

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

DESIGN OF BOOLEAN NETWORK ROBOTS
FOR DYNAMICS TASKS

Elaborata nel corso di: Intelligenza Artificiale L-S

Tesi di Laurea di:
Matteo Amaducci

Relatore:
Prof. Andrea Roli

Correlatore:
Prof. Marco Dorigo
Dott. Ing. Mauro Birattari
Ing. Carlo Pincioli

ANNO ACCADEMICO 2010–2011
SESSIONE III

Key words:

Boolean Networks

Robotics

Memory

Contents

| | |
|---|------------|
| Acknowledgements | iii |
| 1 Introduction | 1 |
| 1.1 Outline of the work | 2 |
| 2 Boolean Networks | 5 |
| 2.1 Boolean Networks model | 5 |
| 2.1.1 Structure | 5 |
| 2.1.2 Dynamics | 6 |
| 2.2 Random Boolean Networks | 7 |
| 2.2.1 Structure | 8 |
| 2.2.2 Dynamics | 8 |
| 3 Boolean Network Design | 13 |
| 3.1 Introduction | 13 |
| 3.2 Background Concepts | 14 |
| 3.3 Related work | 17 |
| 3.4 Boolean Network Robotics | 17 |
| 3.4.1 BN-robot coupling | 18 |
| 3.4.2 Metaheuristics techniques for BN design | 19 |
| 4 Memory | 23 |
| 4.1 Introduction | 23 |
| 4.1.1 Basin of attraction | 24 |
| 4.2 Learning using basins of attraction | 27 |
| 4.2.1 Re-wiring learning | 29 |
| 4.2.2 Mutating rule schema learning | 30 |
| 4.3 Emergence of memory | 31 |
| 5 Test cases | 33 |
| 5.1 General setting | 34 |
| 5.2 Phototaxis | 36 |

| | | |
|----------|---------------------------------|-----------|
| 5.2.1 | Robot setup | 37 |
| 5.2.2 | Boolean network setup | 37 |
| 5.2.3 | Training | 38 |
| 5.3 | Obstacle Avoidance | 39 |
| 5.3.1 | Robot setup | 39 |
| 5.3.2 | Boolean network setup | 40 |
| 5.3.3 | Training | 40 |
| 5.4 | Results and analysis | 41 |
| 6 | RBN Sequence learning | 53 |
| 6.1 | Sequence learning | 53 |
| 6.2 | Task definition | 55 |
| 6.3 | Robot setup | 55 |
| 6.3.1 | BN setup | 56 |
| 6.3.2 | Training | 58 |
| 6.3.3 | Results and analysis | 60 |
| | Conclusion | 67 |

Acknowledgements

I would like to thanks Andrea Roli who has offered me the possibility to undertake this work, Marco Dorigo for giving me the possibility to work in IRIDIA and Mauro Birattari for his advices and hospitality.

I would like to thanks Carlo Pincioli who has always supported me with his technical advice and, more important, with his live experiences.

I would thanks also all the IRIDIA family for their hospitality and for the time spent together.

I thanks also to my roommate and colleague Lorenzo Garattoni who has confirmed an excellent fellow of work and to my landlord Claudia Leal for her hospitality, helpfully, advice and for all the great time we have spent together.

I huge thank to my parents for giving me the possibility to achieve my studies and for their continuous support.

Chapter 1

Introduction

Many of the systems that surround us are dynamics complex systems. The goal of understanding their properties is one of the main challenges in scientific and engineering disciplines. Despite the great complexity and variety of systems, natural system such as cells and human brain, are essential to our scientific inquiry and understanding and also they provide metaphors and tools which can be effectively used for analyzing artificial agents, such as robots.

The behavior of these systems is detailed and complex. For example we can think to the functioning of the brain that is considered responsible for sensory processing, motor control, language, common sense and most other aspects of what might be called higher information processing. However, it is reasonable to assume that many of the principles concept upon which those system are designed may be described through a model that takes into account only the essential elements. A concrete example is represented by the neural networks model. It grew out of research in Artificial Intelligence with the aim to modeling the low-level structure of the brain.

Qualitatively, to understand the behavior of a complex system we must understand not only the behavior of the parts that compose it, but how they act together to form the behavior of the whole. Because we can not have information about the whole without describe each part of the system, and because these parts must be described in relation to other parts, the behavior of a dynamics complex system is difficult to understand and analyze.

Often, mathematical models that represent dynamics complex systems are described in a very complex way. For this reason is not possible to study in detail their behavior. This happens even for the neural networks model, also called "black box", in which it is possible to ob-

serve the final result but is not possible to deep analyze the internal mechanism that lead to those result.

We identify Boolean Networks (BNs) as a suitable alternative mathematical model due to its compact and simple structure representation able to show very complex dynamics. They have been introduced by Stuart Kauffman as a model for genetic regulatory networks and as an abstraction of complex systems used to study evolutionary processes mechanisms in living beings.

The goal of this thesis, developed in collaboration with the Institut de Recherches Interdisciplinaires et de Dèveloppements en Intelligence Artificielle (IRIDIA) of the Université Libre de Bruxelles, is to design Boolean networks for robot controller using an automatic metaheuristic technique and analyze how the resulting networks realize the observable robot's behavior.

Thanks to the simplicity of the Boolean networks model and the availability of tools for its analysis it is possible, for the first time, studying the resulting network's behavior by giving a concrete representation of its state space. The analysis have shown that the state space structure can be represented by a finite states automaton in which it is possible to examine, step by step, the path that starting from the inputs generates the target behavior.

Studies are performed on three concrete applications. In the first two we are interested in observe how the state space structure organizes itself in order to achieve two simple task, in particular phototaxis and obstacle avoidance. Then we analyze the state space in a more difficult task (i.e. sequence recognition) in which a memory structure is needed in order to achieve the task.

1.1 Outline of the work

The reminder of the thesis is organized as follows.

In Chapter 2, we present Boolean Networks (BNs) and Random Boolean Networks (RBNs) describing their dynamics and properties.

In Chapter 3, we present the approach, based on metaheuristics techniques, used to design Boolean network robotics systems. First we provide some basic concepts and reasons to use metaheuristic techniques for the design of Boolean network. Then, we describe our approach starting from the network-robot coupling to the analytic method used for the network's manipulation and training.

In Chapter 4, we analyze the state space of e Boolean network.

We first provide a definition of the main structures that compose it. Subsequently, we describe two learning algorithm able to modify the network's state space and that can be used as aid algorithm to meta-heuristics techniques. Finally, we discuss the possibility to use the RBN model to better understand the emergence of memory.

In Chapter 5, we apply our design methodology on two concrete experiments (i.e. phototaxis and obstacle avoidance) in order to analyze the resulting networks. The analysis focuses in understanding the networks behavior by analyzing their state space. In particular we observe the number of states used by a network during the achievement of a task and we study the state space structure emphasizing the presence of macro area that refer to the concept of memory.

In Chapter 6, the analysis is performed on networks trained for a more complicate task, i.e. sequence learning. This task is more difficult because it requires the presence of some memory structures in order to be performed. For this reason we deeply analyze the concept of memory in random Boolean network analyzing the network's state space.

In Chapter 7, we draw some conclusions and we give an outlook for future works.

Chapter 2

Boolean Networks

In this chapter we introduce Boolean Networks, focusing on their structure and dynamics, with emphasis on Random Boolean Networks (RBNs).

2.1 Boolean Networks model

Boolean networks (BNs) were proposed by Kauffman in 1969 as a mathematical model of genetic networks and complex adaptive systems. Despite their simplicity, BNs show very rich dynamics. For this reason, they have been used in several works in biology and complex systems.

In recent works, they have also been proposed for machine learning applications [9,11]. Another important reason that makes very interesting the use of Boolean Networks as learning systems is the fact that it is possible, a priori, define their dynamical state. They are, in fact, a particular case of discrete dynamical networks, where time and states are discrete.

In the following we describe the main concepts on boolean networks.

2.1.1 Structure

The BN offers a simple yet powerful tool based on Boolean function, and as all networks structures it is composed of nodes. The general idea is that each node has a Boolean state that can change over time. The change depends on the state of the connected nodes. In fact, this is realized associating a boolean function to each node and calculating its value at each computation step.

A BN is a discrete-state and discrete-time dynamical system whose structure is defined by a directed graph of N nodes. Each node is associated to a Boolean variable x_i , $i=1, \dots, N$, and a Boolean function $f_i(x_{i1}, \dots, x_{iK_i})$ where K_i represents the number of input of the node i . The arguments of the Boolean function f_i are the values of the nodes whose outgoing arcs are connected to node i and the list of Boolean functions F_i represents the interaction rules between nodes. The state of the system is defined by the array of the x_i Boolean variable values at time t : $s(t) \equiv (x_1(t), \dots, x_N(t))$. The connectivity of a Boolean network is defined by K , and since a Boolean variable can assume only two different values, the state space size is 2^N . Figure 2.1 reports a simple example of Boolean network.

| t | | | $t + 1$ | | |
|-------|-------|-------|---------|-------|-------|
| x_1 | x_2 | x_3 | x_1 | x_2 | x_3 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

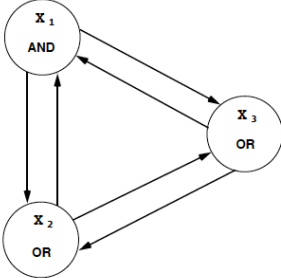


Figure 2.1: A simple example of a Boolean Network with $N=3$, $K=3$ and its function table.

2.1.2 Dynamics

To date, researchers have focused mainly on studying various properties of Boolean networks. However, many questions concerning the dynamics of Boolean networks are still open. With the term *dynamics of the network* we refer to a series of changes that takes place within a network. Boolean network dynamics is basically realized through the update of the values of each node. This update can be done in two ways: *synchronously* or *asynchronously*. In a *synchronous* BN (SBN), the states of all nodes are updated simultaneously, while in an *asynchronous* BN (ABN) not all nodes are necessarily updated at the same time. A consecutive sequence of states obtained by state transitions is called a *trajectory*. State transitions in a SBN are *deterministic*, so a trajectory starting from any state is uniquely determined. Differently, in ARBN, states transition depends on the number of nodes

and which nodes are updated at each updating step. This value could be randomly selected.

Initially the network performs a series of updates that lead to a trajectory characterized by a sequence of states all different among each other and they will not be repeated. This phase is called *transient*. The number of updates of the network that leads to a state different from the previous one defines the length of the transitory. This length can also be zero. Because the network's update is synchronous and the number of states are finite, the dynamics, sooner or later, will encounter the same states or sequence of states that will be repeated. These states are called *attractors*.

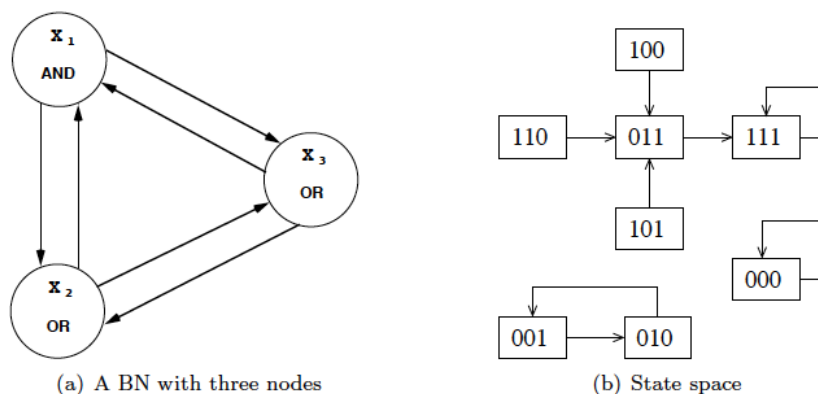


Figure 2.2: An example of a BN with three nodes (a) and its corresponding state space under synchronous and deterministic update (b). The network has three attractors: two fixed points, $(0, 0, 0)$ and $(1, 1, 1)$, and a cycle of period 2, $(0, 0, 1)$, $(0, 1, 0)$.

An attractor is a state or a set of states to which the system evolves after a long enough time. If the attractor has only a single state it is called a *fixed point*, and if the attractor consists of more than one state it is called a *cycle attractor*. There are two important quantities that characterize the dynamics of a Boolean network. The first one is the number of different attractors, and the second is the length of these attractors. Both quantities are network-specific.

2.2 Random Boolean Networks

Random Boolean networks (RBNs), a special category of BNs, were originally developed by Stuart Kauffman as a model of genetic regu-

latory networks (Kauffman, 1969; Kauffman, 1993).

His original model, and the one we will describe in the following, is based on the following three assumptions:

- The nodes are associated to Boolean functions and their state is either on (1) or off (0);
- The number of inputs is the same for each node;
- The topology of the network is chosen at random;
- The update of the network is synchronous in time.

2.2.1 Structure

Random Boolean Networks (RBNs) maintain the same characteristics as the classical Boolean networks described in section 2.1.1 and 2.1.2.

The general way to generate a RBN is to define the number of inputs for each network's nodes and the respective Boolean function. The first value, represented by the parameter K , is arbitrary defined but the wiring scheme is randomly chose. The Boolean function is defined by assigning to each of its entry a 1 with probability p and a 0 with probability $1 - p$. The parameter p is called *bias*. Usually, also the initial state is randomly chosen.

This kind of approach is very useful when the network must model a complex system, or when it contains some not well known parameters. Then, the generic properties of a RBN can be applied to a certain system with the aim to capture the mechanisms that regulate it.

2.2.2 Dynamics

In RBNs nodes are usually updated in a synchronous way. This means that the value assumed by a node at time $t + 1$ depends on the value of its inputs at time t . Actually, the update could be also done in a different way accepting to obtain totally different results, and in some cases also unpredictable. Many experiments are being made on asynchronous update of classic RBNs (Harvey and Bossomaier, 1997; Mesot and Teuscher, 2003; Rohlfshagen and Di Paolo, 2004; Gershenson, 2002; Gershenson, 2004). The effects of these asynchronous updates have been studied on some properties such as length and number of ciclic attractors that a network can achieve. The results showed that, many characteristics and properties of synchronous RBNs are

not present or are substantially different. In the following, when talk about updating scheme, we always refer to the synchronous one.

It has been also studied that, acting on the random Boolean network's parameters K and p (see Section 2.3.3) the entire dynamics change as well as the number and the type of a network's attractors. Depending on the value assumed by these two parameters the network dynamics is classified in three different regime called either *ordered*, *chaotic*, and *critical*.

In the next section we deeply describe the property of the RBN's dynamics. In particular we focus on the three regime mentioned above also characterizing the network value that carry out each particular regime.

Ordered, chaotic and critical phase

In this section we describe differences and characteristics of the three regimes of a random Boolean network. These regime can be identified with different methods, since they have several unique features.

We can imagine to plot the state of every node of a network in a square lattice and to let the dynamics proceed. In this way it will be possible to observe how much and which nodes change over the time. The dynamics of a network forces some states to assume fixed values over time, while others keep on changing. To distinguish them we color the fixed states in red while the others in green. For certain configuration of p and K (setting in according to 2.1) it is possible to observe that the network state seldom change. After having selected an initial random state, the nodes value start to change and to assume the green color, but quickly the dynamics stabilize. This phase gives as result a red sea with few and isolated green islands. We call it *ordered regime*. For a different configuration of p and K , the number of states which, from green become red, are numerically smaller. This indicates that the network states undergo continuous changes bring the dynamics to be less stable. We call this the chaotic regime. For some value, in between the two cases mentioned above, we observe a regime that displays a stability analogous to the ordered regime and a number of attractors analogous to the chaotic one. The phase transition from the ordered to the chaotic regime, also known as the *edge of chaos* or *critical regime*, occurs when the ordered green sea breaks into green islands, and the red islands join and percolate through the lattice (Kauffman, 2000, pp. 166-167).

Another interesting and studied feature of these dynamic regimes

is related to *sensitivity to initial conditions*, *damage spreading*, and *robustness to perturbations* which are different ways of measuring the stability of a network. For example we can measure the damage caused by a random change within a network just changing, damaging or perturb its nodes by altering the value they assume, or changing the topology of the network acting on the links of the nodes or in their Boolean function.

In ordered regime the damage is not propagated within the network. A perturbed network, in fact, immediately resumes the behavior before the perturbation. This is because changes cannot propagate from one green island to another. On the other hand, in the chaotic phase, a small perturbation involves a great damage to the dynamics of the network. Indeed, it is quickly propagated because perturbations can propagate through the green sea making the network itself very unstable. This behavior, also known as "butterfly effect", is typical of chaotic systems, in which a small perturbation has consequences on the entire system making it unstable. Finally, at the edge of chaos, changes can propagate, but not necessarily through all the network.

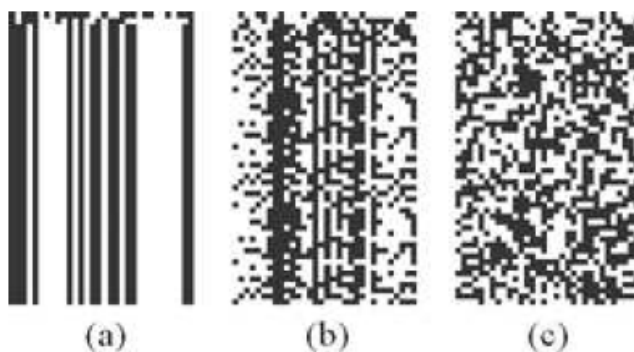


Figure 2.3: *Trajectories through state space of RBNs within different phases. a) ordered, b) critical c) chaotic.*

The propagation of perturbations in RBNs can be measured in several ways. A well-known technique is the *Derrida annealed approximation* that takes two random initial configuration, and measures their overlap.

Given two copies of the same network we can perturb one of them by altering the value of a node. Since the nodes of the network influence each other because of links between them, it is possible to map the Hamming distance (H) defined as the number of nodes with different value between the two networks. Repeating this procedure in time

and changing, for every network update, one of the N nodes value, it is possible to draw the average value of this distance and distinguish between the three different regimes. In particular, in 1986 Derrida and Pomeau, with this method, showed that dynamical phase transition is controlled by the parameters K and p .

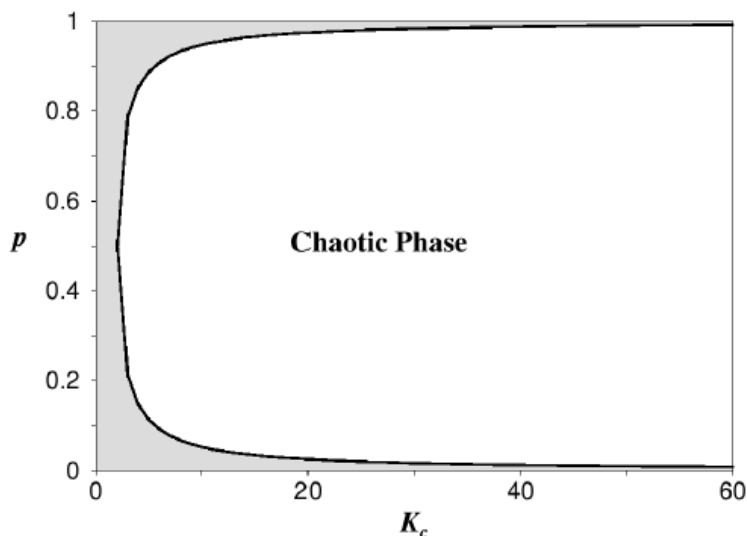


Figure 2.4: *Phase diagram for the standard NK model. The black line is the critical phase, while the gray space is the ordered phase.*

They proved that there is a value of connectivity of the network, i.e. the number of inputs that a node can accept, that serves as a threshold. This value is calculated as follows:

$$Kc(p) = [2p(1-p)]^{-1} \quad (2.1)$$

For $K < Kc(p)$ the system is very robust to the initial state perturbation (ordered phase) for $K > Kc(p)$ the system is in a chaotic phase. According to Kauffman, Stauffer, and other authors, only when $K = Kc(p)$ (the critical phase) does the NK model have the required stability properties compatible with the order manifest in the genetic networks of living organisms. Living systems need a particular stability but, at the same time, flexibility, to be able to evolve and explore the whole area of the states. This has led many people, including Kauffman, to propose that the life of natural systems evolves in a more natural and robust way in the "Edge of chaos", or in ordered phase near to critical.

Observing the plot in figure 2.4 we can see that, for example, fixing the value of $p = 0.5$, the model converges to the stable point when $K < 2$, meaning that different states tend to converge and we are in the ordered dynamics. At $K = 2$, this point becomes unstable and for $K > 2$ we are in the chaotic dynamics, in fact different states tend to diverge.

So far we have said that the dynamics of a boolean network depend on the parameter K and p . Actually, these are not the only two parameters that characterize the three different phases. There are some Boolean functions with the property of taking a specific value (0 or 1) depending just from the value assumed from one of its inputs. These functions are called *canalizing functions*. An example of canalizing functions could be the *or* function which, if it has one of its input equal to 1, it will give 1 as output value independently from the value of the other inputs. Suppose we have a network in which the functions are all *or* functions. This has the consequence that, if a node assume, at time $t - 1$, the 1 value, all the other nodes connected with it (its output nodes) will take value 1 at time t . This forcing the network dynamics to create a loop that tend to strengthen the stability of the network. It means that the number of *canalizing functions* in a network has an impact on the dynamics of the network itself.

Chapter 3

Boolean Network Design

The goal of this chapter is to outline the approach used to design Boolean network robotics systems, from the robot's input/output mapping with the BN's input/output nodes, to the analytic methodology used for the BNs manipulation and training. In particular we focus on evolutionary computation methods and, in general, hybrid meta-heuristics techniques, the computational method we will use to design BN.

3.1 Introduction

The design of complex systems has always been one of the challenges faced in the field of engineering discipline. The scientific community is used to realize mathematical models able to represent complex system. The aim is to better understand their characteristic.

Even if the Boolean network model is a fairly simple model it is characterized by a complex dynamics that can well represent a complex system (seen in Chapter 2). In addition, it is quite simple to analyze it thanks to the availability of new tools and the developing research. This is one of the main reason because, in the robotics field, it could be interesting to use this model and execute it as a robot controller. The aim of a controller, in general, is to control the robot and allow it to reach a target behavior. The idea is to design a Boolean network able to do this. We use the term "design" to indicate the process used to "sculpt" the network dynamics with the goal to carry out a specific robot's behavior. Such configuration could be also made by hand flipping, for example, some value in some Boolean function. The problem is that we don't have any guideline and also the task become quite impossible in case of big networks. In this case it be-

comes normal to think about automatic techniques as meta-heuristics techniques that we better describe in section 3.4.2.

Finally it is important to underline that, classic Boolean networks design, is aimed at the simulation of natural complex systems. In such a way it is possible to study and understand the dynamics that govern them. In the robotics field, on the other hand, the objective is to obtain a network that can be used as a robot controller. We are not interested to reach some target states inside the network dynamics, but we just focus on the displayed behavior of the robot.

3.2 Background Concepts

In this section we describe some basic concepts and reasons to use Metaheuristic techniques for the design of a Boolean network.

Combinatorial problems are used in many disciplines, including computer science and artificial intelligence. In this typology of problem, given a set of solution components, the objective is to find a combination of these components with certain properties. To solve this problems it is possible to use algorithm with the aims to find solutions that meet certain conditions or constraints of the problem itself. Combinatorial problems can be divided into two categories: decision problems and optimization problems:

- Decision Problem: computational problem in which given a problem instance, the objective is to decide whether it satisfies a certain property.
- Optimization Problem: computational problem in which, given a problem instance and objective function f , the goal is to find a candidate solution of the instance that minimizes (or maximizes) f .

BN design can be seen as a combinatorial optimization problem in which the components are the Boolean functions. The aim is to give a complete assignment to those function so that it is possible to minimize or maximize the problem's object function.

One of the main techniques used to solve these types of problems is the local search techniques. Local search algorithms start from some initial solution and iteratively try to replace the current solution with a better one found in an appropriately defined neighborhood of the current solution. These algorithms are usually *incomplete algorithms*,

which means that it is not guaranteed that, eventually, the optimal solution will be found.

In the '70, a new kind of approximate algorithms have emerged. They basically try to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are commonly called *metaheuristics*. Metaheuristics techniques (a.k.a. Stochastic Local Search - SLS) are well known for their performance on combinatorial optimization problems. They are general search strategies upon which a specific algorithm, like for instance local search, can be designed. They use a different idea, compared to the non metaheuristics techniques, for exploring the space of candidate solutions. Also they use learning techniques to improve the search for a solution as close as possible to the optimum. They consists, for example, in structure, as additional memory, that allow the algorithm to exit from local minimum (maximum) that a standard local search algorithm may achieve.

Intensification and *diversification* are the two characteristics that allow these high level strategies to have a dynamic balance between performance and results. The term intensification means, usually, the ability to exploit knowledge bases accumulated during the research process, while the term diversification refers to how to move within the search area. The balance between these two variables is important because it allows us to quickly identify regions in the research space with high quality solutions avoiding visited regions which would not lead to good solutions.

Metaheuristics can be usually classified into two different families : *trajectory methods*, and *population based methods*. The search process of the first one may be seen as an the evolution in discrete time of a discrete dynamical system. A initial solution (state) is selected, and then iteratively, a trajectory is drawn within the search space. The system dynamics depend on the type of technique; simple algorithms can be represented by simple trajectories, composed essentially of a *transient* and a *fixed attractor*. In addition, the characteristics of the trajectory can provide useful information regarding the optimization algorithm trend and its effectiveness in the specific instance of the problem. Trajectory methods are often based on local search-based methods.

Regarding the population based methods, rather than working on a single solution, they manage a set (population) of solutions. The algorithms that are based on these techniques can be seen as models of evolutionary processes. These algorithms start with an initial popula-

```

 $s^{(0)} \leftarrow \text{InitialSolution}()$ 
while termination conditions not met do
   $s' \leftarrow \text{Choose}(\mathcal{N}(s^{(t)}), \text{history})$ 
  if  $\text{Accept}(s')$  then
     $s^{(t)} \leftarrow s'$ 
  end if
end while

```

Figure 3.1: A generic trajectory based method scheme.

tion and, iteratively, a number of operators are applied to each single individual of the current population to generate the individuals that make up the next one. The main operators are the *recombination* or crossover and the *mutation*. The first one, starting from two individuals, combines them and generates other two or more new individuals. The second operator consists of a small individual perturbation that allow the adaptation of the individual itself. Finally, the individual's selection occurs through the evaluation of its fitness (which could be the value of the objective function), individuals with higher fitness have more probability to survive and to be selected as a possible solution. Genetic algorithms are part of this category.

```

 $P \leftarrow \text{GenerateInitialPopulation}()$ 
 $\text{Evaluate}(P)$ 
while termination conditions not met do
   $P' \leftarrow \text{Recombine}(P)$ 
   $P'' \leftarrow \text{Mutate}(P')$ 
   $\text{Evaluate}(P'')$ 
   $P \leftarrow \text{Select}(P'' \cup P)$ 
endwhile

```

Figure 3.2: A generic population based method scheme.

Both local search and population based methods have been used for system design, tuning and neural network training. At the end we have that the system parameters are the problem decision variables, and the objective function is defined by evaluating some resulting parameter from the system simulation. In the section 3.4.2 we will see how the techniques mentioned above can be used for the design and the training of Boolean networks, with the aim of obtain networks

that serve as robot controller.

3.3 Related work

In this section we discuss a series of studies related to the design of boolean networks. A first proposal was made by Kauffman, who introduced an automatic methodology for the BNs design. The aim of the work was focused on obtaining networks able to reach an attractor matched to a target state. The algorithm is a modification of genetic algorithm in which only the mutation operator was applied. It has both the possibility to modify the connections between the network nodes to flip the result of one entry in a node Boolean function. In addition it applies a very compelling selection methodology: when an good individual is found, the entire population is replaced by the latter. Then a mutation is performed to set the new population. The work of Kauffman is extended by Lemke, with the aim to find networks whose attractor is not only a fixed point attractor, but also a cycle attractor.

Many techniques have been proposed for the design of robust boolean networks. With the term robust we mean a network able to maintain a stable behavior even in the face of a perturbation, as a flip of one of its nodes value. For this purpose, Mihaljev, use a genetic algorithm applied to only networks with canalising functions, and Fretter extends these studies generalizing them in networks with any type of function. Further work has been proposed by Roli et al [9]. which has introduced a genetic algorithm for the design of networks with attractors of given length. In Section 3.4.2 we explain how the training of boolean networks, carried out with the same techniques used in literature, can be extended and used to teach to the networks also not trivial tasks.

3.4 Boolean Network Robotics

Boolean network robotics concerns the use of Boolean networks as robot control code. The network that controls the robot is designed through an automatic procedure based on stochastic local search techniques. The reason to use boolean networks for robot control is intrinsic in the properties that characterize the networks itself. In fact BNs have a very simple structure, but they are able to capture complex behavior thanks to their dynamics. This makes them an excellent model for the representation of complex systems and a perfect tool on which

to base the development of robot controller.

The Boolean Network robotics design is composed of two steps:

- Definition of the mapping between sensors/actuators and networks inputs/outputs
- Design the BN that serves as robot program

In the following section we describe in detail what methods were used by us to carry out the design of boolean networks robotics.

3.4.1 BN-robot coupling

The general approach to use boolean networks on robots, is to use one networks that serves as a robot controller. In this way the behavior of the robot is described through the elements that characterize the dynamics of a network, such as trajectories, attractors and basins of attraction. The available number of tools able to study these elements is high. This makes it easy the analysis through which it is possible to study the robot behavior in terms of network dynamics. This approach enable the identification of all the network parameters that affect the behavior of the robot.

Before being able to map the network nodes with robot's sensors and actuators it is necessary to identify the network size, in terms of nodes. It is reasonable to think that the size of the network grows with the complexity of the tasks that it will be required to perform. For example, with the increase of sensors number, also the number of network nodes will be increased. After this first operation we can proceed with the BN-robot coupling.

Usually, Boolean networks are studied as autonomous (or isolated) systems, in other words systems that do not receive perturbation from the outside world. With the operation of coupling the network interacts with the outside world. The nodes are divided into three categories: input nodes, internal nodes and output nodes. A simple choice for the mapping is to associate each sensor value to an input node and each actuator to an output node. This direct encoding is not always possible, due, for example, to the high number of sensors from which the net receives data. So it is possible to carry out various techniques of mapping. We have chosen a group of sensors that are all connect

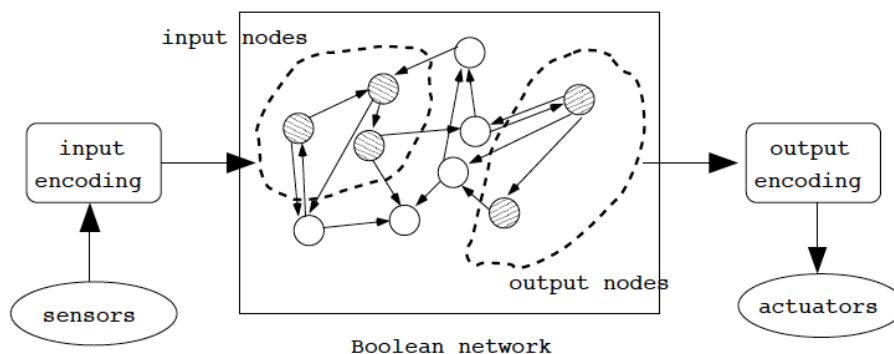


Figure 3.3: *The coupling between BN and robot.*

to the same input node. So, for example, if we have eight sensors and we group it by two, we just need of two inputs nodes instead of eight. This allows us to maintain the size of the network not too high and, at the same time, to take advantage of a good number of internal nodes acts to the propagation of data toward the output nodes.

Figure 3.1 shows the scheme of the coupling between BN and robot. Once the input and the output mappings are defined, the BN that forms the robot program has to be designed. This operation is generally performed by stochastic algorithms.

3.4.2 Metaheuristics techniques for BN design

The design of a boolean network has the aim of obtaining a network able to maximize the robot performance in terms of observed behavior. The process can be transformed in to a constrained optimization problem with the aim to maximize the robot performance. It is important to emphasize that, in these problems, it is not important to give a proof of optimality (we are not sure to find the optimal solution if it exists), since the criteria that defines the quality of the solution is an approximated criteria. We just want to find a good solution in a not too high time. This makes the metaheuristics techniques preferable to exact one, that conversely always reach the optimal solution but with a time complexity that scales polynomially with the instance size. The usage of metaheuristic technique have also other advantages. They are able to to increase, incrementally, the complexity of the research strategy. This mechanism could be useful when a robot has to learn more then one task. In fact, it is possible to implement an in-

cremental learning, starting with the most simple task until the most complex one. Moreover, it is possible to explore quickly and efficiently wide portions of the research space. That is because with have the possibility to use some problem specific information (such some well know data) inside the metaheuristic technique and make the robot learning procedure faster.

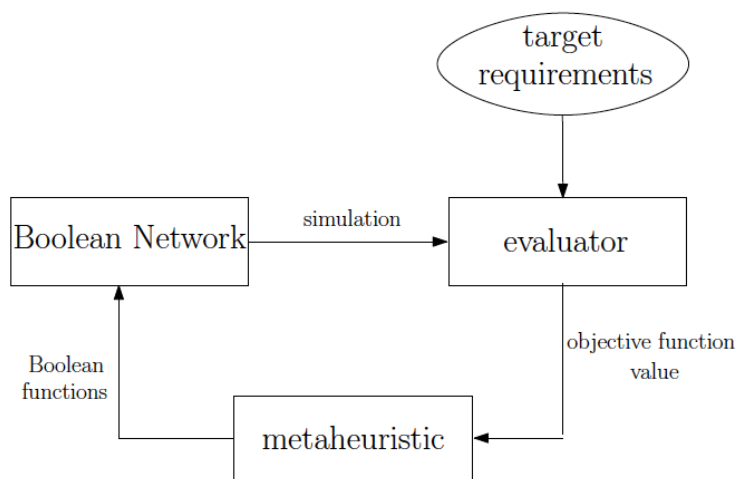


Figure 3.4: *Metaheuristics techniques approach*

The approach used is described in figure 3.4. It starts by an initial BN whose parameters are number of nodes (n), number of input for each node (K) and the value of homogeneity (p). The optimization process consist in changing, at each iteration, some of network's parameter and then evaluate the resulting network. In our case, the metaheuristic technique operate mutating one of the Boolean function value in a single nodes, then the new network is executed and evaluated. Because the network represent the robot controller, evaluate it, means evaluate the robot performance in a simulated environment. The criteria for a robot (network) performance evaluation is closely linked to the type of task assigned to them. In general, the evaluation is more good as the performance of the robot is high. In a phototaxis task, a performance measure could be the distance of the robot from the light at end of an experimental run. In robotics field, and also in our case, the simulation process coincides with a robot simulation that seeks to achieve a given target under specific initial conditions. The reason because it is very frequent simulate a robot behavior is that, training process, is often expensive in terms of time and setting on real robot. After the phases of simulation and evaluation we have,

as a result, a value that represents the goodness of the network. This value is passed to the metaheuristics which can decide whether to keep the network, or discard it. The process ends when it reaches a maximum number of iterations or a desired robot behavior represented by a good evaluation.

Chapter 4

Memory

In this chapter we introduce the concept of memory applied to discrete dynamics systems with special focus on RBN. After an introduction of the main concepts, we analyze the space-time patterns and the attractors of random Boolean networks. Subsequently, we briefly describe some learning algorithm able to modify the network dynamics creating new attractors and changing the transient. Finally, we discuss the possibility to use the RBN model to better understand the emergence of memory in discrete dynamics system.

4.1 Introduction

RBN are usually taken as sparsely connected systems where the number of inputs of each node (K) is much smaller than the number of nodes (N). This justifies their applications in the biology field, where a gene is directly regulated by few of many other genes in the genome, and neural networks application, where a neuron has a relatively low number of direct input from other neurons of the brain.

The number of inputs per node in a random boolean network influences its dynamics. Random links imply a special geometry of the network. In the case of completely connected networks, the spatial disposition of the network element is irrelevant for the dynamics of the network itself because each elements is connected to all the others. In sparsely connected networks, on the other hand, each element has a limited effect on the others. The extent of such effect increases with the connectivity degree of the nodes. In the following we use the term "element" to indicate a node of the network, and the term "neighborhood" to identify the nodes to which a specific network element is connected to.

John Hopfield (1982) presented an asynchronous recurrent neural network model. It is capable of producing, what he calls, "general content addressable memory". The system dynamics can reach areas in the state space in which the number of states remains stable. This brings the system to the equilibrium. The addressable memory is the result of a hierarchical classification of the states space of the network. RBN's attractors may be seen as "content addressable" memory in the way Hopfield refers to it, but in RBNs we have an additional element. The model proposed by Hopfield, in fact, captures only the attractors, with any notion about reliable convergence to transient outside the attractors. In the RBN, since the transient deterministically emerge, the state space undergoes hierarchical classification even along the transient, starting from the root of each subtree. In this chapter we refer to the concept of "memory" view from this last concept. Synchronous random network could have great potential in the realization of content-addressable memory thanks, not only the classification in attractors, but also within the transient subtrees that leads the formation of complex hierarchies usable as memory.

In biological networks, as well as in the brain's neural network, the topology of the attractor basins field and its subtrees must have a structure capable of working in a wide variety of contexts. The dynamics of these networks must, therefore, be versatile but not chaotic, to ensure reliable behavior. In other words, there must be a balance between order and chaos. In RBNs the balance between order and chaos is naturally made by the transition between these two regime. Simply tuning some network's parameters, like wiring between nodes, or degree of canalisation function (see chapter 3), we can move the dynamic behavior across the transition. The range of topology of basin attractor and its capability of complex categorization, suggest that basin may merge as the network's cognitive substrate (Wuensche 1992b). The difficulty in explaining the memory, in biological neural networks, just through attractors, could reside in the presence of very long transient in order to achieve an attractor while the reaction time in biological systems is extremely fast.

4.1.1 Basin of attraction

In this sections, following to Andrew Wuensche (1996) [1], we introduce some concepts used to describe the basin of attraction of a discrete dynamical system.

The behavior of a RBN can be analyzed from two points of view.

The first concerns the study of local dynamics, by extrapolating space-time patterns from individual trajectories, while the second refers to the global dynamics. The global dynamics emerges from the study of basins of attraction, attractors and its transient. The basins of attraction, can be reconstructed and portrayed by a diagram that connects all the global states according to their transition. An example is shown in Figure 4.1. Attractors may be fixed points or cycles. The basin of attraction's task is to attract, various regions of phase space, within the basin of attraction through a transient. As we can see from the image, in discrete dynamical systems, such as cellular automata (CA) and RBN, transient emerge outside the attractor.

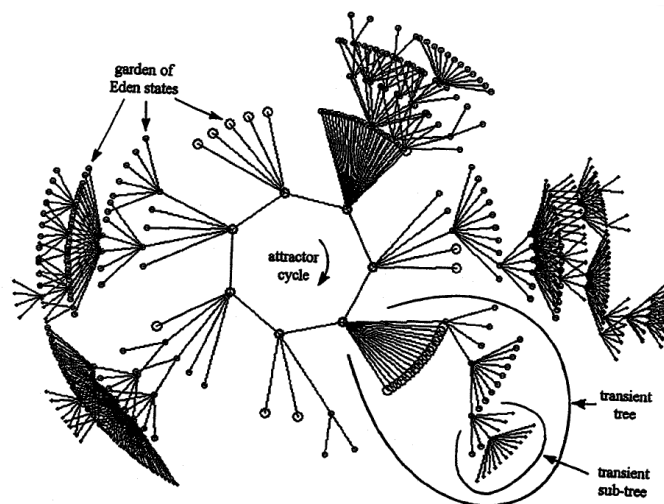


Figure 4.1: *Basin of attraction of RBN ($n=13$ $k=3$). The attractor has period 7. The direction of time is from the garden-of-Eden states to the attractor, then clock-wise*

The dynamics is driven by an iterative updating procedure that produces a succession of global states, the *network trajectory*. Every single trajectory through states is represented by a space-time pattern, i.e. it is possible to identify regularities in the network trajectories, in the form of space-time patterns. Setting a specific nodes value for an initial RBN at time t_0 , a sequence of states at time $t_1, t_2, t_3 \dots$ will be generated through the application of its updating process. The connection between the elements is not a completely connection, so the layout of the network and the conditions that occur in his boundary, are important parameters to understand the dynamics of the system.

A RBN state has only a successor but could have an arbitrary number of predecessors, which is usually called pre-image. States without

a pre-image are called garden-of-Eden. They can not be reached by the normal evolution of the network but must be imposed from outside. Even if an initial state can set always a single future dynamics, at each iteration the intermediate state may have a different history. The path followed by the dynamics can be very different but it could finish always in the same attractor.

The state space of a network with n nodes is 2^n . Each path, realized by the network dynamics, meets repeated states and, sooner or later, must reach a states cycle (the attractor). This because the number of states is finished and the updating of the network is synchronous. So that, the attractor can be a single state, a stable point cycle to itself or could have a period of arbitrary length. All the paths that lead to the same attractor, including the attractor itself, form the *basin of attraction*. The state space is typically subdivided into several basins of attraction. The set of those basin of attraction form the *basin of attraction field*. A trajectory is simply a particular path within the basin of attraction. The portion of trajectory that remains outside the attractor is called *transient*, and usually forms, together with other transients, a branching tree with the garden-of-Eden like leaves. A *transient sub-tree* is the set of all paths from garden-of-Eden states leading to a state within a transient tree. The basins of attraction have typically a topology described by trees (with leaves that start far from the attractor) directed toward a cycle.

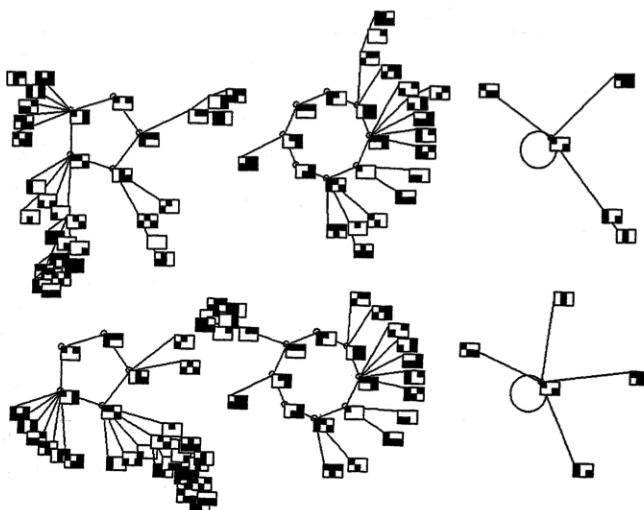


Figure 4.2: *The consequences of mutating a single bit in the rule table of just one element of a small RBN, $n=6$, $k=3$.*

The study of the basin of attractors is also important to characterize the stability of the network in presence of perturbations. Figure 4.2 shows the change in the basin of attraction as a consequence of the mutation of a single bit in a node's Boolean function. The network in the example is small, so we are able to show more clearly the configuration of each state. Larger networks are affected in a similar way. Since a single bit perturbation will be reflected across the whole system dynamics, it could happen that, even in very large networks, this might lead to drastic changes as the breaking of a cyclic attractor.

In the top three examples depicted of the image three basins of attraction are shown that compose the basin of attraction field of a RBN. In the bottom examples it is possible to see the changes in basins of attraction field after one bit mutation was performed in a single network element.

4.2 Learning using basins of attraction

As described in the introduction, the basins of attraction represent the memory of the network. Separated basins of attraction within the basin of attraction field, together with the different nodes in which the dynamic evolves, characterize the state space. All states, except the garden-of-Eden states, are addressable memory states. All the attractors and the complex hierarchies created by transient realize the network's collective memory.

It is reasonable to think that, altering the structure of a state space, we can modify the network's knowledge to let the system perform a specific behavior. The concept is very similar to the one proposed by Kauffman (Kauffman 1993) and which he describes as " a walk in parameter space seeking good attractors". In other words, he proposes to perform several movements within the fitness landscape looking for good attractors. In our case finding good attractors is not enough. We are also interested in good basins of attraction and its subtrees because we use Boolean networks as control system and the robustness is fundamental.

There are two types of algorithms that allow a RBN to learn (or forget) "from experience". Both can be used as aid algorithm to meta-heuristics techniques. The idea is to perturb either network wiring or Boolean function value. These algorithms require the presence of a "teacher" that specifies the pre-image that a state must learn or forget. The effect of learning, namely the change in the basin of attraction

field structure, could be significant in the case in which the pre-image to learn requires many changes to the network. For what concerns the procedure of forgetting, the effects are usually minor. These learning algorithm are not effective in cases in which we want to achieve a desired behavior starting from a random network because the network would be moved away from the desired dynamics. It should be emphasized, in fact, that the methodologies described below are suitable for tuning networks that are already close to some desired behavior.

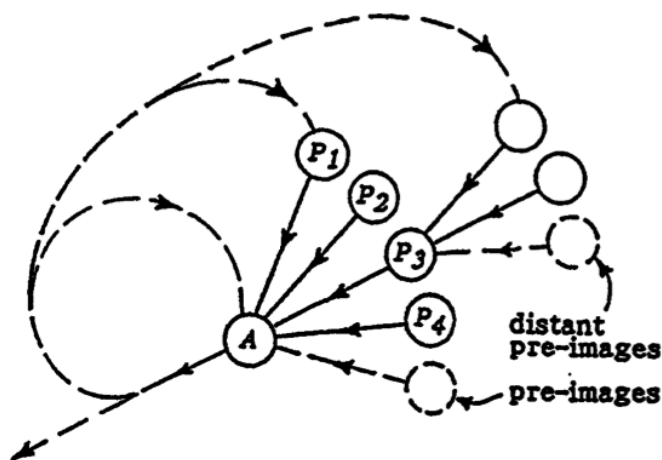


Figure 4.3: *Example of pre-image in a transient subtree.*

With reference to the image 4.3, suppose we want to set P1 as a pre-image of A. Any discrepancy between the two states, i.e. the real successors of the state P1 and the real pre-images of the state A, may be corrected in one step by changing the network wiring or Boolean function called also rule scheme. Through these two methods we can create for example fixed point attractors doing learn A as a pre-image of itself. We could create also a cyclic attractors if A is learnt as a distance pre-image of itself. Finally, if A is learnt as a pre-image of some one other state inside the basin of attraction field, its transient subtree may be completely or partially transplanted along with A.

Even if these two methodologies are used in isolated networks, i.e. networks that don't have any input from the environment, and their goal is to find target attractors instead of reaching an emergent target behavior, the study of the effect inside the basin of attraction field is very interesting in robotics field as well. In fact, the peculiarity of the re-wiring and mutating rule schema learning is that they prove the importance of they basin of attraction field in learning. In other

words, they show how to lead a network to evolve and organize its addressable memory by working on not only its attractors, but also on its transient tree and on the entire basin of attraction field.

In the following we briefly describe the two algorithms mentioned above.

4.2.1 Re-wiring learning

Imagine having to teach a network that the pre-image of the state A must be the state $P1$. As the natural dynamics of the networks develops, the successor of $P1$, and therefore the real pre-image of A , is generated according to wiring-rule scheme. This state, which we will call $B1$, is likely to have a number of nodes values different from A equal about $n/2$ if the states A and $P1$ are chosen at random.

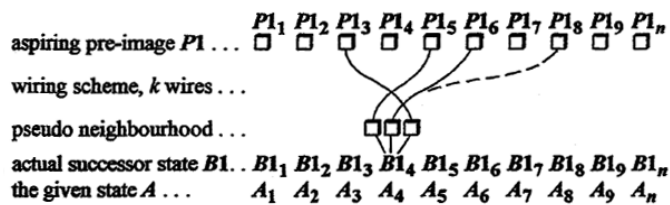


Figure 4.4: Correction of a value mismatch between $B1_4$ and A_4 by performing a single re-wiring move.

Suppose the target state and its real pre-image have a mismatch only in a single value, in our case the difference is between $B1_4$ and A_4 . One reconnection, or more if necessary, could be performed to correct the mismatch between the two values. We will limit our analysis only to the case of single wiring moves. Also, it must be noted that this is a stochastic method so we could have more than one good move that lead us to correct the value mismatch.

Assuming that $P1$ has a roughly equal proportion 0s and 1s, there will be approximately $n/2$ alternative positions in which a wire move will result in a good change. If the nodes have K inputs, the alternatives become $(K \cdot n/2)$. Finally, if the proportion of 1, in the rule table of the nodes we want change, is around $1/2$, the possible moves that allow to have $B1_4=A_4$ are $(K \cdot n/4)$. The choice to re-wire may initially be selected at random from one of the possible re-wiring moves.

After have learnt $P1$ as pre image of A , what if we learnt another pre-image of A without forgetting $P1$? If several pre-images are to be

learnt without forgetting any previous pre-image learnt, the space of possible re-wiring moves will be reduced. In fact, each change effect on the whole Boolean function of the same node. The output value of the Boolean function could change depending on the value taken from the nodes of the new pre-image and it could be different from the value used to learn the first pre-image. The learning procedure is very sensible to the initial re-wiring choice and to the learning order. We could perform good initial re-wiring moves that allow the network to learn each pre-image without forgetting the previous one. This happens very frequently when the pre-images to be learnt are very close to each other in terms of Hamming distance. We will have, in this case, a low number of mismatches to correct, and the network learning potentially increase. In conclusion, the ability of learning by re-wiring depends on: the original wiring-rule scheme, the difference from a pre-image to another, the re-wiring moves and their order.

4.2.2 Mutating rule schema learning

Correcting a mismatch between two values by re-wiring has a huge impact on the whole system dynamics and it is very difficult, in some cases, to learn a pre-image without forgetting the previous one. Correcting a mismatch by mutating the rule schema it means that we perform a flip in a single element's Boolean function ($0 \rightarrow 1$ or $1 \rightarrow 0$). Using this method we can learn a pre-image without forgetting the previous one because any mismatch between two elements can not relate to the same entry of the element's Boolean table. With this

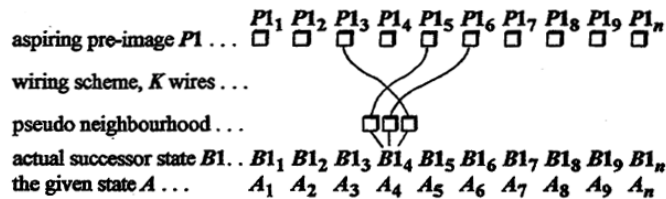


Figure 4.5: Correction of a value mismatch between B_{14} and A_4 by performing a specific flip in the node's rule table.

method there is no limit to the number of pre-images that can be learnt without the risk to forget previously learnt pre-images.

Imagine to be in the same situation described for the re-wiring method. We have a mismatch between B_{14} and A_4 . There is only one

option to correct the value by flipping a single bit in the nodes Boolean function. Of course, this mutation forces the network to "forget" some other transitions in the basin of attraction field, but always far from the pre-image to be learnt. The probability to "forget" this kind of transition decreases with the increase of the number of input nodes K . For a network with K wiring, the probability is $1/K$, i.e. for $K=3$ the probability is $1/8$, for $K=5$ is $1/32$ and so on.

As in the re-wiring learning, if two pre-images to be learnt are close in terms of Hamming distance, there will be fewer side effects in the basin of attraction field. The number of flips in this case is low.

4.3 Emergence of memory

The aim of this section is propose describe the similarities between the basins of attraction of a discrete dynamics network, and the human brain. This can bring us to better understand the similarity in the emergence of memory and find, if possible, some pattern in its structure. First of all, it is better to underline that the paradigm we use is in contrast with the one that considers the brain as a computational system, like a Turing machine. Also, the definition of the brain as a system able to manipulate symbols or representations is not considered here. The comparison between human brain and discrete dynamic system arises from the fact that the brain's structure can be describe as a billion of sparsely neurons connected together forming a complex network. The structure is very close to the structure of a RBN. Cognitive functions such as memory and learning seem to emerges as high level properties from the activity of these neurons. Now the question is : what can we say about networks that can display a behavior we are used to see in the human, or animal, world?

One of our assumption is that a network that performance "human task" should be able to categorize its state space of distributed patterns used to recognize the situation and lead to the system stability. There are not only a single activation pattern, but they are organized in hierarchy that represent the entire network knowledge. As it happens in the human brain, depending on the activated pattern, the network dynamics can have different trajectories and bring the system to different (or even not) stable points. State spaces are not just categorized by attractors, but also by the long transient tree, that emerge even far from attractors. They drive the dynamics from the chaos to the ordered state. The self organization of the state space is

the only way, for the network, to organize a space-time abstraction of its memory. This portions of memory could be reused by the network itself. We could think about a network able to perform an infinite sequence of *task1* and *task2*. In the state space, in this case, we could observe two macro hierarchy used from the network, one for each task. This means that the network can, first of all, recognize the task, and second, reuse the same portion of memory to perform it. The same happen when we link up together more networks. The entire system is able to access the specific memory field and reach an higher level cognitive state. RBNs represent a simple, but at the same time, powerful model to study the basin of attraction in this context.

The ability of reusing resources, together with the tendency to compact the knowledge in hierarchies, can lead us to think that the number of states that are truly useful for the task performance are considerably lower than those potentially usable. Of course, the number of states also depends on the type and number of tasks that the network must achieve.

In addition, systems with discrete dynamics have shown that they can perform adaptive behavior. In fact, they can continue to realize their task even if the environment in which they work undergo some perturbation. This means that the reliability of the system is very high and the access to its internal memory very fast. The natural way to organize the internal knowledge representation in order to perform an adaptive behavior is the memory-categorization in hierarchies of sub-categories. This could be also the reason why the notion of memory simply as attractors seems to be inadequate to account for the dynamics of these system.

In conclusion, a discrete dynamics system with synchronous updating uses its transient tree for a reliable categorization of its state space, far from equilibrium. The same happens with the attractors. A network that has evolved or learnt a specific dynamics should be able to reach the right section of its internal memory in few updating steps, possibly just one. It is important to notice that transients compose the the main elemtn in which the memory can be organized thanks to its sub-categories structure.

Chapter 5

Test cases

The goal of this chapter is to understand what happens inside a network while it performs a task. In order to do this, we need to study the network state space. This because the network state space, that consists in the state set used by the network dynamics, actually represents the network behavior.

Moreover, during the design process (see Chapter 3), we operate on the network Boolean function, i.e. on the network structure. Operating this modification we produce, also, a change in the network dynamics and so in the network behavior. The problem is that, just by studying the network structure changes, we don't have any information about the network behavior. The only way to study it is to study the network state space.

It is important to underline that we can study the entire state space because we are using Boolean networks. In fact, a state is described by a sequence of bit (0 and 1), so that we can think to decode it and represent the result in a graphical. This possibility is not given by neural networks. In fact, in the neural network model each neuron is described by a differential equation. Currently, the mathematical complexity of fully analyzing the state space of a large neural networks is still an unsolved problem

Specifically, in Section 5.4, we focus on analyzing the dimension of the state space. In particular we observe the number of states used by a network during, and at the end, of a design process. In addition we study the state space structure emphasizing the presence of macro are that refer to the concept of memory described in Chapter 4. Finally, we provide a new approach to represent the network state space, both using a graph structure and a pseudo code algorithm.

In order to conduct our analysis, we make use of two concrete

experiments: phototaxis and obstacle avoidance. We chose two simple and well known tasks because we are interested in deeply analyzing how this task is performed by a Boolean network..

5.1 General setting

Because running all the experiments represents a big cost in terms of time and resource (such as battery power for the robot), we need to simulate the entire experiment process in order to evaluate the robot performance. To this aim, we use ARGoS, an open source multi robot simulator. The robot we use for the simulation is a small circular robot called E-puck [17]. It is equipped with eight proximity sensors, eight light sensors and one ground sensor. The proximity and light sensors are disposed on the top of the robot as displayed in Figure 5.1, while the ground sensor is placed under the robot. The data coming from those sensors are values between 0 and 1. Regarding the proximity sensors, it signals the presence of an obstacle near by the robot. The 0 value means no obstacle, while the 1 represents an obstacle very close to the robot. The value coming from the light sensors, instead, represents a light intensity. The maximum intensity is represented by the value 1. Finally, the ground sensor is able to recognize a grayscale from white (1), to black (0). Regarding actuators, the E-puck robot is provided with two motors used to control the two wheels and eight red LEDs disposed around the top of the robot. Depending on the task typology, we can choose the right configuration of sensors and actuator to achieve the goal.

Each experiment is executed starting from 300 different Boolean network. The initial node connections and Boolean functions of these networks are randomly generated with $K=3$. The probability to have 1 in the nodes Boolean function is equal to p that assumes the value 0.5, 0.788675 and 0.9. Such homogeneity values statistically correspond to chaotic, critical and ordered regime, respectively (see Section 2.3.2). We generate 100 networks for each value. The number of the network's nodes can change from a problem instance to another.

During the experiment the Boolean network is executed as robot controller, and its dynamics are used to control the robot behavior. In order to do this, as described in Section 3.4.1, we need to realize the BN-robot coupling and configure the input/output network's nodes. Input nodes are not influenced by the network dynamics, but the values they assume are set from the robot sensors. The output nodes,

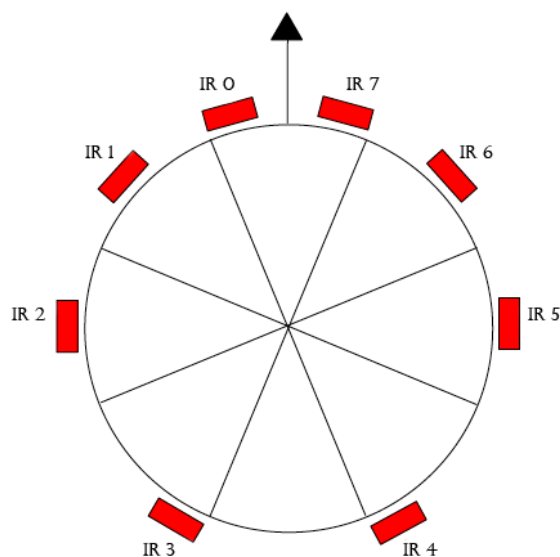


Figure 5.1: *E-puck light and proximity sensors disposition.*

on the other hand, are the nodes whose values change with the network dynamics. They assume a value according to their Boolean function and wire-scheme. This coupling phase is experiment specific, for this reason we describe it in each experiment section.

The simulation process is time discrete, it is organized in a sequence of *ticks*. The updating of inputs/output nodes is made at each tick. In particular, in a simulation step, the simulator engine takes the new data coming from the sensors. This data represents the new value assumed by the network's input nodes. After this, the network performs an update, and the value assumed by the output nodes is used to set the robot actuators.

Regarding the optimization algorithm, we use in both the experiments a stochastic descent algorithm. A general scheme is displayed in Figure 5.2.

In order to apply this algorithm to our task, we need to instantiate its problem-dependent components: the solution representation, that is, a state in the search space, the objective function and a suitable neighbor definition.

A state in the search space is represented by a BN. An initial solution s is a random Boolean network with defined n , K and p . The neighbor defines the modification, or *move*, performed on the current solution. For both experiments, first we randomly choose a node func-

```

INPUT: A SOLUTION  $s$ , AN OBJECTIVE FUNCTION  $F$ , A NEIGHBOUR
DEFINITION  $\mathcal{N}$ 
 $s_{best} \leftarrow s$ 
 $\nu \leftarrow \mathcal{N}(s_{best})$ 
repeat
  randomly pick a neighbour  $s_0 \in N$ 
  if  $F(s_0) < F(s_{best})$  then
     $s_{best} \leftarrow s_0$ 
     $\nu \leftarrow \mathcal{N}(s_{best})$ 
  end if
until timeout
return  $s_{best}$ 

```

Figure 5.2: *Stochastic Descent general scheme.*

tion, then we flip a bit in its Boolean function. The objective function depends on the specific task to be accomplished, therefore it will be described separately for each case study presented. In the problems that will be discussed, the goal of the search is to minimize the objective function, which can thus be considered as an error function. We also bring a little modification to the algorithm acceptance criterion. In fact, a solution is chosen if $F(s_0) \leq F(s_{best}) + \epsilon$ ($\epsilon = 0.0001$).

5.2 Phototaxis

The robot's task consists in the realization of phototaxis, i.e. a movement in the light source direction. The task is one of the most popular in the robotics field, in particular in the field of evolutionary robotics.

A concrete example is presented by Di Paolo [5] in which a population of 30 robots is evolved using a standard genetic algorithm with truncation selection. In truncation selection the candidate solutions are ordered by fitness, and some proportion P of the fittest individuals are selected and reproduced $1/P$ times. The experiment consists of two independent evolutions. Each of them is characterized by the sequential presentation of two different light sources. Only one source at a time is proposed to the robot for a period t chosen at random from the system. Robots are equipped with two light sensors (their simulated model includes noise) and two motors. The motors are controlled by a neural network. The neural network consists of six nodes and is fully connected except for self connections. The whole system is simulated.

With our stochastic descent and RBN we want propose an alternative the paradigm based on genetic algorithms and neural networks, for single-task robot learning.

5.2.1 Robot setup

Below we describe the robot setting used during the simulation.

- **Sensors:** In order to perceive the light source and understand the position in which it is located we use the eight light sensors.
- **Actuators:** In order to be able to move toward the light source, the robot uses the two wheels, activated by the two motors. The maximum speed is 15 m/s. For the experiment the motors that control the wheels may take only two values, ON and OFF. ON corresponds to a constant wheel speed of 5 m/s, while OFF to the wheel speed of 0 m/s, i.e. the robot doesn't move.

5.2.2 Boolean network setup

In this experiment the input nodes receive the information from light sensors that represent a light intensity perceived by the robot. In order to maximize the number of internal nodes, without increasing the network size, we gather the light sensors in pairs. Each pair corresponds to one input node. When at least one of the two sensors perceives a light variation over a chosen threshold, the corresponding input node assumes the value 1. In Figure 5.2 we report the input mapping.

| | Input nodes | Value | IRs state |
|--|-------------|-------|-----------------|
| L I G H T S E N S O R S | X0 | 0 | IR0 and IR1 OFF |
| | | 1 | IR0 or IR1 ON |
| | X1 | 0 | IR2 and IR3 OFF |
| | | 1 | IR2 or IR3 ON |
| | X2 | 0 | IR4 and IR5 OFF |
| | | 1 | IR4 or IR5 ON |
| | X3 | 0 | IR6 and IR7 OFF |
| | | 1 | IR6 or IR7 ON |

Figure 5.3: Mapping between light sensors and BN's input nodes.

In Figure 5.3 we show the mapping between the wheels and the output nodes. The mapping used provides that when the output node, associated with one wheel, takes the value 1, the wheel is moving at a constant speed of 5 m/s, otherwise the wheel stops.

| | Input nodes | Value | Wheels state |
|----------------------------|-------------|-------|-----------------|
| W H E E L S | X4 | 0 | LEFT wheel OFF |
| | | 1 | LEFT wheel ON |
| | X5 | 0 | RIGHT wheel OFF |
| | | 1 | RIGHT wheel ON |

Figure 5.4: *Mapping between wheel actuators and BN's output nodes.*

5.2.3 Training

The environment consists of a square arena (5x5 m) in which a light source is placed. This source is located in one of the 4 corners of the arena at a 1 meter height from the ground. We set the light color to yellow.

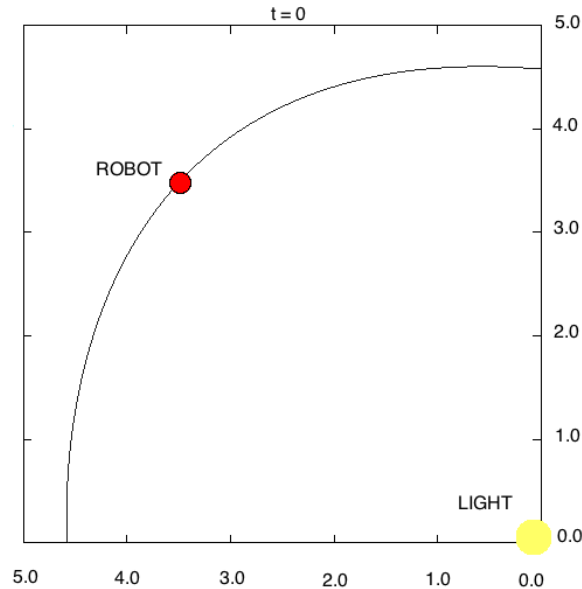


Figure 5.5: *Arena used for the robot training in the phototaxis task.*

The experiment consists in a set of trials in which the position of the light source is fixed. On the other hand, the position and orientation of the robot is changed in each single trial. We decided to place the robot at 4.5 meters away from the light (which is in the origin), with z axis angle among 15 and 75 degrees, with steps of 12 degrees. Each of these points corresponding to two different random robot orientations: the first one between 0 and 144 degrees and the other between 180 and 324 degrees.

During the training process we evaluate the network by evaluating the robot performance. We describe the robot performance minimizing the error function ($E \in [0, d]$) displayed in 5.1.

$$E(\Pi) = d \tag{5.1}$$

The term d represents the distance between the robot final position and the light at the end of each trial. The more the robot moves close to the light, the better is the performance. Whereas the arena size and the maximum speed at which the robot can move, a time of 120 seconds for a single trial allows it to achieve, starting from any initial configuration, the required objective.

Launching some experiments starting from a small group of networks we have observed that 1000 iterations of the optimization algorithm were sufficient to obtain networks able to successfully complete the task. At each iteration the optimization algorithm performs 10 trials which differ in the initial conditions. The performance is evaluated at each single trial. As a result we have 10 evaluations. The goal of the optimization algorithm is the minimization of the average value assumed by the error function during the all 10 trials.

5.3 Obstacle Avoidance

In this section we talk about an avoidance experiment. The Boolean network is trained to reach the ability to control a robot in a unknown space without colliding with objects. This task is more difficult then the phototaxis described int the Section 5.2 because the number of cases in which the robot could be operate is bigger and less predictable.

5.3.1 Robot setup

Below we describe the sensors and actuators used to perform the task.

- Sensors: In order to perceive the presence of an obstacle we use the eight proximity sensors able to detect an obstacle in a range of 4-5 cm.
- Actuators: The only actuators we need in this experiment are the two wheels that allow the robot to perform random walk in the environment.

5.3.2 Boolean network setup

In this experiment the input nodes receive the information from the proximity sensors that represent the presence, or not, of an obstacle. The mapping and the way in which this work is displayed in Figure 5.3 and 5.4. The only difference is that, for the sensor mapping, we use the proximity sensors.

5.3.3 Training

The environment for this experiment consists in a corridor 6.5 meters long and 0.5 meters wide. The corridor is one of the typical shape used for this kind of task because reduces the robot movements. It forces the robot to run in a very small space in which is impossible do not encounter a wall. So, if the robot is able to exit from the corridor, it is also able to avoid the walls.

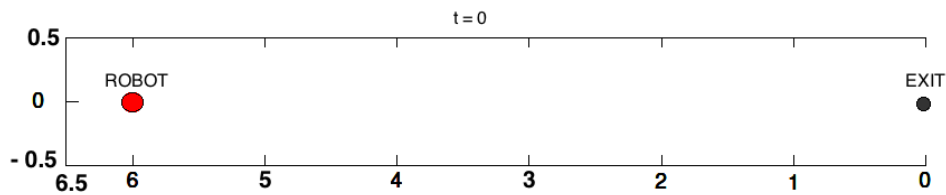


Figure 5.6: *Arena used for the robot training in the obstacle avoidance task.*

The experiment consists of 6 trials. In each of these, the robot position is always the same. It is placed 6 meters far from the end of the corridor corresponding to the origin of the axis. The initial rotation is chosen among 120 and 240 degrees with steps of 24 degrees. The time available in a single trial is 120 seconds, dimensioned in order to allow the robots to reach the corridor exit starting from any initial condition.

The evaluation of the network is done by evaluating the robot performance. We describe the robot performance minimizing the error function ($E \in [0, d]$) displayed in Figure 5.1. The more the robot moves towards the exit without collision, the better is the performance. To this end is important to underline that, if the robot hits a wall, the specific trial is immediately stopped without waiting the expired of 120 seconds. This guarantees that the robot doesn't reach the exit just crawl a wall.

For this experiment we execute 1000 iteration of the optimization algorithm. At each iteration the robot is evaluated on the 6 trials. As a result we have 6 evaluations that correspond to the same number of robot performance. The goal of the optimization algorithm is the minimization of the worst case value assumed by the error function during the all 6 trials.

5.4 Results and analysis

The analysis we conduct in this section focuses on studying the state space of a network trained to achieve a target behavior. In particular we are interested in analyzing how the network uses and organizes its state space depending on task and network size. We also provide a graphic representation of the entire state space to better underly the structures that emerge from it. Finally we represent the same state space by using a explicit finite states automaton.

Analyzing the results we will often talk about the three network regimes. In this regard it is necessary to specify that the network's Boolean functions are randomly changed during the training process, therefor even the network dynamics changes. For this reason, the dynamics of the final solution could belong to a different regime compared with that of the initial network in the optimization process. Nevertheless, we can classify the final networks using the starting dynamics regime because the difference showed by the results demonstrate that the initial regime properties influence also the final solution.

To provide a more thorough analysis we compare results obtained by training networks with different size and so, with different state space size. In particular we study the case of network with $N=20$ and $N=40$.

Starting with the phototaxis experiment with $N=20$ we can observe the final error values in Figure 5.7 left. Because of task sim-

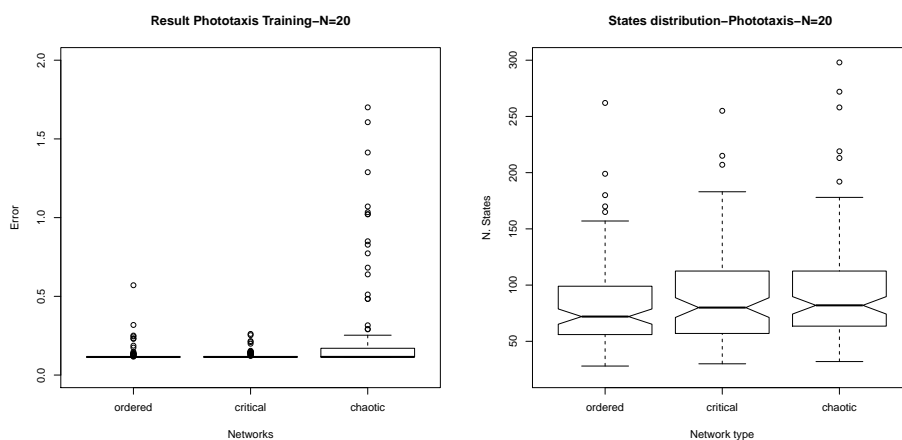


Figure 5.7: *Phototaxis experiment with $N=20$ and 1000 algorithm iterations. Left: final error value. Right: state space usage.*

plicity, we obtained a very good results with a final error value very close to zero. Now we want to observe the state space portion used to perform the phototaxis task. So we count the number of states used by each network. In order to do this, we need to make some assumptions. In the state space used by the trajectory, we could encounter some transitory states. It means that they are not essential in the task realization, but they are just used to pass from a state to another. In addition, it could happen that some states are used in just few trajectories. Again, we assume that these states are not important for the task achievement. So that, we introduce two arbitrary thresholds. A state is counted if it is visited at least the 20 percent of the average visit number in a single trajectory or if the state is present at least in the 20 percent of the trajectories. The collected data are plotted in Figure 5.7 right. What emerges from the graphics is that the number of states is very low compared with the entire state space. In fact, the networks use about 70-80 states when the number of available states has a dimension of 2^{20} states. This confirms that the networks do not use of the entire space, this also suggests that the state space must be organized according to some mechanism that allows the networks to use just the portion needed to carry out the task.

At this point we want to observe what happens if we double the network dimension ($N=40$) passing from 2^{20} to 2^{40} possible states. The final error value are plotted in Figure 5.8 left while the state space usage in plotted in Figure 5.8 right. What we can observe from the first plot is that we need to double the optimization algorithm

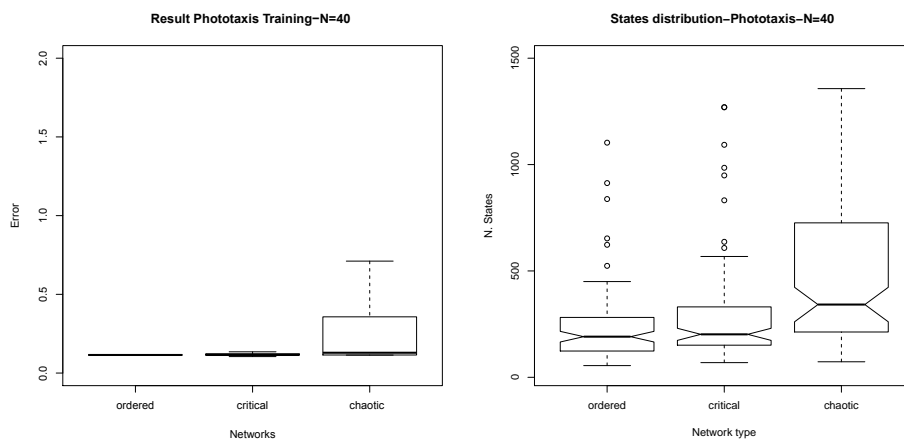


Figure 5.8: *Phototaxis* experiment with $N=40$ and 2000 algorithm iterations. Left: final error value. Right: state space usage.

iteration to achieve almost the same quality results. Actually, this is quite obvious if we think that increasing the number of nodes in the network we also increase the search space dimension. For this reason more iterations are necessary to find a solution. What is less obvious and even more interesting is that also the number of used states are very small compared to the entire state space size. In fact, increasing the state space by a 2^{20} factor the used space is about double for the ordered and critical regime and about triple for the chaotic one compared with the $N=20$ case. This confirms the results obtained with $N=20$.

Another interesting element that the plot in Figure 5.8 suggests is a presence of a relationship between the quality of the results and the state space use. In fact, what we can observe is that the best results belong to networks that use a small portion of their state space, i.e. ordered regime. Vice versa, networks that use a bigger number of states, such as the chaotic one, tend to obtain worst results. This suggests that the ordered dynamics moves in a more compact way within the state space. The network trajectory avoids to explore space in which there are not any good solutions. This could also explain why there is a difference between the results in different network regimes. While in the case of $N=40$ this difference is quite clear, for $N=20$ the medians of the three state distribution are very close. For this reason we decide to analyze the distribution of the number of states using the Wilcoxon test. The Wilcoxon test is a nonparametric test that compares two paired groups. The test calculates the difference

between each set of pairs and analyzes these differences. If the test returns a value (p -value) less or equal to 0.05 we can assert that the two compared distributions are different. The resulting p -value between the ordered and chaotic regime is 0.01994 that confirms a statistically significant difference between the distributions. This confirm the observation made for the $N=40$ case.

We made the same kind of analysis for the obstacle avoidance experiment both with $N=20$ and $N=40$. The results are showed in Figures 5.9 and 5.10. All the plot confirm the observation made for the phototaxis experiment. The state space portion used by the networks is very small compared with the the size of the state space. In addition, because this task is more difficult than the phototaxis one, the relationship between the quality of the results and the number of the states used is more evident. In particular we can observe that in the ordered regime, in which the number of states is low in both configurations, the error value continues to be very close to zero. Differently, for the chaotic regime, it is more difficult to keep the number of used states low, and also the error value assume value away from zero.

All these observations suggest the presence of a "compression mechanism" inside the state space and that it is more efficient in the ordered regime compared to the chaotic one. Such compression allows the network to map its behavior in a very small portion of the state space. This compressed space composes the internal knowledge representation, or rather the concept of memory described in Chapter 4. The low number of states implies that the trajectories within the state space do not need to visit a large number of states to perform the target behavior. This means that the reliability of the system is very high and the access to its internal memory very fast.

So far, we have analyzed the network at the end of the design process. This gives us information about the final state space structure, but it does not give us any information about how this state space evolves during the optimization. In particular, we want observe how the states compression mechanism works during the design process and what is its role during task learning. In order to answer to these question we trained 300 random networks as described in Section 5.2.3 and 5.3.3. At each iteration we collected the trajectory of the chosen network. In this way it is possible to have a dynamic history and observe the changing that have led to the final solution. It is important to underline that we want observe the generic trend of the three regime and not the specific states number. For this reason we plot a repre-

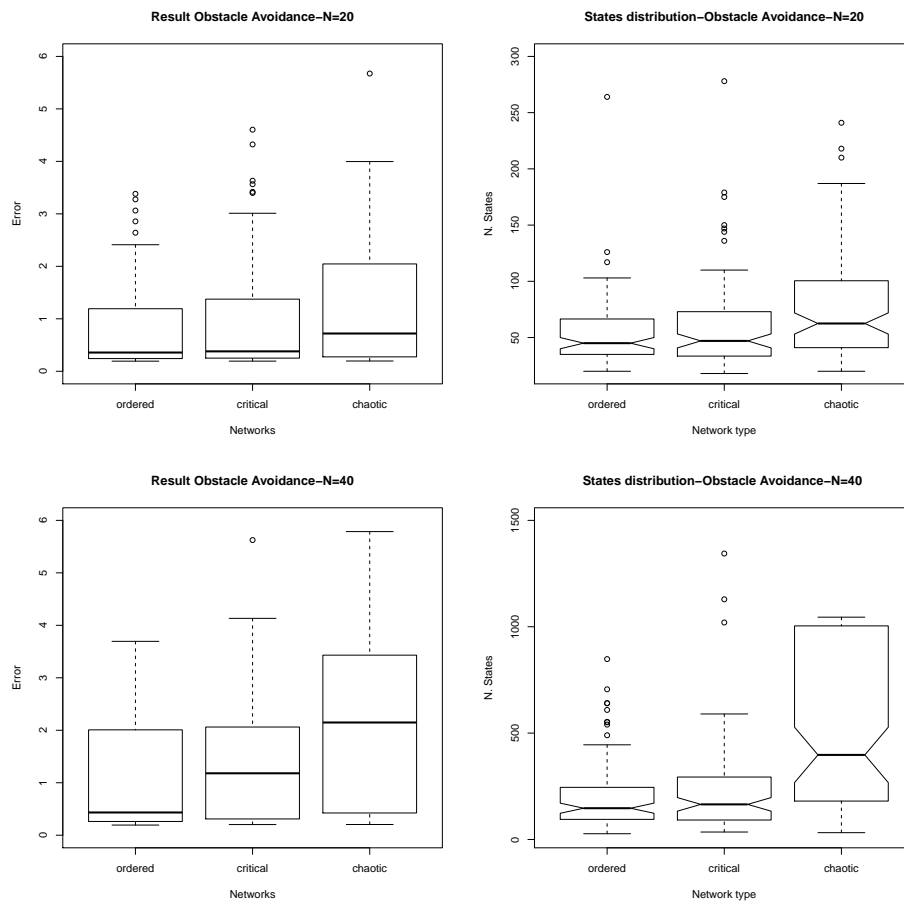


Figure 5.9: *Obstacle avoidance experiment. Final results (left) and state space usage (right). Top $N=20$ and 1000 algorithm iterations. Bottom: $N=40$ and 2000 algorithm iterations*

sentative sample of the 300 initial networks. The results are showed in Figure 5.10. The figure shows that the critical and chaotic regime

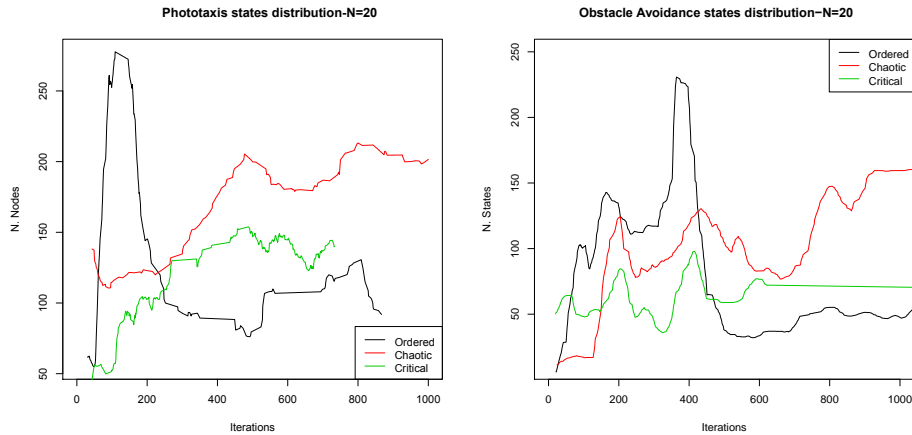


Figure 5.10: *States distribution in the phototaxis and obstacle avoidance experiment during a training process. Left: compare between the three network regime in phototaxis. Right: compare between the three regime in obstacle avoidance.*

have a similar trend. Starting from a random number of states, the training process brings the two dynamics to explore an increasingly large state space. After a good solution is found, the dynamics tends to stabilize using approximately the same number of states used to find the solution. The main difference between the two regimes is in the final number of states. In fact, the chaotic regime confirms to be the regime that uses of the highest number of states and that, usually, reaches the worst final results.

The ordered regime, instead, is very interesting to analyze, not only because the ordered networks are the ones with the highest rate of success, but also because their trend suggests a more complex behavior. We can distinguish the training process in two phases. In the first one the network makes an exploration process. It starts from a very low number of states and suddenly it increases them until a peak is reached. Because the ordered dynamics is difficult to perturb, the network needs to increase the number of states to increase the chance to find a good solution. After this, the network continues with an exploitation process. Starting from the peak, the number of states quickly decreases. This phase occurs when the network finds a right solution and starts with the state compression: the states are grouped

together and the number decreases until the dynamics finds a configuration with a low number of states that allows the network to perform the task. Thanks to this process the network dynamics becomes more robust. The results in Figure 5.10 right confirm the same results even for the more complex experiment of the obstacle avoidance.

To confirm the relation between the two phases, exploration and exploitation, and the final solution, i.e. error value, we can plot in the same graphic the two distributions. In order to this, we chose 30 ordered networks and during the training process we collected, at each iterations, the number of the used states and the corresponding value assumed by the error function. The results are showed in Figure

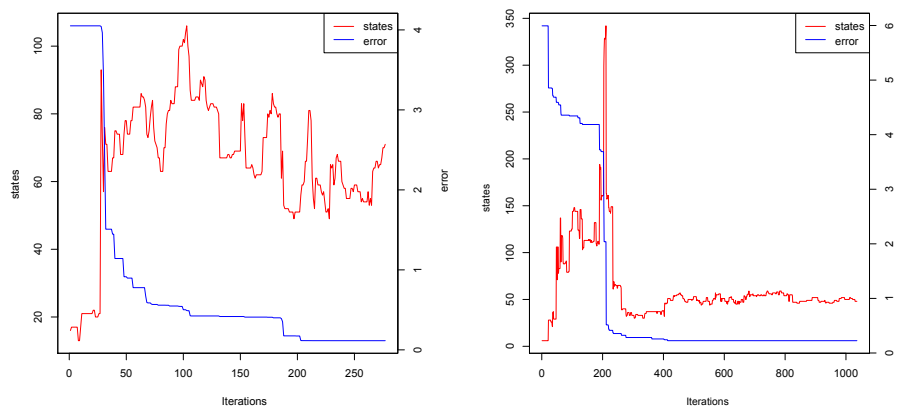


Figure 5.11: *Comparison between states distribution and error function during a training process. Left: Phototaxis. Right: Obstacle Avoidance.*

5.11. Because the phototaxis task (Figure 5.11 left) is a quite simple task, the exploration phase does not need to increase too much the number of used states. According to this, even the exploitation phase does not drastically reduce the number of states. The used space stay low for the entire training process. Nevertheless, we can observe that, during the exploration phase we have a quick decrease of the error value. This is more visible in the obstacle avoidance case (Figure 5.11 right). The plot shows how, in correspondence to the exploration phase, the error function quickly decreases its value. This confirms that, increasing the state space size we also increase the number of chances to find a good solution. With abuse of terms we could say that, in the exploration phase the ordered regime becomes chaotic to better explore the entire state space. This phase continues until

the error function assumes values very close to zero, which means that a good solution was found. After this, the regime returns to be ordered and the solution is refined by the exploitation phase. The improvement performed on the error value are very low, conversely, the number of states decreases significantly. The state space becomes smaller to confirm the "states compression" capability.

Now that we have analyzed the states compression mechanism and we have confirmed that the network behavior is mapped in a very low number of states, we want give a graphical representation of the state space and analyze the structures formed inside it. It is important to underline that this analysis step is possible due to the low number of states and to the their binary representation. In fact, more complex models such as neural networks, it is impossible to represent the state space due to the mathematical complexity of the model. So that it is practically impossible to describe in depth the structure of the state space. Conversely, with the Boolean networks we are able to accurately represent and analyze the behavior of a network.

In order to do this, we collected 200 trajectories during an experiment and plotted Figure 5.10. We decode each possible state to make it more comprehensible and we plot the graph that represents the state space used by the trajectory. To further reduce the number of states and make the representation clearer, we group together states with the same input/output value without consider the remaining state configuration. The results are showed in Figure 5.12. Each graph consist of a group of ellipsis that represents a single state in the state space. The number on the arrows represents the probability to move from a state to another. Inside each ellipsis we can find a string that summarizes some state's information. It consists of three different sections separated by the symbol " / ". They represent, from the left, input/output decoding, total number of visits for the single state and total number of times that the state appears in a trajectory. The input/output decoding consist of two strings separated by the symbol "-". The first on the left indicates a cardinal point from which the robot sensors are reading some data. The north direction corresponds to the front of the robot. The one on the right side, instead, represents the actuators, or rather the wheels.

What emerges from the plot is that the state compression groups together states with the same behavior. This causes the emergence of hierarchies. Also, this confirms the idea described in Chapter 4 where we explain that hierarchies represent the simplest way to organize the internal knowledge and, in the same time, they increase the network

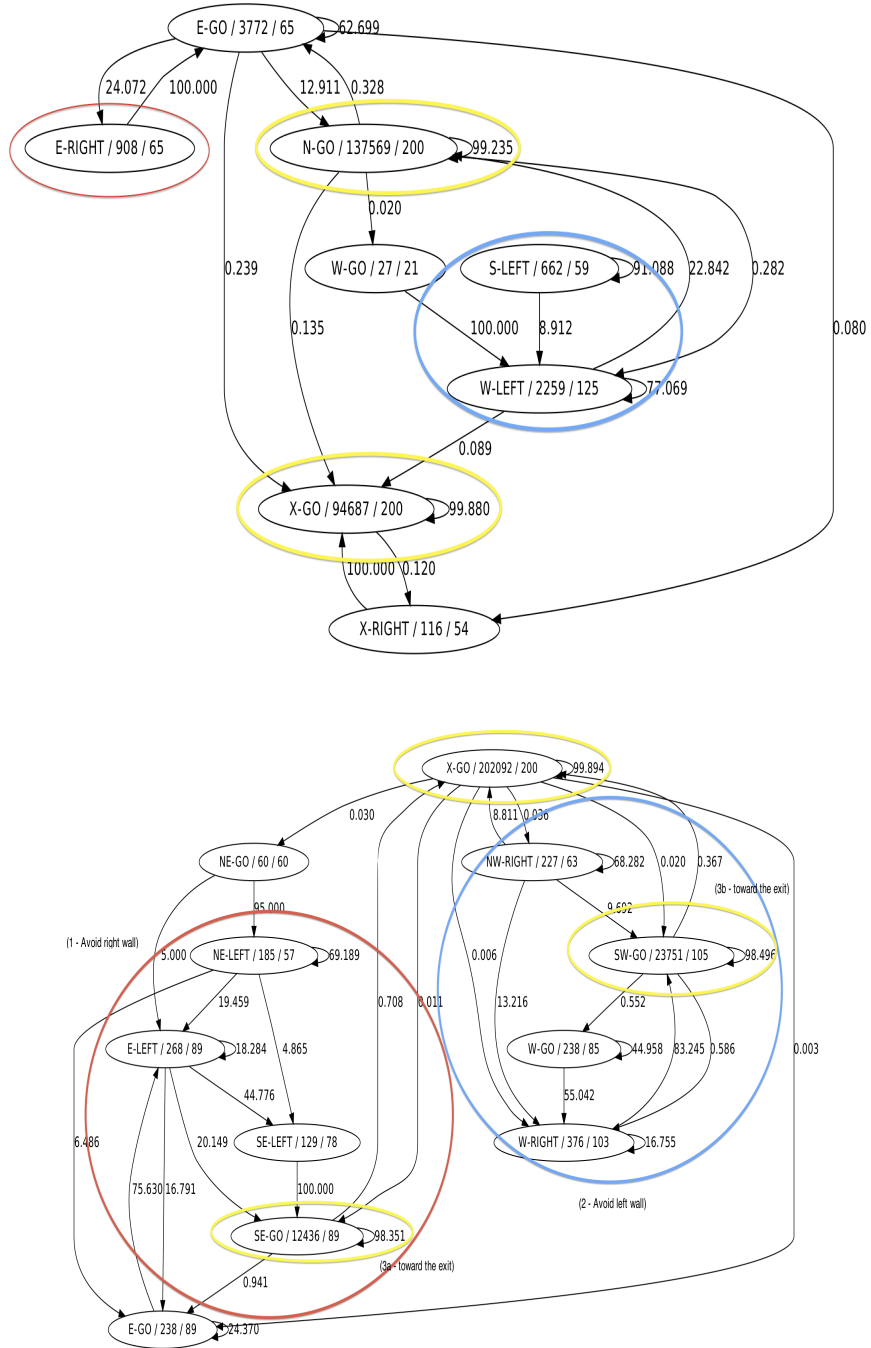


Figure 5.12: Network trajectory graph over the state space. Top: Phototaxis. Bottom: Obstacle Avoidance.

stability and reliability.

Another important observation can be made on the elements that characterize the network behavior. The general idea is that the dynamics of a complex system is characterized by its attractors and all the information concerning the network behavior could be extracted by studying the structure of the state space's attractors. What emerges from the plots 5.12, instead, confirms that this is not true. In fact, the network behavior is not described just by attractors, but also by the states that compose the transient tree that leads to the attractor itself. This confirms what said in the Chapter 4 where we described that the network knowledge is "stored" in the attractors and their basin. So that the entire state space contains a large amount of informations that are not possible to observe just with attractors.

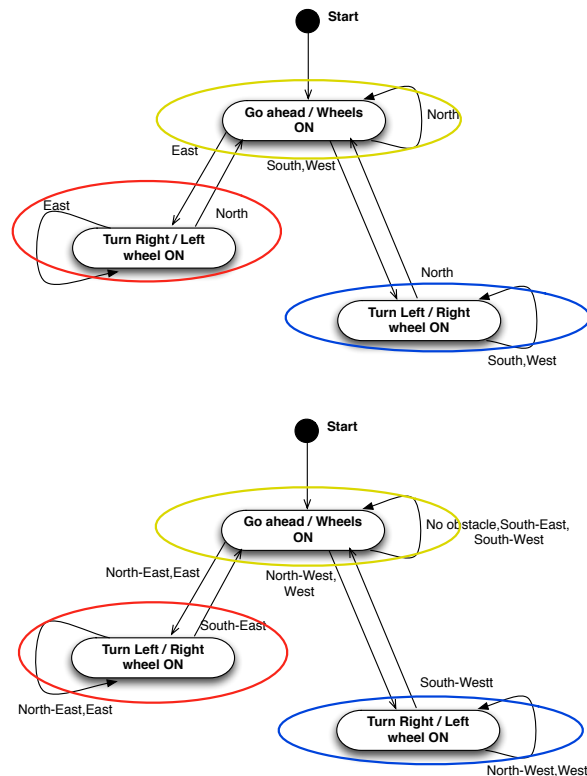


Figure 5.13: *Finite states automaton representing phototaxis (top) and obstacle avoidance (bottom) task.*

Now we focus on the role of the states that compose the network behavior. The phototaxis task is clearly represented by two macro actions or categories of actions: orientation toward the light and move-

ment toward it. The first action is performed by two different groups of states (red and blue circle). When the robot senses the light source in the east position (E) it performs a right rotation, in the same way, when the robot perceives the light on the west (W) or south (S) position, it performs a left rotation. The rotation continues until the robot perceives the light source from the north (N) direction. It indicates that the robot maintains a certain type of behavior until the final condition is not satisfied. This kind of logic belong to the "while loop logic". It makes the robot behavior more robust and able to generalize a task. The second action, i.e. the movements toward the light, is performed by the two states in the yellow circle. Actually, they also represent the two network attractors in which the robot continue to move toward the light. The obstacle avoidance diagram, Figure 5.12 bottom, confirms what said for the phototaxis one. The network achieves the task by using two regions of the state space. The one in the blue circle is used to avoid obstacles on the left side by performing a clockwise rotation, while the area in the red circle is used to avoid obstacle in the right side. When the robot does not perceive any obstacle (identify by the symbol X) the dynamics converge to the three attractors in the yellow circle. They allow the robot to go ahead.

What emerges from this analysis is that the network behavior is represented in the state space in a very straightforward way. The state set contains the description of all the possible cases that the robot could encounter during the task performance. In other words, the network follows a proper finite state automata to perform the target task. In order to prove that, we propose it in Figure 5.13.

Chapter 6

RBN Sequence learning

In this Chapter we discuss a sequence learning problem. This is a more difficult task compared to the two described in Chapter 5. The difficulty of the sequence task learning is in the fact that the network have to dynamically show a given behavior depending on what happened in its past. In our case, the possibility to recognize if a color is in sequence or not depends on the color that the network had visited in the past. This makes necessary the presence of a memory concept.

Our goal is to make use of this task to test our solution methodology and confirm the results presented in the Chapter above. In addition, we deeply analyze the concept of memory in random Boolean network analyzing the network's state space.

6.1 Sequence learning

Sequence learning has always been one of the most used learning methods by human and animals. It is part of human's abilities, and it is essential to the development of intelligence. Sequences of action or information can be found in task daily performed. We could think, for example, to the sequential movements of a musician who plays an instrument, or the sequence of actions that leads a man to drive a car.

Actually, the interest in the ability to learn and the usage of sequences, for processing information and actions, is by no means a recent phenomenon. Despite this, sequence learning already plays a fundamental role in all application domains in which it makes use of intelligent systems such as planning, speech recognition, DNA sequencing and robotics. While there is a variety of motivations for experimentation in sequence learning three main reasons characterize this interest:

- Sequencing of information and actions is a fundamental human ability.
- Sequence learning is an easily studied example of skill acquisition.
- Sequence learning may be a complex form of implicit learning. With implicit learning we mean the activity of learning complex information in an incidental manner, without awareness of what has been learned. It may require a certain minimal amount of attention and may depend on attentional and working memory mechanisms.

There are several categories of sequence learning problems, and these categories are distinguished by the goal of the given problem. Below we propose a list of the main sequence learning problem.

- *Sequence prediction:* $s_i, s_{i+1}, \dots, s_j \rightarrow s_{j+1}$. Given a s_i, s_{i+1}, \dots, s_j sequence we want to predict the s_{j+1} element. If $i=1$ we want to predict the new element of the sequence based on all the previous elements in the sequence. If $i=j$ we want make a prediction based just on the last element of the given sequence.
- *Sequence generation:* $s_i, s_{i+1}, \dots, s_j \rightarrow s_{j+1}$. The formal definition is essentially the same used to define the sequence prediction. The difference is in the fact that we do not want to recognize the element s_{j+1} , but we want generate it.
- *Sequence recognition:* $s_i, s_{i+1}, \dots, s_j \rightarrow \text{yes or not}$. Given s_i, s_{i+1}, \dots, s_j we want to recognize if the subsequence belongs to a target sequence.
- *Sequential decision making:* $s_i, s_{i+1}, \dots, s_j; s_G \rightarrow a_J$. Given a sequence s_i, s_{i+1}, \dots, s_j we want to chose an action a_J at time step j that eventually allows us to achieve the goal s_G . For the trajectory oriented problems, given s_i, s_{i+1}, \dots, s_j and the next state s_{j+1} , we want to select the right action a_J at time step j that allows us to reach the desiderated state s_{j+1} .

How humans learn sequential procedures has been a long-standing research problem in cognitive science and currently is a major topic in neuroscience. Research work has been going on in several disciplines, including artificial intelligence, neural networks, and engineering.

A concrete application, in artificial intelligence field, is given by Brian M. Yamauchi in his article [4]. The cases study proposed in this article belong to sequence generation category. The aim of his work is to explore the use of genetic algorithm to evolve continuous-time recurrent neural networks capable of sequential behavior and learning. He evolves networks able to generate a fixed output sequence in response to an external trigger occurring at varying intervals of time. He also evolves networks that can learn to generate one of a set of the possible sequence. The learning mechanism is based on reinforcement learning, in which the networks optimize their behavior in the face of rewards and punishments. This work display that small (3-9) continuous-time recurrent neural networks are capable of solving simple sequence generation and learning task.

6.2 Task definition

The case we want analyze belongs to sequence decision making category. The aim of our task is to obtain a network able to learn and recognize a sequence. The sequence is composed by the color black and gray, and it consist in their ciclic repetition. The background color (white) of the floor in which they are placed is used as separator. So that, a color is always followed by a white space.

The network represents a robot controller. The robot must be able to move on the colors and turn on its LEDs when it encounters a color in the right sequence position. Vice versa, when it encounters a color not in sequence, or a color that not belong to the sequence at all, the LEDs must be off.

The peculiarity of this task is that the network must remember which was the color previously encountered to be able to recognize the next color in the sequence. This brings out a strong concept of memory. Is important to emphasize that, we don't provide to the network any explicit memory structure. To perform this sequence recognition task, in fact, the network must be able to organize its internal structure in such a way to develop its addressable memory.

6.3 Robot setup

In this section we describe the robot's characteristic in terms of hardware configuration. In particular we describe all the actuators and sensors whereof the robot is equipped with.

- **Sensors:** The main thing the robot must be able to do is recognize a color. For this reason we have equipped it with one ground sensor that can recognize grayscale, from black (0) to white (1). We assume that a value less than 0.3 represents the black color, a value bigger than 0.6 represents the white color while a value between 0.3 and 0.6 is the gray color. Actually, the robot has also another kind of sensor, the proximity sensor. This one is not strictly necessary to the fulfillment of the task, but may be useful to the robot depending on the environment in which it is trained. For example, it could be necessary to avoid some obstacle in order to move on the colored area without getting stuck. These sensors are arranged as shown in figure 5.1.
- **Actuators:** In order to be able to move, the robot has two wheels used in the same way described in section 5.1.1. In addition, to signal when the robot recognizes a color in the right sequence, it is equipped with eight LEDs.

6.3.1 BN setup

Because of the high complexity of the task and the high number of input nodes, a network with 30 nodes can be a good deal. The set of the network used in this experiment is the same described in section 5.1.

| | Input nodes | Value | IRs state |
|---|-------------|-------|-----------------|
| P R O X I M I T Y | X0 | 0 | IR0 and IR1 OFF |
| | | 1 | IR0 or IR1 ON |
| | X1 | 0 | IR2 and IR3 OFF |
| | | 1 | IR2 or IR3 ON |
| | X2 | 0 | IR4 and IR5 OFF |
| | | 1 | IR4 or IR5 ON |
| | X3 | 0 | IR6 and IR7 OFF |
| | | 1 | IR6 or IR7 ON |

Figure 6.1: Mapping between proximity sensors and BN's input nodes.

After the network generation, we make the robot-network coupling. First we want to map the proximity sensors. Because the presence of obstacles is not an essential information in order to perform sequence

| | Input nodes | Value | Color |
|----------------------------|-------------|-------|-------|
| G R O U N D | X4 | 0 | BLACK |
| | X5 | 0 | |
| | X4 | 0 | GRAY |
| | X5 | 1 | |
| | X4 | 1 | WHITE |
| | X5 | 1 | |

Figure 6.2: Mapping between the ground sensor and BN's input nodes.

recognition, we chose to have one input nodes for two proximity sensors. So, at the end, we have the eight proximity sensors mapped in just four input nodes. This enables us to save some internal nodes and, in the same time, have a quite good information about the obstacle presence. The second sensor we want to map is the ground sensor. It must be able to recognize three different situation: when it is on a black, gray or a white area. For this reason we map it into two input nodes. These two nodes assume a different value configuration according to the color type that the sensors can perceive from the ground. All the sensors mapping are displayed in figure 6.1 and 6.2.

| | Input nodes | Value | Wheels state |
|----------------------------|-------------|-------|-----------------|
| W H E E L S | X6 | 0 | LEFT wheel OFF |
| | | 1 | LEFT wheel ON |
| | X7 | 0 | RIGHT wheel OFF |
| | | 1 | RIGHT wheel ON |

Figure 6.3: Mapping between wheels actuators and BN's output nodes.

| | Input nodes | Value | LEDs state |
|-------------|-------------|-------|------------|
| L E D | X8 | 0 | LEDs OFF |
| | | 1 | LEDs ON |

Figure 6.4: Mapping between the LEDs and BN's output nodes.

Regarding the actuators, we chose to map the wheels with two different output nodes, one for each wheel, while we map the eight

LEDs with a single output node. This because we are not interested to control each single LED. In fact, they always assume the same status: ON or OFF. In the figure 6.3 and 6.4 is possible to see the actuators mapping.

6.3.2 Training

The entire experiment is built and tested with the Argos simulator. The choice of the right environment in which train the robot is a quite sensitive problem. The presence of too much elements that do not stringently belong to the problem could dramatically interfere with the sequence learning task. In other words, we need to reduce the robot mobility and force it to pass over the colored area quite often. So, we chose to execute the train process in a corridor arena, 6.5 meters long and 0.5 meters wide, with a colored-striped floor. The corridor is the perfect shape to reduce robot mobility, moreover the striped floor force the robot to encounter the colored area without the necessity to perform a random walk to find them. Figure 6.5 shows the arena configuration used for the training process.

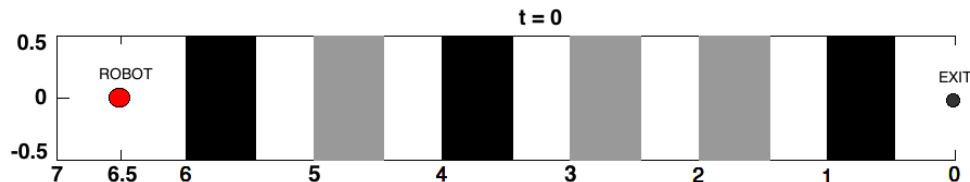


Figure 6.5: *Arena used for the sequence learning training.*

The experiment consists of 10 trials of 130 seconds each. The robot is placed in a white area within the corridor at 6.0 meters far from the exit, and oriented toward it. For each trial we propose, to the robot, a different sequence of color for the arena floor, keeping always the black and the gray area separated by a white one.

During the training process, the quality of a network is calculated by evaluating the robot performance. To evaluate the robot performance we must be able to know when it recognizes a color in sequence. For this purpose, we define a rule based on how long the robot takes on its LEDs. As mentioned in Section 5.1, the simulation process is time discrete, so that, the training process is shaped by a sequence of *ticks*. We use these ticks as unit of measure to count the time spent by the robot in a color. We assume that a color is detected as in sequence

if the robot takes on its LEDs a number of ticks greater than or equal to $2/3$ of the ticks that it spends on that color. Vice versa, it means that for the robot the color is not in sequence. The threshold value $2/3$ it's arbitrary, but the reason because we introduce it is intrinsic in the network dynamics. In fact, it could be possible that, the input information (the color of the area), can not be propagated inside the network in just one updating step. This brings the robot to have a wrong behavior, not because it fails the task, but because its outputs (LEDs status) need more then one update to change. In addition, when the trial starts, the network is randomly initialized. Therefore, even the output nodes assume a random configuration that we don't identify as a robot mistake. Because of this, providing a threshold we take into account this technical problem.

In the following we use the term *wrong color* to identify a color not in the right sequence position, and the terms *right color* to identify a color in the right sequence position. Now that we have a rule for the colors detection, we can define a training methodology. The idea is to stop a trial when the robot makes a mistake. The possible mistakes are:

1. When the robot does not switches on the LEDs for a $ticks \geq 2/3$ on a right color.
2. When the robot switches on the LEDs for a $ticks \geq 1/10$ on a wrong color (even the white stripe).
3. When the robot crash in a wall.

The first point describes the case in which the robot doesn't recognize the right color, the second point describe the case in which the robot turns on its LEDs too many ticks in a wrong color but without recognizing it. Actually, this behavior does not represents a real mistake, because the robot doesn't recognize the wrong color as in sequence. Despite that, we want a clear division among the behavior assumed by the robot. We want that, the number of ticks in which the LEDs are turned on, in the case of wrong color and right color, are significantly different. Finally, the last point could force the robot to adopt two different behaviors: or it starts to use its proximity sensors to achieve obstacle avoidance, or it starts to go ahead. For the sequence learning task we are not interested in which of the two options the robot takes.

At this point, we are ready to define the performance measure. We describe this value by minimizing an error function $E \in [0,1]$ as displayed in 6.1.

$$E(\Pi) = 1 - (\alpha \cdot (1 - d) + (1 - \alpha) \cdot \frac{\textit{Right ticks}}{\textit{Tot. ticks}}) \quad (6.1)$$

The main term is d . It represents the distance between the final robot position and the corridor exit. The final robot position could be either the position reached after the 130 seconds (the end of a trial) or the position after a robot mistake. We normalize this value between 0 and 1. This kind of evaluation brings out the concept of implicit learning. In fact, we don't reward the robot directly for the sequence recognition, but we push the robot to reach the corridor exit. For our training methodology, if a robot can reach the exit it is also able to recognize the sequence.

The second term is used to discriminate between robots that reach the same final position but that they have used the LEDs in a different way during the trial. We increment *Right ticks* for each ticks in which the robot takes on the LEDs in a right color and when it takes off the LEDs in the wrong one. At the end of a trial we normalize this value using the total number of ticks (*Tot. ticks*) in which the robot has lived. If the robot uses the LEDs in a perfect way the fraction assume the maximum value 1. Summarizing, the more the robot moves towards the exit without mistakes and using the LEDs in a proper way, the better is the performance.

The multiplier α is used to give more or less weight to the two terms. In our setting α assume the value 0.8.

Regarding the optimization algorithm we use several metaheuristic techniques. We optimize the error function using stochastic descent, variable neighborhood descent, iterated local search and genetic algorithm. A more detail study on the usage of these algorithm in this task refer to [3].

6.3.3 Results and analysis

The goal of this section is to analyze the network behavior by study its state space during the perform of a complex task. The idea is that, even if the task is more complicate compared with the two proposed in the Chapter 5, some basic concept regarding the state space structure can be confirmed. In addition, we deeply analyze the emergence of memory studying the structure of the state space.

As said before, sequence learning is not a trivial task due to the necessity of some memory structures. The task complexity has impact in two analysis aspects. The first one concerns the percentage of success. After a training process, depending on the optimization algorithm, the number of working networks is among 8 and 20 percent, while in the phototaxis experiment the percentage was around the 100 percent and in the obstacle avoidance was about 95 percent. This is important because our analysis focus on study working networks and their low number makes the analysis less general. The second is the difficulty to collect and manipulate the data. In fact, the optimization algorithm needs several iteration (around 100000) to find a good solution that imply an high number of hours for the computation. In addition, the data collected during the training process reach various dozen of gigabytes that means an high investment in terms of physical resource. The final result of each optimization algorithm is showed in Figure 6.6. Due to the low number of good solution the analysis is

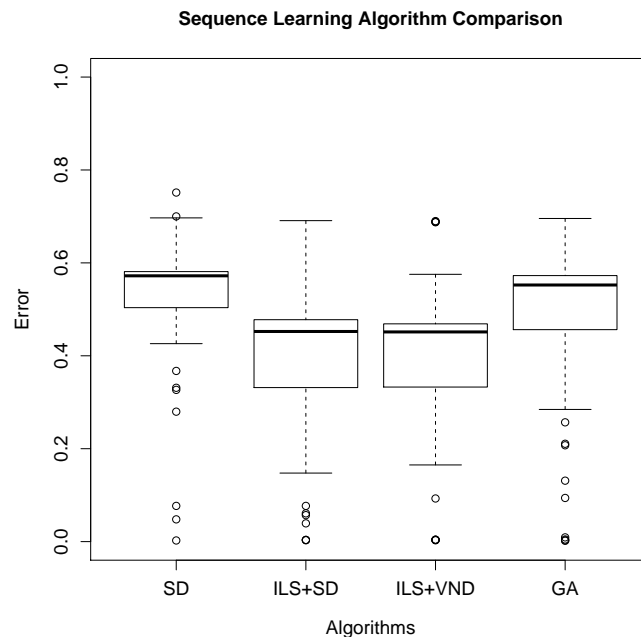


Figure 6.6: *Sequence learning algorithm results.*

not carried out in large scale, for this reason i allow myself in some conjectures and speculations.

The first aspect we want analyze is the state space usage during the training process. In order to be able to plot this information we

collected data regarding the number of states corresponding to each algorithm iterations using the same criteria described in section 5.4. In Figure 6.6 there are two characteristic trends regarding the states number associated with the optimization algorithm iteration.

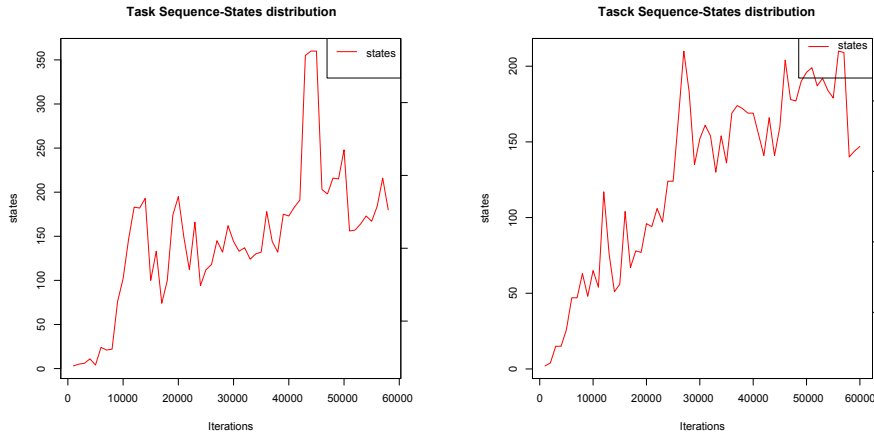


Figure 6.7: *Task Sequence states distribution during the training process.*

The trends showed by the two graphics is quite similar. We can observe that the number of states tends to increase during the training process, but the states number does not grows indefinitely. Both graphics display that the number of used states tend to stabilize with the increasing of the algorithm iterations. In addition the number near by it stabilizes is very low, we have in fact a value around 200 states for both the networks. If we think that the state space is composed of 2^{30} possible states this confirms that the states compression mechanism still present.

Another element that emerges is that the two trends are characterized by several peaks among which is possible define once higher than the other. The fact to have more then one peak could be explained by the difficulty of the task and the way in which the networks learn it. Sequence recognition could be seen as composition of subtask. For example when network learn to recognize the black color, some memory structure is created within the state space. When the network learns to recognize also the second color, the memory needs to change and even the state space is reconfigured. This suggest that we could have a peak for each memory modification, and the highest one could represent the moment in which the network brings the final modification to its memory structure that allows to recognize the entire sequence.

At this point it is interesting to observe the trend of the error function compared to the states number. For this purpose we plot the two value distributions in Figure 6.8. What we can observe is that

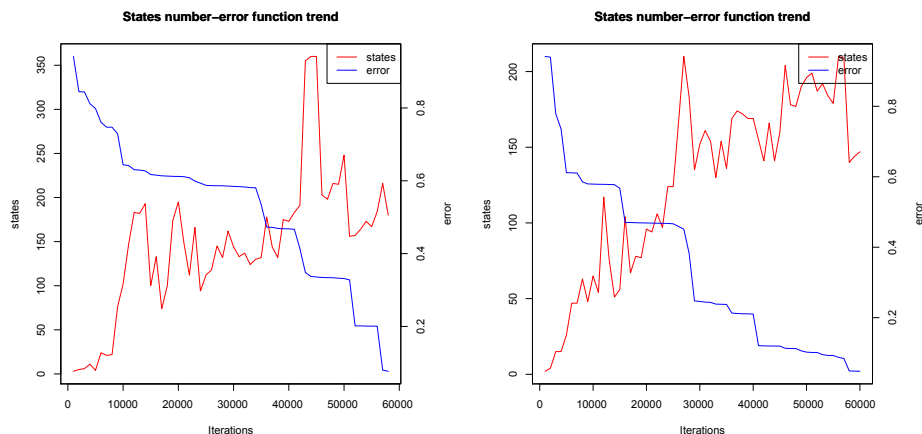


Figure 6.8: *Task sequence. Comparison between states distribution and error function values.*

the error function has a step trend. It could be explained with the memory modification described above. Each object function improvements corresponds to a new subsequence recognition. In particular, observing the value assumed by the two error function's terms, we can confirm that for an error value ≥ 0.6 the network is able to recognize the first sequence position, i.e. the robot turns on the LEDs if it encounter the black color, and for an error value ≥ 0.5 the network is able to recognize a sequence of length two, i.e. all possible configuration that the two colors can assume. Finally, with an error value ≤ 0.4 the network is able to generalize the two tasks above and recognize a sequence of any length. This last step makes the difference between working and not working network. Because this step corresponds to the moment in which the states number is higher, i.e. the peak, we are inclined to affirm that the exploration and exploitation phases are fundamental to the goal achievement.

Summarizing, the analysis suggests that the state space compression is still present in networks with good results. In addition, even the relation between the error function and the state space continue to appear during the training process. Finally, it seems that the exploration and exploitation phases have an huge impact for the memory structure creation. Now that we have analyzed the state space structure, we want to represent it in the same way adopted for the test cases. In

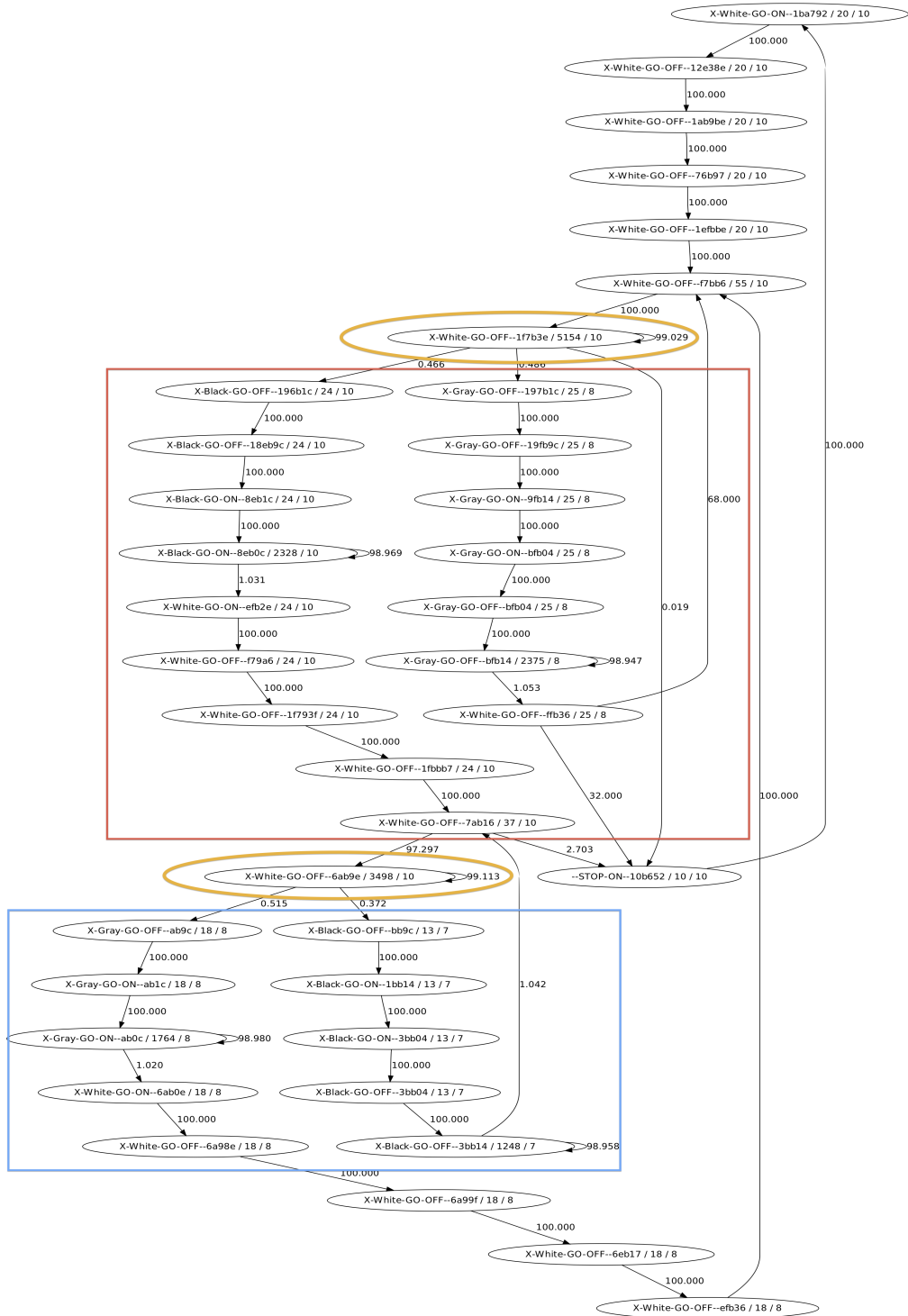


Figure 6.9: Task Sequence trajectory graph over the state space.

Figure 6.9 is showed the trajectory graph over the state space. The first thing that emerges from the plot is the very low number of states. This suggest that, during the training process, the network can both find a solution to the proposed problem and refines it thanks to the compression mechanism that can be seen as an optimization process. The optimization process produces also group of states that cause the emergency of hierarchy. Each hierarchy has a fundamental role in the target achievement. This confirm that hierarchies are the simplest way, for the network, to organize the internal knowledge and increase its stability and reliability. This also confirm that the network behavior can not be represented just by its attractors, on the contrary, the entire state space contains a large number of informations that could be lost by studying only the singles attractors.

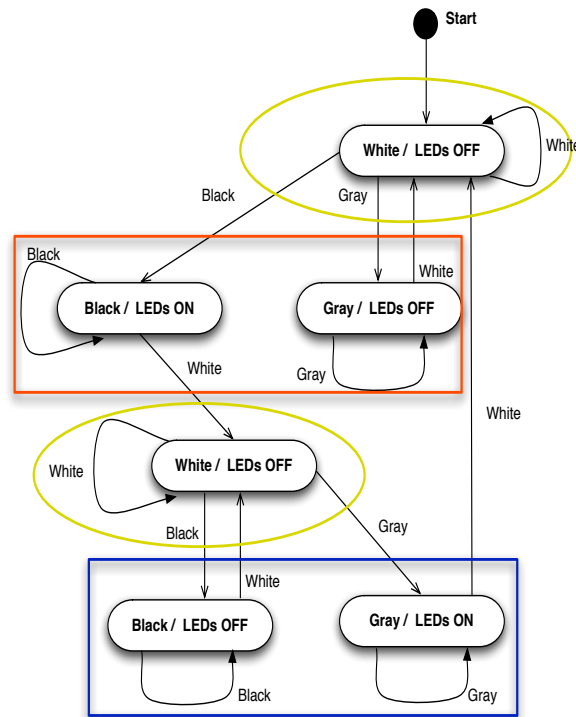


Figure 6.10: *Finite states automata for the task sequence dynamics.*

As said in the introduction, sequence learning needed the emergence of memory to be realized. In our case the recognition of a color depends on the color previously recognized. So, the network needs to organize its state space structure in such a way to generate the concept of memory. From plot 6.9 it emerges that the network attains

memory by duplicating some portion of its state space and placing it in the right position of the hierarchy.

The first consideration we can do is that even if the two portions have the same states structure they realize two different behavior depending on the position that they assume in the state space. In fact, the red rectangle achieves the black recognition while its duplicate, in the blue rectangle, recognizes the gray. The elements that discriminate which of the two behavior have to be performed are the states in the yellow ellipses. They can be seen as the decision point of the system. Even these state are duplicated and depending on the position that they assume in the state space they have a different meaning. In fact, the first ellipse on the top leads to the black recognition while its duplicate allow the gray recognition if and only if the black has already been recognized. Both, the decision and the color recognition portion, are reused from the trajectory during the sequence recognition. The entire system dynamics can be well represented by the finite states automata showed in Figure 6.10 that perfectly represents the state space just described.

Another consideration regards the duplication process. In fact, the mere states duplication would lead to an uncontrolled usage of the state space, with the effect to dramatically increase the states number. We could think to recognize a sequence of 100 colors. The duplication mechanism tends to duplicate the portion used to recognize a color hundred times. And what if we have a sequence with more colors? We could reach a situation in which the state space dimension could not be big enough to recognize the entire sequence. This suggest that the compression states mechanism makes a solution optimization in terms of state space usage but also it allows the resource reuse, and so, the task generalization.

What emerge from the analysis is that, the resource reuse together with the state space compression and duplication are the elements used from the networks to organize their state space and realize memory.

Conclusion

In this thesis we have first tested an automatic method to synthesize robot controller based on Boolean networks (BN) as robot agent program and metaheuristic techniques as optimization algorithm. Then we have analyzed the resulting networks focusing on the state space structure.

In order to validate the method and perform the first analysis we experimented two simple tasks: phototaxis and obstacle avoidance. We obtained good performance in a short computation time. The state space study has revealed that the networks tend to use a low portion of state space to achieve the target task and it has highlighted a relationship between the used states number and the results quality. In particular we have noticed that in the ordered regime, where the number of used states is lowest, we obtained better results compared with the chaotic regime in which the number is highest. Moreover, analyzing the states distribution during the design process we have noticed that the number of states undergoes a phase of exploration (i.e. states augmentation) and exploitation (i.e. states decrease) that suggest the presence of a "states compression mechanism". To make the analysis stronger we have executed the same two experiments with networks of different size, more precisely we doubled the networks size augmenting the state space of a 2^{20} factor. Even with this configuration the results have been confirmed in both the experiments.

Finally we plotted the entire state space to better represent its structures. The results have showed that the network's knowledge stored within the state space is organized in hierarchies that increase the stability and reliability of the network. This analysis has confirmed that the entire state space contains a large amount of information that is not possible to extract just by studying attractors. In addition, we have demonstrate how the entire state space can be easily represented by a finite states automaton that describes, step by step, the network behavior.

Because of the good results and the interesting observations re-

sulted by the analysis, we have decided to move on a more complex scenario. In particular we focused on a task (i.e. sequence learning) in which the robot is required to keep an internal memory to achieve a given goal. At the end of the design process we obtained networks able to perform the recognition of a sequence confirming the presence of memory structure within the network state space. Analyzing the states usage we have confirmed also the low number of states. Plotting the state space graphic we have observed that the network realizes memory by duplicates some portion of its state space and placing it in the right position of the hierarchy. This mechanism is realized by the states reuse together with the state space compression and duplication. Finally, even in this more complicate task the network behavior can be represented by an finite states automaton.

Summarizing, we have observed that the state space undergoes a states compression mechanism during the training process. This causes the emergence of behavior's hierarchies that characterize each single state space portion. The hierarchies organization allow a fast access to the internal knowledge and makes the dynamics more reliable. In addition, we have noticed that the attractors states give a partial information about the network behavior. In fact, the state space categorization happens even in the attractors transient tree. Finally we have observed the structure used within the state space for the emergence of memory. What has emerged is that the network realizes its memory structures by duplicating some state space portion and connecting it in a specific position of the state space. The duplication mechanism that is in contrast with the low states number confirm that the compression mechanism can be seen has an automatic optimization process realized within the state space.

Future work consists of performing the analysis is a more complicate task. Starting from the sequence recognition we will add more subtask such as obstacle avoidance and random walk, that is required to find the sequence's elements. The aim is to observe how the state space changes and organizes its structure to achieve more complex behavior.

We will also interested in analyze the state space structure of not working networks and compare it with the working one. In this way we could identify the dynamics properties that make possible the goal achievement. We also plan to use these pieces of information to make some improvements in the design method by forcing the process to recreate those structures within the network's state space.

Finally we will study the possibility to develop an automatic tool

to map the network behavior in algorithm exploiting its finite states automaton representation. This brings out the possibility to have more reliable and robust control systems due to the fact that they are the result of an automatic design process instead of a "by hand" implementation.

Bibliography

- [1] Andrew Wuensche: *Attractor Basins of Discrete Networks - Implications on self-organisation and memory*. October 1996 Revised Feb 1997
- [2] Ron Sun and C. Lee Giles: *Sequence Learning: Paradigms, Algorithms, and Applications*.
- [3] Lorenzo Garattoni: *Advanced stochastic local search methods for automatic design of Boolean network robots*
- [4] M. Yamauchi: *Sequential Behavior and Learning in Evolved Dynamical Neural Networks*
- [5] Ezequiel A. Di Paolo: *Evolving spike-timing-dependent plasticity for single-trial learning in robots*
- [6] Josh C. Bongard: *Spontaneous Evolution of Structural Modularity in Robot Neural Network Controllers*
- [7] Dario Floreano, Laurent Keller: *Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection*
- [8] Marco Frison, Nam-Luc, Tran, Nadir Baiboun, Arne Brutschy, Giovanni Pini, Andrea Roli, Marco Dorigo, and Mauro Birattari: *Self-organized Task Partitioning in a Swarm of Robots*
- [9] Stefano Benedettini, Mattia Manfroni, Marco Villani, Roberto Serra, Antonio Gagliardi, Carlo Pinciroli, Mauro Birattari, Andrea Roli,: *Elsevier Editorial System(tm) for Neurocomputing*
- [10] Arne Brutschy, Nam-Luc Tran, Nadir Baiboun, Marco Frison¹, Giovanni Pini, Andrea Roli³, Marco Dorigo and Mauro Birattari: *Costs and benets of behavioral specialization*
- [11] Mattia Manfroni: *Toward Boolean network design for robotics application*

- [12] Carlos Gershenson: *Introduction to Random Boolean Networks*
- [13] Daizhan Cheng y, Hongsheng Qi y, Zhiqiang Li y, Jiang B. Liu: *Stability and Stabilization of Boolean Networks*
- [14] Katsumi Inoue: *Logic Programming for Boolean Networks*
- [15] M. Giacobini, M. Tomassini, P. De Los Rios, and E. Pestelacci: *Dynamics of Scale-Free Semi-Synchronous Boolean Networks*
- [16] Christian Blum, Andrea Roli: *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*
- [17] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klap-tocz, S. Magnenat, J. C. Zuerey, D. Floreano, and A. Martinoli. *The e-puck, a Robot Designed for Education in Engineering*. In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, volume 1, pages 5965, Portugal, 2009. IPCB: Instituto Politecnico de Castelo Branco.
- [18] Holger H. Hoose, Thomas Stutzle: *Stochastic local search, Foundations and applications*.