crystals. Von Neumann was trying to develop a self-replicating robot and faced several difficulties, when Ulam suggested to design the robot around a mathematical abstraction, such as the lattice network he was using to study the growth of crystals. Since Neumann was interested in the essence of reproduction, and not in the particular implementation of the process, he followed Ulam's suggestion and concentrated on the simplest mathematical framework that would allow information to reproduce [1]. The efforts of John von Neumann resulted in the first models of CAs and culminated in the Universal Copier and Constructor (UCC), an automaton with a 2D lattice network and an algorithmical self-replicator.

However, CAs became popular to the public years later only, when Martin Gardner popularized the now famous 2D CA, invented by John Conway (for example see [1]), named the *Game of Life*. With just two states and three rules, this automaton has an impressive diversity of behavior, from apparent randomness to order, moreover it has been shown that it could emulate a Universal Turing Machine [2].

The main idea of this work is to slightly alter the formulation of a CA. One of the key components of these systems is the existence of a simple set of rules that control the automaton, from which the interesting behavior of CAs emerges. These rules are symbolic constructions, mainly expressions of the kind of the *if-then* mathematical function. Since Artificial Neural Networks (ANNs) are equivalent to mathematical functions, our aim is to replace a symbolic function in the CA by a biologically inspired one, an ANN. We designated these new systems *Neuro-Cellular Automata* (NCA). This reformulation implies several questions: What kind of ANN is necessary to encode the needed rules? Can neural networks be designed and trained to act as rule systems? How can we measure the complexity of the patterns of CAs in order to compare their behavior? How can we develop these systems? The work described here is the first step towards the definition, design and development of NCAs.

The paper is structured as follows: the description and the definition of Neuro-Cellular Automata is presented in section 2; in section 3 we describe how NCAs can be designed; in section 4 we discuss some issues regarding NCAs and finally, section 5 presents some conclusions.

2 Neuro-Cellular Automata

2.1 Cellular Automata

A Cellular Automaton consists of an infinite, regular *grid of cells*, where this grid can be in any finite number of dimensions, and each cell is in one of a finite number of *states*. The automaton runs through *time*, which is also discrete, and the state of each cell at time t is a function of the states of a finite number of cells, designated as the *neighborhood* of the cell in time $t-1$. Every cell has the same rule set for updating based on the values of the neighborhood, and for each time the set of rules is applied to the grid, a new *generation* is produced (the set of rules must be applied to every single cell in the grid). The Cellular Automaton can be viewed as a Multi-Agent System (MAS) where each cell in the grid is a single agent. They can be adapted to model phenomena, mainly in the physical sciences, biology and mathematics, such as the spread of species, but also in social sciences, such as ethnic segregation, the formation of cliques, relations between political states and attitude change [3].

The simplest forms of CAs are the uni-dimensional models. In these kind of models, the cells are arranged along a line, where the left side joins the right side forming a circular space. The number of states is two and the neighborhood is defined by the adjacent cells on either side of the cell and the cell itself. With this

in mind, we can see that there are $2^3 = 8$ possible patterns and $2^8 = 256$ rules. These 256 CAs are usually named after the decimal number that is given by the binary number presented in the rule table, with the eight possible neighborhoods listed in reverse order, as defined by Wolfram [4]. As an example, figure 1 and 2 shows the table defining *rule 30 CA* (since 11110 is 30 in binary) and *rule 110 CA* together with their patterns for 15 and 250 generations, respectively.

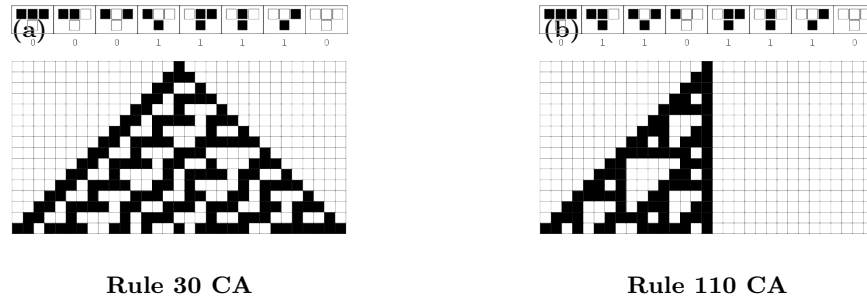


Fig. 1. Rule tables and patterns for *rule 30 CA* (a) and *rule 110 CA* (b) for 15 generations.

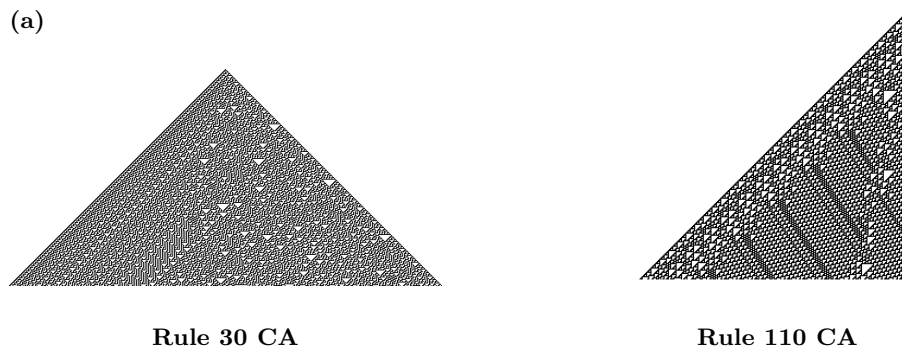


Fig. 2. Patterns for *rule 30 CA* (a) and *rule 110 CA* (b) for 250 generations.

A table completely defines a CA rule set, where each rule is the simple *if statement* given by the current neighborhood and the next state of the cell in analysis. For example, rule 30 CA says that if the current neighborhood had the pattern 1 0 0, then the middle cell becomes 1 in the next time step. Rule 110 CA says the opposite for that particular case.

These rule sets are good examples of uni-dimensional CAs that present an interesting behavior. Rule 30 CA is chaotic while rule 110 CA exhibits class 4 behavior [2], which is neither completely random nor completely repetitive.

2.2 Combining Cellular Automata and Neural Networks

The main concept of this work is to link Cellular Automata and Artificial Neural Networks. As previously described, a standard CA is composed of a rule table, i. e., a function that determines the next state of each cell, which can also be seen as a set of simple symbolic rules. The purpose of our work is to replace this function with a biologically inspired one, in this case, an Artificial Neural Network, and study some of the aspects of this type of CA.

The first question that arises is: why should we use an ANN, besides the biological inspiration? The main reason is the difficulty of designing CAs for more complex tasks. Using ANNs in simple uni-dimensional CAs, or even for two-dimensionals, does not provide any kind of advantage; the fact is that these CAs are usually interesting only to study and to show certain types of behavior. However, when developing CAs for tasks, for example in arts, it becomes very difficult to define the rule set by hand, so why don't we leave this task for Evolutionary Computing techniques [5]? Although it sounds to be a good possibility, the question is, can an ANN provide an advantage over these symbolic systems? The question is valid since ANNs have proven to be a robust approach to function approximation and classification problems for example. For more complex tasks, the NCA may be a viable option. A secondary objective of this work is the possibility of studying and joining different aspects of complex system research. For our preliminary studies to assess the viability of this system, we will use the simpler uni-dimensional CA. This allows us to study the different aspects of defining and designing NCAs, namely: the necessary type of ANN to encode the set rules; can an ANN act as a set rule; how the evolutionary system must be designed to allow the evolution of ANNs; and, how to measure the complexity of the pattern of a CA.

3 Defining and Studying Neuro-Cellular Automata

3.1 Neural Networks as CA Rules

The starting point for replacing the symbolic rule set by an ANN is to determine the kind of network that should be used. Given the number of different architectures, topologies, artificial neurons, without even considering the learning methods, designing a neural network for a CA seems to be a very hard task. Like stated before, the use of an Evolutionary Computing method is the desired way to follow, in order to avoid the choice of an architecture. While this approach is required, for the design of NCAs for complex tasks, it might not be always the case. In fact, even for a CA like the *Game of Life*, a two-dimensional CA with two states and three rules, a standard ANN should be sufficient. In this section,

we will concentrate on determining the simple and minimal aspects of an ANN for an NCA, while in the next sections, we will focus on the evolutionary process.

For uni-dimensional CAs with a neighborhood of three, like rule 30 and rule 110, the most natural choice for the neural component of the NCA is a single perceptron (it is not an objective of this paper to give an explanation of ANNs; for a comprehensive review see [6]). For this particular NCA, the perceptron has three inputs corresponding to the states of the cell and its neighbors, and a single output that gives the next state of the cell. Nevertheless, an important question arises: can this configuration, a single perceptron, learn all the possible rule configurations? For certain rules, like rule 1 and rule 15, the answer is very clear: it can. The reason is very simple: the perceptron can be viewed as a representation of a hyper-plane decision surface in the n -dimensional space of instances, in this case, the states of the CA. If the plane is linearly separable, this implies that the perceptron can learn the represented function [6]. Drawing the hyper-plane of rule 1 and rule 15, it is very simple to have a plane that separates the points belonging to state 1 and state 0 in two regions. Applying this process to all the 256 rules, it is possible to determine which rules can be learnt by a single perceptron and which cannot be. The later ones are more interesting since a solution must be found. Looking at the hyper-planes of rules that exhibit chaotic or more complex behavior, we can verify that single perceptrons cannot handle the complexity of those rules, even though they present a simple symbolic rule table. In figure 3 we can observe the hyper-planes for rules 30 and 110, and it can be seen that they are not linearly separable, demonstrating our previous statement.

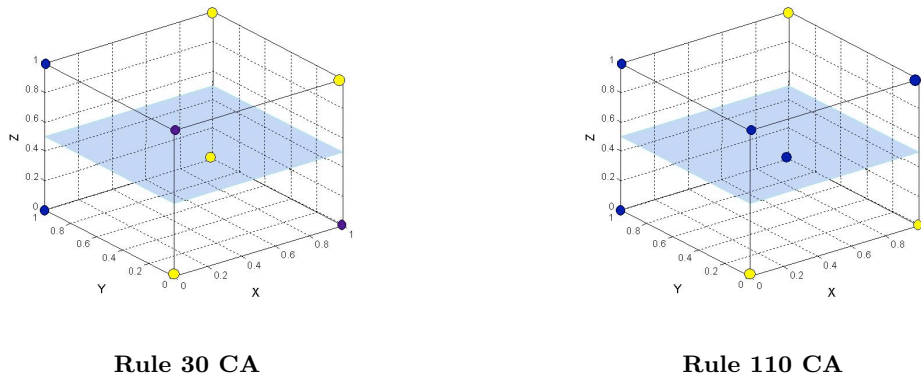


Fig. 3. The hyper-planes for rules 30 and 110 are not linearly separable.

Since perceptrons are not suitable as a general neuronal component of NCAs, the next logical step is to use a multi-layer feed-forward ANN. The reasoning is straight-forward: it has been proved that networks of this type can learn any function, as long as the hidden layer contains enough neurons. The standard

configuration of three layers is used: input layer, hidden layer and output layer. The input layer has the purpose of receiving the states from the grid, the output layer produces the next state and the hidden layer is responsible for giving the network the capability of learning (through the weighted connections between layers). The main question of using this kind of architecture is the number of neurons in the hidden layer. From ANN literature, we know that if this number is too small the network might not be able to learn and if it is too high, the ANN may run into the problem of over-fitting. Thus, it is important to determine the minimum number of neurons in the hidden layer that is capable of learning the more complex CA rules too, like rule 110.

The method to determine the minimal architecture for a one-dimensional NCA is easy: starting with the minimum number of neurons in the hidden layer, the ANN will be trained by using the Backpropagation algorithm for a given number of epochs to learn rules with a high degree of complexity (once more, rules 30 and 110). The process ends when the ANNs have learnt the rules. Applying the Backpropagation is possible since the example given for the learning method is the rule table of the chosen CA rule. Our experiments have determined that the required number of neurons in the hidden layer is four. With this number, ANNs are able to learn all the 256 rules. If the number had been smaller, for example three, the ANN could almost learn the entire set of possibilities with a small error, but for rules like 30 and 110, the ANN was incomplete. Figure 4 shows the error learning curve for rule 110. It is possible to verify that in a very short series of epochs, the ANN with four neurons in the hidden layer can learn rule 110; on the contrary, with three neurons in the hidden layer after 1000 epochs the ANN still has problems with learning it. In figure 5 we show the minimal topology for a one-dimensional NCA.

3.2 Evolving Neuro-Cellular Automata

To successfully design an NCA it is necessary to use evolutionary techniques since building the rule table of a CA for a given task is a very difficult problem. These methods have been applied before with success to the evolution of CAs [7]. Evolutionary Computing (EC) has also been widely used on the evolution of several different kind of ANNs, for example for evolving neuro-controllers for robotics [8]. It is especially useful to evolve ANNs when it is extremely hard (or even impossible) to provide a data set of examples to use with traditional training algorithms such as Backpropagation; and it also allows the design of a better network architecture.

In the case of NCAs it is also necessary to use EC techniques. In the most general version of our system, the evolutionary component is responsible for designing the full architecture of the ANN, as well as the respective training. This configuration is complex and at this stage, our goal is only to verify that the system is feasible. Since the focus of this study is the one-dimensional NCA, the EC system is not necessary to fully evolve every single aspect of the ANN. The EC component is only needed to train the weights of the ANN. The architecture is the one defined in the previous section. In spite of this, it would be interesting

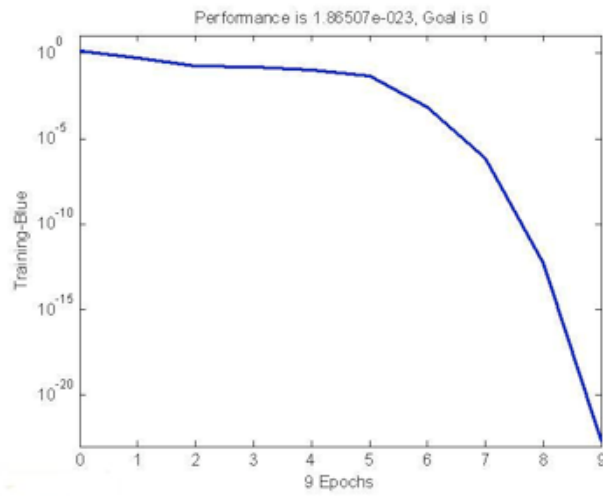


Fig. 4. The error learning curve of the ANN for rule 110.

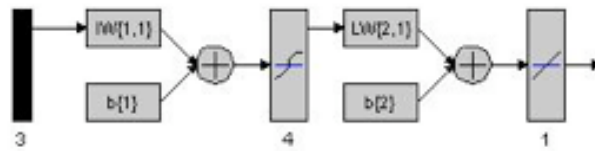


Fig. 5. Minimal topology to encode a simple one-dimensional CA.

to see if the EC algorithm, in its full configuration, would evolve the ANN in the same direction as the one we suppose.

In the one-dimensional NCA, the EC method is responsible only for evolving the weights of the ANN. In every EC system, there are three fundamental aspects to define: the representation of the problem solution (the genotype-phenotype mapping); the definition of the genetic operators that manipulate the encoding (the genotype); and, how to evaluate the corresponding problem solution (the phenotype). In the simplest form, the evolution of the ANN weights, our genotype is a single real-value vector. The phenotype corresponds to the full working ANN. Since the EC system needs to manipulate only real-value vectors, the genetic operators should be mainly inspired by gaussian based mutation operators, like the ones used in Evolutionary Strategies [5]. The reason is simple: the algorithm will be less disruptive, for this particular case. With this in mind, it is easy to set up an EC algorithm to train the ANN of the NCA.

For other types of NCA, especially for those that need to perform a given task, the previous approach might be insufficient. The system would need to evolve the entire ANN and for this task, the previous representation is incomplete. The best way to address the problem is to consider a gene on the genotype that represents an aspect of the ANN. For example, a gene that represents the number of layers, a gene with the number of neurons in each layer, another one for the weights of connections, etc. This will lead to a genotype that is variable in size and has several genes of different types that are dependent on each other. For this case, we feel that an approach based on Genetic Algorithms is the most appropriate if the genetic operators are constructed in a way, that it manipulates the genotype in a meaningful manner, i.e., it doesn't generate invalid individuals. As in the previous EC method, the phenotype is the same, the full working ANN, and it must be evaluated in the same manner.

How can we evaluate the full working ANN? How can we distinguish a good network from a less good one, or even, from a completely bad ANN? The issue of evaluating the phenotypes, the ANNs, is of extreme importance and difficulty. For one-dimensional NCA, like those that act like the traditional *rule number CA*, a fitness function could be built from the known rule table sets. The candidate solution, the ANN, would be used in the NCA and the next state indications would be compared to the ones included in the rule table set. But this strategy is not what we wish, since it does not allow the evolution of different kinds of NCAs and, even worse, it is not scalable at all. The correct way to evaluate the resulting ANNs is to devise a fitness function that allows the *emergence of NCA*. This means that the function cannot have any kind of knowledge of the objective NCA pattern. The function must be able to evaluate characteristics presented by the NCA, in other words, the fitness function must evaluate the pattern produced by the NCA in question and must give higher scores to NCA who produces patterns that can be considered chaotic or more complex, and give lower scores to NCA patterns not interesting from a complexity point of view (for example, a pattern with just one state).

The fitness function should be a *complexity measure*, or a composition of measures, in order to the EC to be able to allow the emergence of a complex NCA. This approach is also valid for a higher level of dimensions and states. In fact, the complete framework to evolve NCAs must contain a diverse set of these measures to allow a better space exploration, according to the NCA we want to have. When evolving NCAs for a given task, the fitness function should be designed in accordance with the goal of the task. The next section will provide a brief overview of the complexity measures that we use for fitness evaluation.

3.3 Complexity Measures as Fitness

Two different measures have been applied in order to evaluate the complexity of a CA. To compare the performance of the different measures standard three-neighborhood one-dimensional CA rules have been used (see [4]) with a 300×141 automaton lattice in each case.

The first complexity measure is an adapted version of the one described in [9]. In this case complexity is essentially computed from the state changes within each column of the CA matrix. Let f_i denote the frequency of state changes occurring in the i -th column of the matrix given by

$$f_i = \frac{f_c}{t} \quad (1)$$

where f_c is the number of state changes within the column and t is the number of time steps of the CA. Subsequently the frequency range for the overall matrix is given by $f_{range} = f_{max} - f_{min}$, and the mean value of f is defined by

$$f_{mean} = \frac{\sum_{i=1}^n f_i}{n} \quad (2)$$

where n is the number of columns. This information is used to compute the spread factor ω of f across the automaton matrix which is defined as

$$\omega = \frac{f_{range}}{f_{mean}} \quad (3)$$

As suggested in [9] this spread factor ω is used to define the complexity of the given CA matrix.

For the second complexity measure a different approach has been taken by using the *Kolmogorov complexity* for the last row of a given CA matrix. According to Kolmogorov, the complexity of a binary sequence of finite length is defined by the number of bits of the shortest computer program that can be used to fully describe or generate the sequence (see [10] for an elaborate discussion on Kolmogorov complexity and its applications). Several algorithms have been established to implement this universal complexity measure. In the present case the version established by Lempel and Ziv [11] has been used. This algorithm allows only the operations *copy* and *insert* to generate the binary sequence. Additionally Lempel and Ziv are not using the bit length n of the program itself to

define complexity, but a complexity value $c(n)$ to describe this bit length, however, it has been proven that $c(n)$ is an appropriate measure for the Kolmogorov complexity [11]. For a detailed description of the algorithm see [12].

The first tests have been carried out in order to examine the performance of the complexity measures, where Kolmogorov complexity gives higher complexity values to those CA rules Wolfram ([4]) has classified to be complex (e.g. rule 110) or chaotic (e.g. rule 30). It is interesting to see that these measures actually provide a way to distinguish the several types of CA rules. We use these measures as a fitness function for the evolution of NCAs. In fact, they can be used to see if the system can effectively attain in the end of the evolutionary process that an NCA act like the rule 110 CA or rule 30 CA. Furthermore, it may also be interesting to see if the measures can also be used to evolve and obtain interesting NCA patterns when changing the neighborhood, the number of states, or even the number of dimensions in the CA component of the NCA.

4 Some Notes and Future Directions

The work presented here mainly describes the definition of a different type of CA based on ANN. It was one of our main goals for this project to explore connections between different topics, such as the concepts of learning, evolution and complexity (see figure 6). In spite of that this work is still at the beginning stage, several different and important aspects were addressed and carefully analyzed. In this section, we will provide some comments about the current investigation and some directions for future work.

One of the key issues of this system is the definition of a good fitness function, which means, a good function that can measure the complexity of a pattern produced by a CA and, more importantly to use this information to guide the search process. The measures studied so far are able, to a certain degree, to accomplish this task, nevertheless it is an essential part of this work to study these measures more in depth and to find and develop new measures of complexity. One interesting way could be also the application of EC techniques. By means of Genetic Programming (GP), we could evolve a tree-based program which would be our complexity measure. To evaluate the population of GP solutions, a fitness test case built from the one-dimensional CA rules would be sufficient. The CA rules would be ranked according to their complexity and the fitness value for each GP solution would be given by how well the function is able to classify each ranked CA rule. The evolution of complexity measures can also be important to related fields of interest.

Another important question is: how should the evolutionary component be designed? This is fairly important since with a poor EC component, we might not be able to attain useful or interesting NCAs at all! The most important issue regarding this question is the representation (besides the evaluation issue, as previously discussed). For one-dimensional NCAs it might be a solvable problem, but for other types of NCAs it is not. The EC algorithm must represent an entire ANN in such a way that its search space does not become highly irregular. It

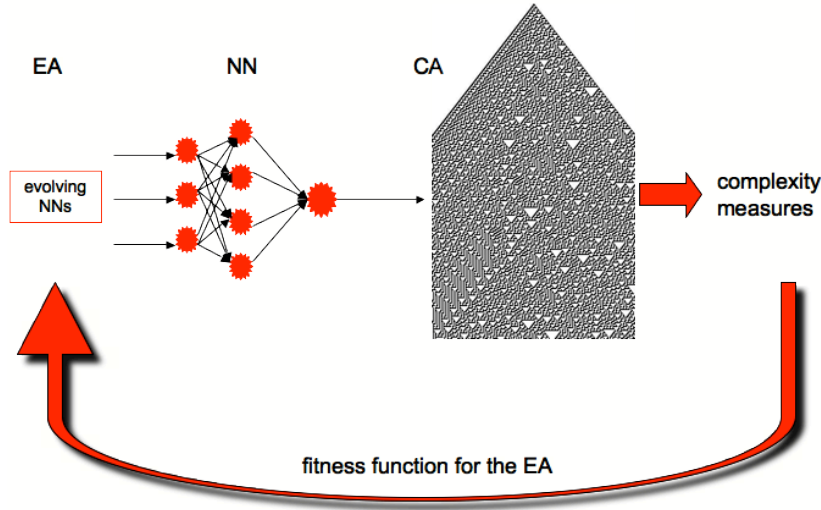


Fig. 6. The evolution of the Neuro-Cellular Automata.

seems that genotypes that represent ANNs through some construction rules or grammars, might be more apt to provide a good evolutionary framework for this problem.

Finally, what is the concept of the NCA applicable for? Although for simple one-dimensional CAs and tasks this new approach is unnecessary, it might be useful when it is applied to more interesting tasks. One of such tasks is in the terms of music synthesis and composition. Normal CAs have already been used with success for these purposes (see for example [13]). We feel that using NCAs for this kind of task might yield interesting results. Moreover, in this case, it can also be of interest to analyze how the *Zipf Power Law* can be used as a component of the fitness function. Since this power law has been used to attain more human-appealing works in the field of arts (music and images) [14], it might be useful to guide the evolutionary search process to regions of the space where the NCA for music synthesis is more appealing. This application of the NCA can also be used to image creation.

5 Conclusions

A novel kind of Cellular Automata was introduced in this work. The common way of representing the set of rules is replaced by an Artificial Neural Network. This provides a biologically inspired automata. Some of the studies performed

have showed that these automata can be developed. Several links between Cellular Automata, Neural Networks, Evolutionary Algorithms and Complexity have been addressed. This work used the simple one-dimensional Cellular Automata, providing the basis for an understanding of the needed complexity for the Neural Networks, how to evaluate the automata by using complexity measures, as well as the evolution of simple Neuro-Cellular Automata, as a mean for the design of such automata. Some directions for future research were also presented as well as the need to carry out further tests for the presented complexity measures, study the emergence of simple NCA and investigate the application of CAs in terms of music synthesis and composition.

References

1. Flake, G.W.: *The Computational Beauty of Nature*. 1st edn. The MIT Press (1998)
2. Wolfram, S.: *A New Kind of Science*. 1st edn. Wolfram Media Incorporated (2002)
3. Gilbert, N., Troitzsch, K.G.: *Simulation for the Social Scientist*. 1st edn. Open University Press (1999)
4. Wolfram, S.: *Universality and Complexity in Cellular Automata*. *Physica 10 D* (1984) 1–35
5. Eiben, A.E., Smith, J.: *Introduction to Evolutionary Computing*. 1st edn. Springer-Verlag (2003)
6. Haykin, S.: *Neural Networks: A Comprehensive Foundation*. 2nd edn. Prentice Hall (1999)
7. Mitchell, M., Crutchfield, J.P., Das, R.: *Evolving cellular automata to perform computations: A review of recent work*, Moscow, Russia: Russian Academy of Sciences, First International Conference on Evolutionary Computation and its Applications (EvCA '96) (1996)
8. Floreano, D., Nolfi, S.: *Neural Networks: A Comprehensive Foundation*. 1st edn. The MIT Press (2000)
9. Ward, E., Blank, D.S., Rolniak, D., Thompson, D.R.: *Complexity as Fitness for Evolved Cellular Automata Update Rules*. In: Genetic and Evolutionary Computation Conference Late Breaking Papers, San Francisco, California, USA (2001) 463–468
10. Li, M., Vitanyi, P.: *An Introduction to Kolmogorov Complexity and its Applications*. 2nd edn. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1997)
11. Lempel, A., Ziv, J.: *On the Complexity of Finite Sequences*. *IEEE Transactions on Information Theory* **IT-22** (1976) 75–81
12. Kaspar, F., Schuster, H.G.: *Easily Calculable Measure for the Complexity of Spatiotemporal Patterns*. *The American Physical Society* **36** (1987)
13. Miranda, E.R.: *Evolving Cellular Automata Music: From Sound Synthesis to Composition*, Prague, Czech Republic, Workshop on Artificial Life Models for Musical Applications, European Conference on Artificial Life (2001)
14. Manaris, B., Vaughan, D., Wagner, C., Romero, J., Davis, R.B.: *Evolutionary Music and the Zipf Mandelbrot Law: Progress towards Developing Fitness Functions for Pleasant Music*, Essex, UK, Lecture Notes in Computer Science, Applications of Evolutionary Computing, LNCS 2611, 1st European Workshop on Evolutionary Music and Art (2003)