ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

ADVANCED STOCHASTIC LOCAL SEARCH METHODS FOR AUTOMATIC DESIGN OF BOOLEAN NETWORK ROBOTS

Elaborata nel corso di: Intelligenza Artificiale

Tesi di Laurea di:
Lorenzo Garattoni

Relatore:
Prof. Andrea Roli

Correlatore:
Prof. *Marco Dorigo*
Dott. Ing. *Mauro Birattari*
Ing. *Carlo Pinciroli*

ANNO ACCADEMICO 2010–2011
SESSIONE III

KEY WORDS

Boolean Networks

Metaheuristics

Robotics

Contents

Acknowledgements

First of all, I would like to thank Andrea Roli, who has given me the opportunity to undertake this experience and for being always very kind and helpful. I thank also Marco Dorigo for giving me the possibility to work in IRIDIA and Mauro Birattari for his support. Many thanks to Carlo Pinciroli for his fundamental help. I am honored to have known you and worked with you.

I thank also the whole IRIDIA family: Alessandro, Alexander, Ali, Arne, Eliseo, Franco, Gianpiero, Giovanni, Manu, Giovanni 2, Gabriele, Michele, Vito, Tarik, Leonardo, Roman, Leslie, Mikael, Simon, Nithin, Rachael and Sara. Thank you for the great time we spent together.

I would like to thank my university colleague Matteo. We have been a great team during the last two years, especially in this last experience.

Many thanks to Claudia for hosting me but mostly for immediately becoming a good friend. Without you this great experience would have not been the same.

I would like to thank my parents for giving me the possibility to achieve my studies, for their continuous support in all my decisions.

Finally, thanks to all my friends for the important role they have in my life and simply for being my friends.

Chapter 1

Introduction

Humans have always been fascinated by the origins of intelligence. Several disciplines have tried to tackle the issue of what is the intelligence and how it can arise, from psychology to philosophy through artificial intelligence. During the history of this latter discipline, the question has gradually changed becoming: "Can machines behave intelligently?". In this context, besides answer to such question, one of the main goals of artificial intelligence is to build intelligent machines (e.g. intelligent robots). For this purpose, two main categories of synthesis methodology can be defined: the first category involves a series of approaches based on a design of intelligent entities by hand. The main problem of this approach is that, with the increase of the system complexity, the synthesis would be too difficult to conceive by a human designer. Moreover, the basic characteristics of intelligent systems are the ability of adaptation and learning, which are features hard to obtain with an approach by hand. Thus, recent studies have shown that, in many cases, it is preferable the employment of automatic design methodologies. Automatic processes are able to synthesize more flexible and robust entities and, in some cases, it has been shown that they can lead to innovative design solutions.

Typically, the automatic design process can be treated as a search problem based on two main components: the first is the model through which the robot behavior is represented and the second is the optimization algorithm that, working directly on the model, is designed to find the optimal configuration of its components.

Concerning the former aspect, several models exist but the most used so far are the artificial neural networks, usually trained by means of techniques inspired to the Darwinian evolution (e.g. genetic algorithms). In general, using genetic regulatory networks to model the

dynamic behavior of intelligent entities brings relevant advantages: these models, created to simulate both the structure and the functional behaviors of cells, provide intrinsically the same basic features of adaptivity and flexibility peculiar of biological cells, which are able to react to environment stimuli maintaining their internal organization. In this context, the model employed for this work are the Boolean networks (BNs), introduced by Stuart Kauffman. The main advantages of BNs with respect to neural networks is their compactness and simplicity and the fact that it is possible to precisely define and study their dynamical behavior.

The heart of the automatic design methodology is the optimization algorithm employed to find the optimal setting of the model which allow the intelligent entity, i.e. the robot, to perform a given target task. A suitable choice for the optimization algorithms can be the metaheuristic techniques, which are particularly appropriated for exploring huge search spaces in a limited amount of time.

Recent works in this context have proposed a first proof of concepts of a methodology based on BNs and a basic metaheuristic technique. In particular, the choice of the simplest stochastic technique employed has been intended to reject the null hypothesis by preventing the results from being affected by mechanisms algorithm dependent. In this work, developed in collaboration with the Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA) of the Université Libre de Bruxelles, we use the methodology proposed as starting point, studying some prominent properties of the design process and proposing relevant improvements.

The main goals of the work are two: first, want to analyze some features of the automatic design process. More precisely, the work aims at studying the connection between the search process which trains the robot's BN-based program to perform a specific task, and the trend of some properties of the network dynamic behavior. The hypothesis is that there exists a correlation between the learning trend and the value of certain features of the BN program, in particular in its dynamics. The second goal is the improvement of the methodology by means of advanced stochastic local search methods, drawing conclusion about the characteristic of the search process that affect the results.

In order to achieve the proposed goals, the methodology is tested on three robotic applications. The first two applications involve simple target tasks so as to allow the analysis of a large amount of data during the whole training process and investigate the prominent aspects.

During the last experiment, the most complex since it requires some form of memory to be performed, the main target is the methodology improvement through the employment of advanced stochastic local search algorithms.

The thesis is organized as follows:

In Chapter 2, we describe the model used to represent the robot behavior: Boolean networks. We focus on the most relevant aspect of the structure and the dynamics. Then we also present the state of the art concerning the automatic design procedure of BNs.

In Chapter 3, we introduce the background concepts of the work. In particular, after a brief historical overview about robotics and its connections with the automatic design of biological inspired networks (e.g. BNs) to control the robots, we present the related work and the basic methodology employed.

Chapter 4 provides a wide overview of the metaheuristic techniques. Moreover, we introduce some basic concepts used afterwards to perform the result analysis.

In Chapter 5 we discuss the robotic test cases. First we describe the task to be performed and the experimental settings for the training phase. Finally we analyze the results obtained focusing on the relevant properties emerging and trying to derive some considerations with a view to more complex tasks.

Chapter 6 concerns the final robotic application of the work. The target task is a form of sequence learning, which means that a form of memory is required. After the task description and the experimental settings, we describe the advanced stochastic techniques proposed in order to improve the methodology. Finally we compare the results and we draw important considerations about their analysis.

Finally, Chapter 7 draws some conclusions and gives an outlook for future works.

Chapter 2

Boolean Networks

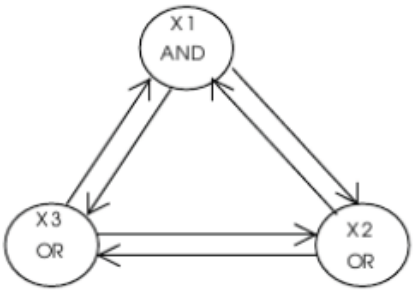
In this Chapter we will introduce Boolean networks. In Section 2.1 we discuss some basic and general concepts and in Sections 2.3 and 2.4 we focus on more relevant aspects for this thesis: Random Boolean Networks and some important studies about the design of Boolean Networks through advanced local search methods.

2.1 Introduction

Boolean Networks (BNs) have been introduced by Stuart Kauffman in 1969 [1] as a model for genetic regulatory networks (GRN). Subsequently, many works have been proposed in several research fields, and the community of complex systems is giving considerable attention to the BNs because of their properties which can model some aspects and mechanism of living system.

A BN is a discrete-state and discrete-time dynamical system. Its structure can be thought as an oriented graph with N nodes, each associated to a Boolean value x_i and a Boolean function $f_i = (x_1, ..., x_K)$ where K is the number of inputs of node i. The arguments of the function f_i are the boolean values of the nodes whose outgoing arcs are connected to i. On this simple structure, it is possible to define the network dynamic behavior as a sequence of state updates (i.e., each node updates its own boolean value according to the value of inputs and the associated boolean function). In a given instant of time t, the state of the system is represented by the array of the N Boolean variable values $s(t) = (x_1(t) ,..., x_N(t))$. Many kinds of dynamics and update schemes exist [2] but the most studied, and the one we consider in this thesis, is *synchronous* state updates of all the nodes and *deterministic*, i.e., the output of a boolean function is

unambiguously computed depending on its arguments. An example of states update in a simple boolean network is showed in Figure 2.1.



t			t+1		
X1	X2	X3	X1	X2	X3
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: *The table describes the dynamics of the BN showing the successor of each state.*

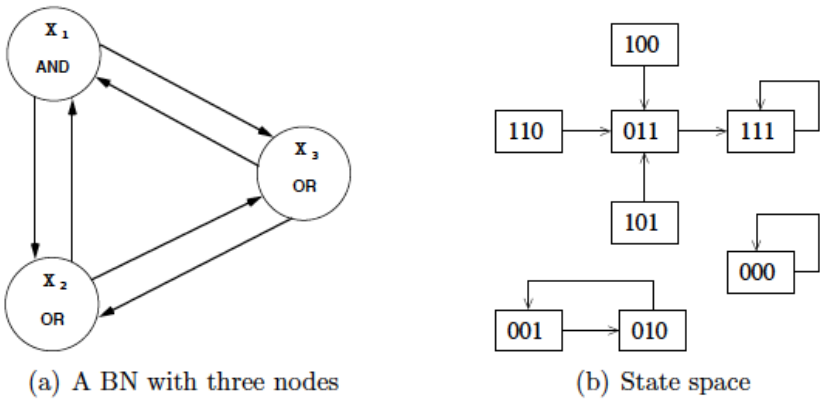
2.2 Dynamics

BN model's dynamics can be studied by means of usual dynamical systems methods, hence the usage of concepts such as *state (or phase) space, trajectories, attractors and basins of attraction* [3]. Chosen an initial state arbitrarily or randomly, depending on the specific application, the dynamics flows according to the update functions and scheme. Since the Boolean functions are deterministic, at each instant of time, the system passes from a state to its unique successor state. During its execution, a network traverses a succession of states, often called a *trajectory* in the state space. Furthermore, since the state space is finite (a node's Boolean variable can assume the value zero or one therefore the state space size is 2^N), the system will eventually assume a state previously encountered. Thus, a trajectory in the state space assumes this form:

- In the first phase, starting from the initial state, each state is different from all the previous ones and it will not be repeated. This states succession is called *transient*. It can also happen that this phase has length zero, i.e., the network does not undergo this phase.

- Eventually, the trajectory will reach a point in which a state, or a sequence of states, will be repeated. This means that the network has reached an *attractor*. If the attractor consists of one state, it is called a *point attractor* or *fixed point*, whereas if it consists of two or more states, it is called a *cycle attractor*. The set of states that lead to an attractor is called the *attractor basin*.

An example of a BN and its corresponding state space under synchronous and deterministic update is showed in Figure 2.2. Figure 2.3 shows instead the dynamics inside an attractor basin of a random boolean network.



(a) A BN with three nodes (b) State space

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).*

Many dynamical features of a network can be studied such as the length of attractors or the sizes of the basins of attraction drained by the state cycle attractors. Attractors have typically been the focus of many studies because of their strong biological implications. Attractors are the harbors in which the system settles down, hence they are the phase in which a system spends the most of the time. The other states in the state space are only transient points leading to such attractors. All dynamical systems, from neural networks to cardiac systems through genomic regulatory networks, exhibit attractors. For these reasons, the characteristics of attractors in system with many elements are of basic relevance in both development and evolution.

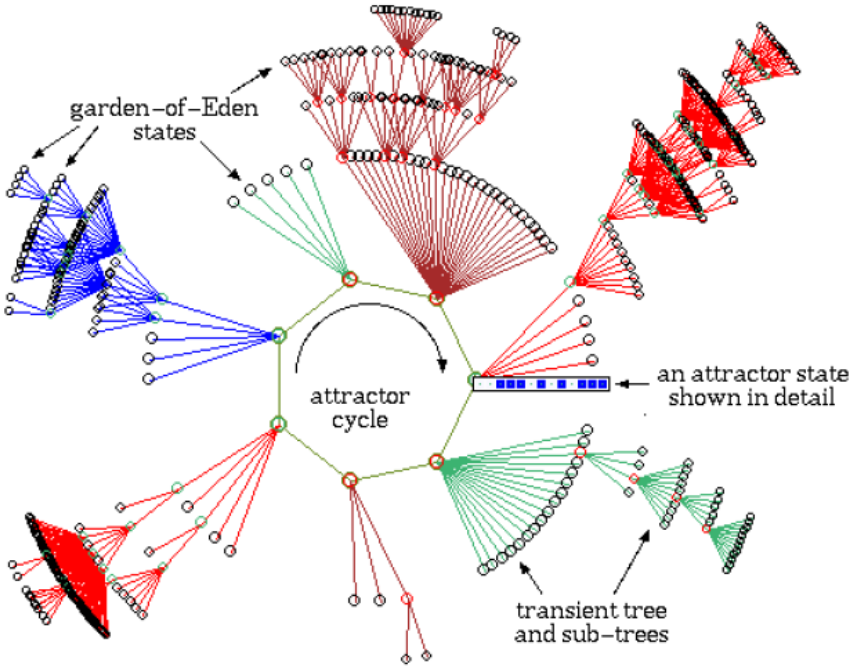


Figure 2.3: *Basin of attraction of a random boolean network with N=13 and K=3. The basin links 604 states, of which 523 are garden-of-Eden states. The attractor cycle has a period of 7*

2.3 Random Boolean Networks

Random Boolean Networks (RBNs) are a special category of Boolean Networks which was also introduced by Kauffman [1] to explore aspects of biological genetic regulatory networks. Since then, thanks to their features that capture relevant aspects and phenomena of complex systems (in particular in genetic and cellular mechanisms), they have been used as a tool in a wide range of areas such as self-organization (e.g., [5]), computation (e.g., [4]) and robotics (e.g., [16]).

Random BNs are typically created by choosing randomly K inputs per node and defining the Boolean functions assigning to each entry of the truth tables a 1 with probability p and a 0 with probability $1 - p$. Parameter p is called *homogeneity* or *bias*. Depending on the value of this parameter and K we can distinguish between three different regimes: *chaotic*, *ordered* and *critical*. Many studies have been carried out about these systems regimes and the properties that a system shows during each of these regime [5]. A system is in ordered regime

when a large part of its elements is fixed. In this kind of regime, the frozen component represents the bulk of the system and it leaves behind small and isolated island of components which are free to change state and have complex dynamics. On the other hand, in systems in chaotic regime, unfrozen elements form a connected component which spreads throughout the system and leaves isolated areas of frozen elements. The critical regime, the border between the ordered and the chaotic one, is also named edge of chaos" and it consists of an ordered sea that breaks into unfrozen islands, and the frozen islands join and percolate through the system. Many works have been proposed on the critical regime because of some its important proprieties, such as the capability of achieving the best balance between flexibility and robustness [6] and maximizing the average mutual information among nodes [7]. This features have suggested several conjectures concerning the fact that living cells, and living systems in general, are critical [8].

One of the most interesting and studied property of these dynamical 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, seeing how changes affect the stable behavior. Resuming BNs, for *perturbation* we mean a permanent mutation in the connections or in the Boolean functions of the BN. A *damage* is instead the cascade of changes in dynamical behavior caused by transiently altering the activity of a single binary variable [5].

In the ordered regime attractors are small and structural perturbations stay in small unfrozen areas, not causing cascade of damages spreading. Hence the perturbed network flows through the state space as the original one. Networks in this regime are also very robust with respect to the initial conditions whereas similar states tend to converge to the same state. In the chaotic regime, on the other hand, attractors are very large and a perturbation of the activity of any single Boolean variable, causes a cascade of damage percolates through the unfrozen sea and the system. This kind of network is very sensitive to changes in initial conditions because different states tend to diverge. Finally, at the edge of chaos, perturbations can spread but damages usually remain restricted to a portion of the system. In critical networks, moreover, nearby states tend to lie on trajectories that neither converge nor diverge in state space [9].

Living systems, and computational systems, need stability to survive, but also flexibility to adapt to either the environment and the changes it could have, and explore their space of possibilities. This

has led some researchers to argue that life and computation occur naturally at the edge of chaos [10], or at the ordered regime close to the edge of chaos.

> Living systems exist in the solid regime near the edge of chaos, and natural selection achieves and sustains such a poised state. *Stuart Alan Kauffman 1993*

There exist many techniques to study the features of BNs, but one of the most used is the *Derrida Plot* [11] which provides a measure of divergence/convergence of network dynamics in terms of *Hamming distance* (i.e., the number of positions at which the corresponding symbols are different) between states. In particular, in a Derrida Plot pairs of initial states are sampled at defined initial distances, *H(0)*, from the entire state space, and their mean Hamming distance, *H(t)*, after a fixed time, *t*, is plotted against the initial distance *H(0)*. For perturbation calculation, we consider two copies of the same RBN and we flip n node values in one copy. Then, we compare the states of the normal network with the state of mutated network by means of the Hamming distance *H(t)*, after a fixed time (i.e. after a state update of both the two networks). Repeating this procedure for different values of n between 0 and N and plotting the results, we can highlights the effect of perturbation and distinguish between ordered, chaotic and critical regime measuring the slope of the plot.

Thanks to these studies, scientists identified another important feature about dynamical regimes: changing the value of K, the dynamic regime of RBNs changes too. In particular, networks with $K \leq 2$ are in the ordered regime, and networks with $K \geq 3$, are in the chaotic regime. Derrida and Pomeau were the first to determine analytically that the critical phase was found when $K = 2$ [12]. Figure 2.4 shows dynamics of RBNs in different phases. To explain the motivation of this dependency between the number of inputs of each node of the network and the dynamical regimes we need to introduce the concept of *canalyzing* functions. A canalyzing Boolean function is any Boolean function having the property that the output is determined by only one input (for instance one input with value 1 is enough to let the OR function assume the value 1). The consequence of having many canalyzing functions inside a network is that a node value can force the elements it regulates to assume the same value at the next moment. Such a mechanism propagated iteratively to all the descendants along with the fact that the network has loops, creates *forcing loops* or *descendant forcing structures* [5]. The creation of large forc-

ing structures produces order in the network since these frozen areas form a large interconnected web that percolates through the whole system. Several works have been produced to prove the relationship between K and the number of canalyzing functions in the networks, and consequently the dynamical regimes [5].

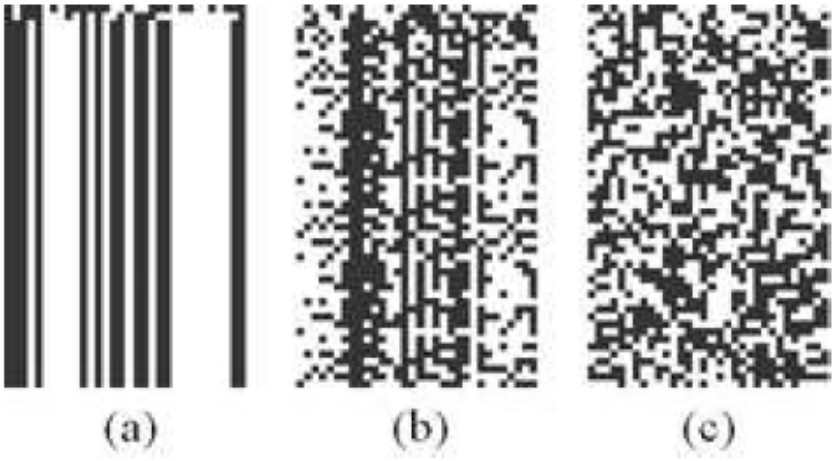


Figure 2.4: *Trajectories through state space of RBNs within different regimes, N = 32. A square represents the state of a node. Initial states at top, time flows downwards. a) ordered, b) critical, c) chaotic*

Derrida and Pomeau also introduced two important generalizations of the classical model: the first concerns nonhomogeneous networks (K is not necessarily the same for all nodes), and another is that they considered the concept of *homogeneity* or *bias p* (as mentioned above each entry of the truth table have a probability p of being one and *1 - p* of being zero), to numerically find the critical line corresponding to the critical regime and to define the relation between the parameters of the networks. Thanks to these intuitions and to the method they used called Derrida annealed approximation, they defined the following equation which describes the critical line:

$$2p(1 - p) = 1/K \tag{2.1}$$

The plot of this equation can be seen in Figure 2.5.

2.4 Boolean Network Design

The design of complex systems is one of the main challenges in scientific and engineering disciplines. Scientists have to face this basic

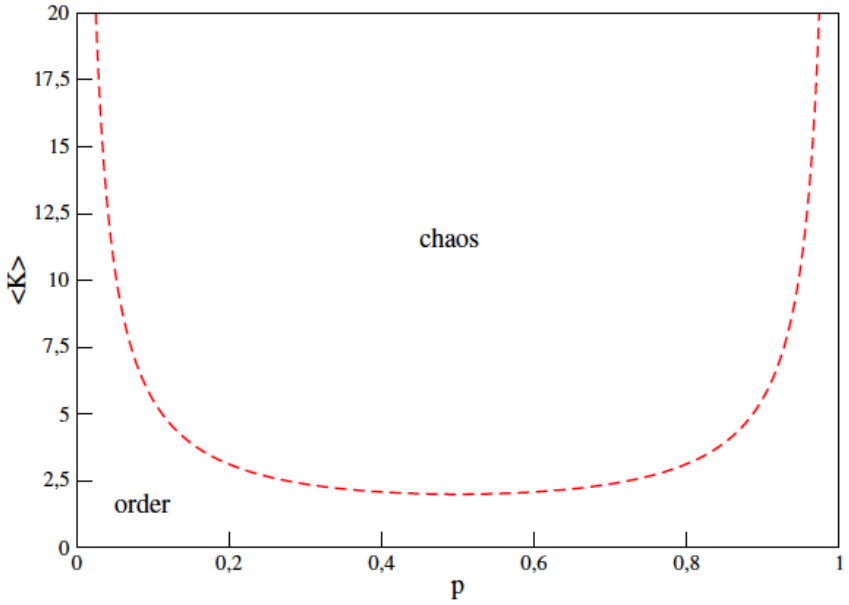


Figure 2.5: *Relationship between p and K in phase transitions*

issue in many research areas, such as reverse engineering of biological and social networks, design of self-organizing artificial systems and robotics. One of the most widely used approaches to tackle the problem is the synthesis and tuning of such systems by means of automatic techniques, most of which are search methods. Despite BNs, especially in the past, have been mainly used to model and study genetic or biological phenomena, more recently, several works have been produced to explore their potential as computational learning systems. One of the main reasons to investigate the learning process of Boolean Networks is that it is possible, in this scenario, to precisely define their dynamical state.

The first contribute concerning the potential of BNs as learning systems was by Kauffman [13], who in his work proposed a modified genetic algorithm (with only a mutation operator) with the objective of generating networks whose attractors matched a prescribed target state. Afterwards, Lemke et al. extended the scenario to cycle attractors targets using a full implementation of genetic algorithm (with crossover). Those studies, and other more recent including the discussion of experimental results on the application of a simple genetic algorithm to to obtain an attractor with a given length [14], are an investigation of the impact of evolution over BNs. In particular, they were meant to evolve some specific features in the networks, mostly

structural or dynamical properties such as the length of attractors or the control of the BN's trajectory to match a target [15]. On the other hand, few works have been proposed aimed at revealing some behavioral aspect of the network so that they can be used to address some specific task. Examples of this second approach are [15], in which the author uses the Boolean networks trained by either a Genetic Algorithm and a Iterated Local Search (ILS) to solve the Density classification problem, or [17] and [16] where the authors exploit a first investigation of the potential of automatic designed BN controllers for robots (we will discuss this topic in detail in the next Chapter).

The results obtained in all the mentioned studies raise interesting questions on the training algorithm, the search landscape structure and the evolutionary dynamics depending on the networks characteristics. We will focus on all these concepts in Chapter 4.

Chapter 3

Boolean Network Robotics

This Chapter introduces some basic concepts regarding the use of Boolean Networks in Robotics. In the Section 3.1 we introduce a series of background aspects required to fully understand the rise of this recent approach. In particular we give a brief historical introduction to robotics, and subsequently we illustrate the relevance of the genetic regulatory network models inside this discipline. Moreover, in Section 3.2 we present some related works and finally, in Section 3.3, we outline the automatic BN-robotic design methodology which we will use as starting point for our studies.

3.1 Background Concepts

In this section we present a historical introduction to Artificial Intelligence and Robotics, focusing on the aspect close to our work. In Section 3.1.2, we describe the basic features of the cellular models and the reasons for which they can be usefully employed in robotics, presenting some prominent examples of these studies.

3.1.1 Artificial Intelligence & Robotics

Humans have always been fascinated by the question about "what is intelligence". This issue has has been tackled by several disciplines, and the debate also includes the ideas of many philosophers and psychologists. Since the beginning of the XX century, thanks to technological advances, the question has gradually changed becoming: "Can machines think?", that is also "Can machines behave intelligently?". The discipline that, in the recent history, has tried to give an answer to these questions is Artificial Intelligence (AI). AI has its origin from

the work of Turing [18] which represented a milestone for the theory of computation and computability and was the basis for subsequent progresses in the field of machine computing. Some years later, in 1956, Newell and Simon [19] presented a program that could demonstrate theorems in logic, and on the basis of this and of another work by Newell and Simon [20], in AI the physical symbol hypothesis became prominent: every intelligent behavior could be simulated by appropriate manipulation of physical symbols. In this direction, artificially intelligent systems were built so that they could solve whatever problem for which knowledge could be modeled in form of logic symbols. Even in robotics, which in the meantime had grown up also thanks to the cybernetic contribution (discipline that studied the animals as if they were machines and tried to model their behaviors using control theory and statistical information theory), reasoning was performed by symbolic manipulation of symbols. In 1980s, a new paradigm in contrast with the symbolic paradigm, connectionism, moved its first steps studying artificial neural networks (bottom-up approach). The real revolution, however, started with Brooks [21] who thought that the study of Artificial Intelligence should start from building machines that interact with the real world, abandoning the top-down traditional approach for which modeling was always required. The idea of Brooks was to turn to a biologically-oriented, bottom-up methodology. He also introduced two basic concepts for modern robotics: situatedness (i.e. robots perceive the world through their sensors, and the world provides them all the information required to execute their behavior) and embodiment (i.e. robots can act on the environment, moving in the world and modifying it, actively determining what will be the feedback they will subsequently receive). These concepts are today the basis of the design of embodied agent programs. The work of Brooks gave rise to a series of new studies about embodied cognitive science, where the importance of embodiment stems from the possibility to exploit the dynamic interactions of the agent with the environment, so that intelligent behaviors can emerge. The same ideas are also the basis of other studies, which use biological or, in general, natural inspired models and processes to synthesize robot agent programs.

3.1.2 Genetic Regulatory Networks

One of the branches of research generated by the studies presented in Section 3.1.1, concerns the exploitation of embodied Genetic Regulatory Networks (GRNs) as real-time control systems for artificial

organisms, in particular robots. GRNs are models used to simulate both the structure and the functional behaviors of cells. This kind of approach in robotics arose from a series of historical considerations about the concepts of embodiment and adaptivity. Maturana and Varela in 1980 [22] argued that adaptivity is not limited to a mere reaction to environmental stimuli. It takes place due to a "structural coupling" between the agent and the environment, during which they are both sources of mutual perturbations. From this point of view, the cognition of an agent is considered as "functional embedding" of an agent in its interaction with the world. This kind of embodiment considers adaptation as the capability of an agent to maintain its internal organization in relation to the perturbations that come from the environment. In Maturana and Varela's view [22], this features of maintaining under control the internal organization is peculiar to living systems and cannot be found in artificial artifacts. The basic examples of such adaptivity property in living systems is the biological cell: it is composed by several parts enclosed within the cell membrane. All these parts are connected to form a network of interactions meant to maintain the cell internal organization.

Embodied genetic regulatory network-driven control systems exploit this natural metaphor to develop flexible and robust controllers, able to dynamically adapt to the environment and continually driving the interplay between agent and environment, giving rise to coherent observable emergent behaviors. The GRN model most commonly used so far are neural networks, models that attempt to simulate either the structure or functional aspects of the biological central nervous system. Nevertheless, several GRN models have been proposed like the Biosys model used in [23], in which a cell consists of Genes and Proteins that regulate the interaction of the cell with the environment. Another important model are the Boolean networks, the model we use in the thesis which is recently having considerable consideration from the scientist thanks to the structural simplicity but in the meantime the dynamical richness.

In GRN-driven controllers, behaviors are not directly specified but rather GRNs encode complex dynamics through which structured behaviors can emerge. In such a way, the agent (the robot in our context) performs autonomously the specific task. In particular, the GRN is continually coupled to the agent's environment, perturbing and being perturbed, acting as a real-time control system: at each control step, first, the sensors values are encoded in the input subset of the network. Then a network update occurs. Finally the effector on the environment

are performed through the actuators according to the GRN's output. The way in which the agent behaves in the environment determines the stimuli it will receive as input in the future. Moreover, it is not unusual that the interaction between the system and the environment is not perfectly known in advance.

Thus, for all the above reasons, it appears preferable to use an automatic technique, such as an optimization algorithm or an evolutionary algorithm, that gradually builds up the GRN control system of an autonomous agent. In order to do this, the automatic procedure exploits the behavior of the system embedded in its environment and the variations in the interactions between the environment and the agent itself. In addition, automatic design procedures can make the process simpler, more robust and general with respect to a customized procedure. The space of solutions explored by automatic technique can be larger and less constrained than that explored by conventional engineering methods. It has been shown that this feature can lead the search process to innovative design solutions [24, 25].

One of the most important example of the automatic design methodology is Evolutionary Robotics (EV), that is a methodological procedure to automate the design of agent programs inspired by the Darwinian theory of natural selection. The first intuition that the Darwinian evolution could be used to generate efficient control systems was by Alan Turing in 1950s, when he suggested that intelligent machines capable of adaptation and learning would be too difficult to conceive by a human designer. They could rather be obtained by means of an evolutionary process with mutations and selective reproduction of the fittest individuals in a population [26]. Historically the agent program model most commonly used in the EV approach are Artificial Neural Networks [27, 28] but, in this thesis, we want to further investigate the potential of Boolean Networks as real-time robotics controller combined with the exploitation of advanced search strategies.

In the following we present some related works about this branch of research, and the description of the methodology used to automate the design process.

3.2 Related Works

The employment of the Boolean Networks in robotics is a recent proposal. Few studies have been presented and the one we focus on is the

thesis [17] and the subsequent paper [16]. These works can be considered the first attempt to automatically synthesize Boolean network robotic controllers by means of a methodological procedure. Through the new methodology proposed in this studies, which we will describe in Section 3.3, the authors were able to prove the possibility to train and tune, with a simple search algorithm, a BN controller. After the training process, the robot could manage either some simple tasks, like a path follower, or more complex tasks like phototaxis and anti-phototaxis. However, the goal of those works was to provide a proof of concept, without focusing on properties of the methodology, such as its success rate on robotics case studies and the possible improvements, or on the features that the training process arises in the network structure and dynamics. This thesis aims to investigate in a deeper way the first of the two aspect just mentioned, that is the learning process improvement in terms of methodology enhancement and employment of more sophisticated search strategies. The work developed in parallel by M. Amaducci [29] has the goal of study the second issue, i.e., the consequences of the training process on the network dynamics and other interesting properties, like the memory representation, that can emerge from the whole process.

3.3 Methodology

In this section we describe the main aspects of the automatic design methodology for BN robot controllers proposed in the work discussed above and used in the thesis as starting point for subsequent improvements and inspirations.

3.3.1 BN-Robot coupling

The first thing required to the process concerns the definition of a coupling between the BN controller and the robot. In other words, we define a mapping between the robot's sensors and the network's input and between the robot's actuator and the network's output. Thus, the Boolean value of the network's input nodes is not determined by the network dynamics but is set according to the robot's sensor readings. Similarly, the output node values are set by the network dynamic and are used to encode the actuators activation. The mechanism is a first distinctive aspect of this approach with respect to most of the studies proposed regarding BNs, in which they have been mainly considered

as an isolated systems, as they have been not assumed to have inputs. Figure 3.1 shows the scheme of the coupling between BN and robot.

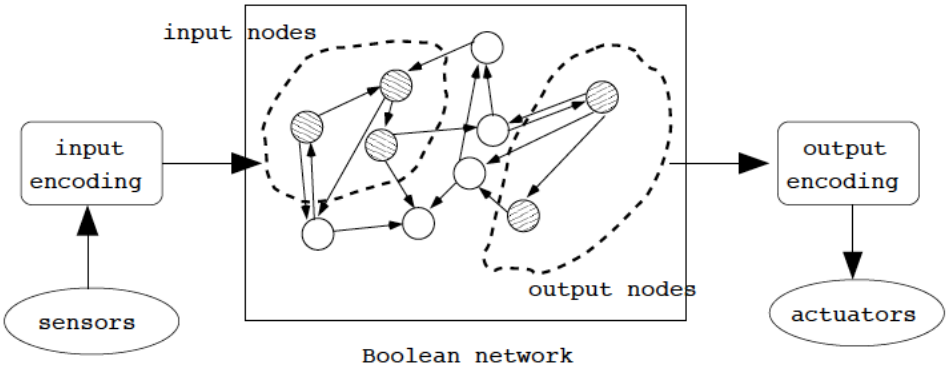


Figure 3.1: *The coupling between BN and robot.*

Several ways of defining the mapping are possible, but the most natural is via a direct encoding. In the Boolean Network model the nodes can assume only binary values. The sensor's values and the actuator's activation thresholds have therefore to be discretized and represented in a binary form in order to be encoded on the corresponding set of network nodes. This last aspect is maybe the main drawback of BNs with respect to other models because a binary encoding is not always feasible or simple in general, while in other continuos values models, like the Neural Networks, the same issue does not exist.

Once the input and the output mapping are defined, the Boolean Network, that is the robot controller, has to be designed in order to obtain a program which can perform a given task. Many approaches can be adopted to achieve this goal, such as designing a BN focusing on its dynamics properties so that they can be mapped into given features of the target robot behavior. Our methodology, instead, models the BN design process as a search problem, in which the objective function to be maximized is the robots performance.

3.3.2 Design methodology

Treating the design process as a search problem, the automatic design methodology is composed by two main components: the robot program or controller and the optimization algorithm. In the matter of the robot program, we already discussed its nature: the process

starts from a RBN, whose parameters are the number of input of each node K and the bias p. Both the topology and the Boolean functions are randomly generated, but while the topology does not change, the truth tables are the subject of the optimization algorithm's moves.

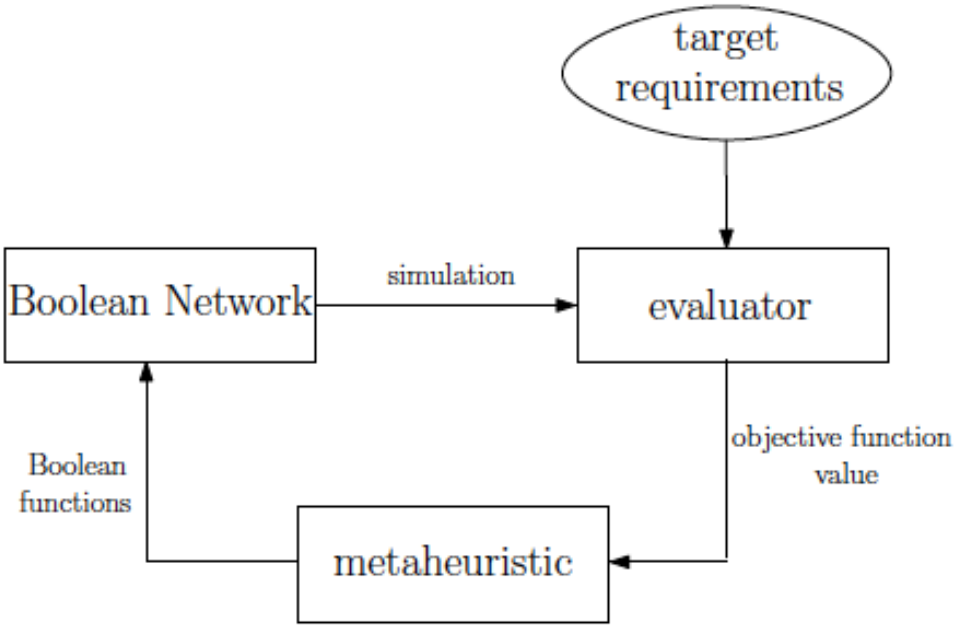


Figure 3.2: *Methodology approach*

The process, which can be modeled as a constrained combinatorial optimization problem, consists of a series of iterations whose cycle can be depicted like in Figure 3.2. At each iteration the optimization algorithm (in our case a metaheuristic technique) operates on decision variables which encode Boolean functions of the BN. The obtained network is then injected" into the robot (according to a suitable input-output mapping defined as described in Section 3.3.1) and executed. The resulting robot behavior is evaluated according to specific target requirements, dependent on the task to be performed. Hence, the objective function is represented by the performance measure of the robot. Typically, the execution on the real robots during the training process is too expensive in terms of time and resources such as the battery charge. For these reasons, both the execution and the evaluation of the robot behavior are performed in a simulated environment by a specific software component. In particular, for all the robotics applications in this thesis, we use an open source modular multi-robot

simulator called *ARGoS* [31] (*Autonomous Robots Go Swarming*). Finally, the performance evaluation is passed back to the metaheuristic technique which can thus proceed with the search process, appropriately modifying the BN's Boolean functions. This process ends when a certain number of iterations is reached or when a certain target performance is obtained. In Chapter 5 and 6, during the presentation of the robotics experiments, we will discuss in detail all the aspects of the process.

The employment of metaheuristics ensures numerous advantages, like the efficient exploration of huge search spaces or the possibility to incrementally improve the process, starting from a simple strategy to a more sophisticated one. However, all the features of metaheuristics will be widely discussed in the next Chapter. Obviously the methodology is not limited to the Robotics field, but it is sufficiently abstract to be employed for several other problems and disciplines.

Chapter 4

Metaheuristics

In Chapter 3 we introduced the automatic design methodology that is the basis of our synthesis process of Boolean Network agent programs. The procedure is strongly based on the metaheuristic techniques which are responsible of the search space exploration. The goal of this Chapter is to provide a wide panoramic of the metaheuristics, focusing on the features that make them particularly appropriated for tackling the issue of automatic design of BNs.

In Section 4.1 we give a brief introduction of the topic with some background concept, such as combinatorial problem and computational complexity. Then, in Section 4.2 we introduce the stochastic local search methods, starting from the basic local search techniques and then describing more advanced local-search based and population based algorithms. Finally, in Section 4.3 and 4.4 we focus on some relevant properties of the result analysis and of the search space structure.

Throughout the Chapter we follow the approach of Hoos and Stutzle in [32].

4.1 Introduction

Combinatorial problems are mathematical problems whose solutions typically consist of grouping, ordering or assignments of a discrete, finite set of objects that satisfy certain conditions or constraints. Combinations of these objects form the potential solutions of the combinatorial problem. Prominent examples of such problems, such as finding the shortest path on a graph or determining whether there is an assignment of truth values of a logic formula under which the formula is satisfied, can be found in many areas of computer science and

other disciplines. It is possible to distinguish between two concepts: a *problem* is the abstract definition of a category of problems such as the examples cited above. A *problem instance*, instead, consist of a specification of all the parameters and constraints of the problem. An abstract problem can be seen as the set of all its instances. For each instance, combinations of variable values form the set of potential solution, namely *candidate solutions*. Candidate solutions are solutions that may be possibly encountered during an attempt to solve the given problem, but unlike solutions, they may not satisfy all the constraints of the problem definition. Combinatorial problems can be subdivided in two categories:

- **Decision Problems:** the solution of each instance of a decision problem is specified by a set of logical conditions. Two variants are possible:

  - the *search variant*, whereby, given a problem instance, the objective is to find a solution (or to determine that no solution exists);
  - the *decision variant*, in which for a given problem instance, one wants to answer the question whether or not a solution exists.

  However, the two variants are related because the algorithms used to solve the search variants are also able to solve the decision variant. For many combinatorial decision problems, the converse also holds: algorithms for the decision variant of a problem can be used for finding actual solutions.

- **Optimization problems:** optimization problems can be seen as a generalization of decision problems, with the addition of a solution quality evaluation through an *objective function*. Depending on whether this function has to be minimized or maximized, the problems can be stated as minimization or maximization problems. We can distinguish between two variants of optimization problems as well:

  - the *search variant*, whereby, given a problem instance, the objective is to find a solution with optimal (minimal or maximal) objective function value.

– the *evaluation variant*, in which for a given problem instance, the objective is to find the optimal objective function value (i.e., the solution quality of an optimal solution).

The search variant is the most general of these, since with the knowledge of an optimal solution, the evaluation variant can be solved trivially.

Often, optimization problems are defined based on an objective function and a set of logical conditions. In this cases candidate solutions satisfying the conditions are called *feasible* or *valid*, and among those, the optimal solutions can be distinguished on the basis of the objective function evaluation.

With respect to these definitions, the problem of automatic design of a Boolean network-based program for robots can be modeled as a combinatorial optimization problem, by properly defining the set of decision variables, constraints and the objective function.

The most natural way to solve a combinatorial decision and optimization problem is to explore and search for solutions in the space of its candidate solutions. Thus, these problems are sometimes also called *search problems*. Unfortunately, given a combinatorial problem instance, the size of the set of candidate solutions grows exponentially with the size of that instance. This issue raises questions about the existence of efficient methods to explore such vast spaces and, specially, the time required for solving an instance of a combinatorial problem.

Questions like these are the core of the *computational complexity theory*, that studies the classification of computational problems, in terms of computation time and memory space required to be solved. Such theory plays a fundamental role in the metaheuristic algorithms, because the primary field of application of such techniques is a class of computationally very hard combinatorial problems, for which no efficient algorithms are known (where efficient means polynomial runtime w.r.t. instance size). Complexity theory usually deals with problem classes rather than the single instances. For a given algorithm, the complexity is characterized by the functional dependency between the size of an instance and the time and space required to solve this instance. Since generally the time complexity is the most restrictive factor, problems are often categorized into complexity classes with respect to their asymptotic worst-case time complexity. Thus, the complexity of a problem is usually defined by the *worst-case*, i.e., the time complexity in the worst case over all problem instances of a given

size. If a suitable definition of the computational complexity of an algorithm for a specific problem is given, the complexity of the problem itself can be defined as the complexity of the best algorithm for this problem.

Two particularly interesting complexity classes are $\mathcal{P}$ and $\mathcal{NP}$. $\mathcal{P}$ is the class of problems that can be solved by a deterministic machine in polynomial time. In this definition, deterministic machine means a machine model whose decisions can be determined unambiguously, basing on its current internal state. On the other hand, $\mathcal{NP}$ is the class of problems that can be solved by a nondeterministic machine in polynomial time. Given the current state, nondeterministic machines make decisions choosing among a set of all the possible alternatives. They are not equivalent to machines that make random choices but rather they are idealized models of computation that have the ability to make perfect guesses for certain decisions.

Every problem in $\mathcal{P}$ is also contained in $\mathcal{NP}$, because it is always possible to emulate deterministic calculations on a nondeterministic machine. As of today, the reverse, that is whether $\mathcal{NP} \subseteq \mathcal{P}$, and consequently $\mathcal{P} = \mathcal{NP}$, is not true because a lot of relevant problems are in $\mathcal{NP}$, but they are not contained in $\mathcal{P}$. This means that no polynomial-time deterministic algorithm is known to solve such computationally hard problems because the best algorithms known so far have exponential time complexity. This question, however, is one of the most open problems in computer science. Many of the hard problems in $\mathcal{NP}$ are strictly related and can be translated into each other with polynomial deterministic time methods (*polynomial reductions*). If given a problem, every problem in $\mathcal{NP}$ can be polynomially reduced to it, then the problem is $\mathcal{NP}$-hard and we say that it is at least as hard as any other problem in $\mathcal{NP}$. This definition means that $\mathcal{NP}$-hard do not necessarily have to belong to the class $\mathcal{NP}$ themselves, as their complexity may be higher than $\mathcal{NP}$ problems. $\mathcal{NP}$-hard problems that are contained in $\mathcal{NP}$ are called $\mathcal{NP}$-complete: $\mathcal{NP}$-complete problems are believed to have at least exponential time-complexity for any realistic machine or programming model; in a certain sense, these problems are the hardest problems in $\mathcal{NP}$.

One fundamental result of computational complexity theory states that it suffices to find a polynomial time deterministic algorithm for one single $\mathcal{NP}$-complete problem to prove that $\mathcal{NP} = \mathcal{P}$. This stems from the fact that all $\mathcal{NP}$-complete problems can be encoded into each other in polynomial time.

Despite many practical relevant combinatorial problem are $\mathcal{NP}$-hard or $\mathcal{NP}$-complete, this does not mean that that it is impossible for a problem to be solved efficiently. We can distinguish between two main possible approaches to solve these kind of problems:

- **Complete techniques:** they are guaranteed to return an optimal solution in finite time, or return failure if the problem is infeasible, for every finite size instance of a problem. The drawback is that for growing problem size, the problem instances become quickly intractable in terms of computation time needed for practical purposes.
- **Approximate Techniques:** they do not return proof of optimality but they find (near-)optimal solutions efficiently, significantly reducing the complete algorithms amount of time.

In Section 4.2 we discuss the general approach of approximate algorithms, which can be characterized as search algorithms. We will focus on advantages and disadvantages of these techniques, highlighting the reasons of the introduction of more sophisticated algorithms, the metaheuristics.

4.2 Stochastic Local Search

All the common approaches for solving hard combinatorial problems can be characterized as search algorithms. The basic idea behind these algorithms is to iteratively generate and evaluate new candidate solutions. Evaluating a candidate solution means to decide whether it is an actual solution in the case of combinatorial decision problems, while it means determining the respective value of the objective function in the case of optimization problems. Despite the time complexity of $\mathcal{NP}$-hard problems is exponential, search algorithms can be much more efficient, i.e. characterized by polynomial complexity. Moreover, the evaluation of candidate solutions is often rather straightforward to implement.

4.2.1 Local Search

Search methods can be classified on the basis of different aspects, mainly the way in which the candidate solutions are generated and the search space visiting paradigm. The most common distinction,

based on the second aspect, is between systematic and local search: *systematic search algorithms* traverse the search space completely, ensuring that eventually either a solution is found (if it exists), or the fact that no solution exists for the problem instance is determined with certainty. This property of systematic search algorithm is called *completeness*. On the other hand, *local search algorithms* start at some location of the search space, representing a feasible solution, and try to improve it by iteratively move from the present location to a neighboring location. Each location in the search space has a relatively small number of neighbors and the moves are performed on the basis of decision based on local information only. This kind of algorithms are typically *incomplete*, which means that it is not guarantee that existing solutions are eventually found, and if no solution exists, this fact will never be determined with certainty.

Local search algorithms are often based on perturbative search: candidate solutions are composed of solution components, such as the values in the Boolean functions of a Boolean network during a learning process. Therefore, given a candidate solution it is easy to move to a new candidate solution, that is a neighbor for the first one, modifying one or more of the corresponding solution components. These small local modifications can be characterized as *perturbations* of the current candidate solution. Hence, the methods based on this space navigation mechanism can be classified as *perturbative search methods*.

Other local search algorithms, characterized as *constructive search methods*, use a different approach: these algorithms try to generate "good" (where for optimization problems, the goodness corresponds to the value of the objective function) complete candidate solutions by iteratively extending partial candidate solutions. In many cases constructive local search methods can be combined with perturbative local search exploiting the advantages of an hybrid approach. Moreover, both perturbative and constructive methods combined with other mechanisms such as the backtracking, can be the basis of systematic search methods.

According to the foregoing discussion, it might be argued that, due to their incompleteness, the local search techniques are generally inferior with respect to systematic algorithms. But this is not the case, because there are many problem instances which are known to be solvable, and hence in these situations the goal is to find a solution rather than understand whether one exists. In addition, in scenarios in which the time to find a solution is limited, that is almost every real world problem, systematic algorithms may need to be stopped

before the search termination. This premature termination can cause problems, specially to the constructive techniques that search through spaces of partial solutions without computing complete solutions early in the search. Thus, local search algorithms are often advantageous in certain situations, particularly if reasonably good solutions are required within a short time, if parallel processing is used and if the knowledge about the problem domain is rather limited.

Many of most widely used and successful local search algorithms exploit the use of randomized choices in generating or selecting candidate solutions for a given problem instance. These algorithms are called *Stochastic Local Search (SLS)* methods. As discussed above, the search process starts from a selected initial candidate solution and then proceeds by iteratively move from a position to a neighboring candidate solution, making decisions on the basis of limited local knowledge only. In SLS methods, these decisions as well as the search initialization can be randomized. In addition, the search process may use additional memory, for instance, for storing a limited number of recently visited candidate solutions.

In order to formally describe a stochastic local search process for combinatorial problems, we need to define some basic components which provide the basis for solving a given problem using the SLS paradigm:

- The *search space* $\mathcal{S}(\pi)$ of instance π, which is a finite set of candidate solutions $s \in \mathcal{S}$ (also called search positions, locations, configurations or states); over this space it is also defined a set of (feasible) solutions $\mathcal{S}'(\pi) \subseteq \mathcal{S}(\pi)$;

- A *neighborhood structure* is a function $\mathcal{N} : \mathcal{S} \rightarrow 2^S$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s. Typically, the choice of an appropriate neighborhood relation is fundamental for the performance of an SLS algorithm and often, this choice needs to be problem specific.

- An *objective function* $f(\pi,s)$: $\mathcal{S}(\pi) \rightarrow \mathbb{R}$ that maps each search position onto a real number in such a way that the global optima of π correspond to the solutions of π. The objective function is used in the search process to assess the candidate solutions in the neighborhood of the current position. Thus the objective function guides the search process, and the efficacy of the search depends on its properties. For these reasons this function is usually problem-specific and often depends on the search space, solution

set and neighborhood underlying the SLS. Notice that often a distinction is drawn between the concept of evaluation function (denoted with g and used to assessing or ranking candidate solutions in the neighborhood of the current search position) and the objective function f. The objective function characterizing the problem is often used as an evaluation function, such that the values of the evaluation function correspond directly to the quantity to be optimized. Thus, in the following we do not use this distinction, simply using the concept of objective function f (for completeness, in the figures that outline the algorithms such distinction is maintained).

- A finite set of *memory states* $\mathcal{M}(\pi)$, which, in the case of SLS algorithms that do not use memory, may consist of a single state only.
- An *initialization function init*(π) which specifies a probability distribution over initial search positions and memory states.
- A *step function step*(π) that map each search position and memory state onto a probability distribution over its neighboring search positions and memory states.
- A *termination condition* which indicates the probability with which the search is to be terminated upon reaching a specific point in the search space and memory state.

The metaheuristic search can be thought as a search process over a graph characterized by the triple $L = (\mathcal{S}, \mathcal{N}, f)$, that is the solution set, the neighborhood function and the objective function. The search starts from an initial node and explores the graph moving from a node to one of its neighbors, until it reaches the termination condition.

One of the most basic examples of SLS techniques is the *iterative improvement*. Iterative improvement starts from a randomly selected point in the search space (all the position in the search space have the same probability to be chosen), and then tries to iteratively improve the current candidate solution with respect to the objective function f. A move is only performed if the candidate solution it produces is better than the current solution. A stochastic version of this algorithm also exists, called *Stochastic Descent* (SD), in which the neighbor to be evaluated is randomly picked into the neighborhood sets.

Note that, following the definition of iterative improvement, in the case in which none of the neighbors of a candidate solution s

realizes an improvement, the search process terminates even if the quality of the current solution is not satisfying. A candidate solution with this property corresponds to a local minimum of the objective function *f*. A *locally minimal solution* (or *local minimum*) can be formally defined with respect to a neighborhood structure $\mathcal{N}$ as a solution $\hat{s}$ such that $\forall\ s \in \mathcal{N}(\hat{s}) : f(\hat{s}) \leq f(s)$. We call $\hat{s}$ a strict locally minimal solution if $f(\hat{s}) < f(s)\ \forall\ s \in \mathcal{N}(\hat{s})$. In cases where an SLS algorithm guided by an objective function encounters a local minimum that does not correspond to a solution, this algorithm can "get stuck" so that it can not find good solutions. For these reasons the basic techniques have been extended and enhanced with strategies that are required to prevent the search from getting trapped in local minima and to escape from them. In the following we present some of the most known SLS algorithm, describing the escape strategies and the advantages of each one, deepening more in those related to the thesis. SLS methods can be usually classified into *trajectory methods*, and *population-based methods*. The former kind describes a trajectory over a search graph while the latter methods perform a search process characterized by an iterative sampling of the search space. We apply the same classification in our following discussion.

4.2.2 Trajectory-Based Methods

Introducing the iterative improvement we highlighted its main limitation, i.e. the fact that it can get stuck in local minima of the underlying objective function. The first simple idea that can alleviate this problem is using a larger neighborhood. As discussed before, the performance of a search method significantly depends on the neighborhood relation definition, in particular, on the size of the neighborhood. Generally, larger neighborhoods contain more and potentially better candidate solutions, and hence they typically offer better chances for finding locally improving search steps. Clearly, the time complexity for determining improving search steps is much higher in larger neighborhoods. In this context, there is a tradeoff between the benefits of using large neighborhoods and the associated time complexity of performing search steps. One possible way to partially solve this issue is to use a *neighborhood pruning* method. This kind of approach consist in using large neighborhoods but reducing their size by never examining neighbors that are unlikely to yield any improvements.

First Improvement & Best Improvement

Another simple method for making the local search technique more efficient is to select the next search step more efficiently. From this perspective, the *Iterative Best Improvement* method is based on the random selection, at each step, of one of the candidate solutions that achieve a maximal improvement in the objective function. One thing to notice is that this algorithm, also known as *greedy hill-climbing* or *discrete gradient descent*, needs a complete evaluation of the whole neighborhood in each step.

The *Iterative First Improvement* algorithm tries to avoid the time complexity of evaluate all neighbors at each search step, selecting the first improving candidate solution encountered during the neighborhood evaluation process. Obviously, the order in which the neighbors are visited during the evaluation, can deeply affect the performance of this strategy. Instead of using a fixed order, random orderings can also be used.

Summarizing, the search steps in first improvement algorithms can often be computed more efficiently than in best improvement, since typically only a small part of the local neighborhood is evaluated. However, the improvement obtained by each step of first improvement is usually smaller than for best improvement and therefore, more search steps have to be performed in order to reach a local optimum.

Variable Neighborhood Search

Resuming the idea of larger neighborhood to increase the search process performance, another common approach can be exploited. It consists of using standard neighborhoods until a local minimum is encountered, at which point the local search switches to a different, typically larger, neighborhood, which may allow the process to escape from the local minimum and reach further progress. This idea is the basis of the *Variable Neighborhood Search* (VNS) framework, which comprises a number of algorithmic approaches including *Variable Neighborhood Descent* (VND). VND is an iterative improvement algorithm that realizes the general idea behind VNS in a very straightforward way. In VND, k neighborhood relations are defined which are often ordered according to increasing size: $|\mathcal{N}_1| < |\mathcal{N}_2| < \ldots < |\mathcal{N}_{kmax}|$. During the search process, the algorithm switches to neighborhood structure $\mathcal{N}_i$ to $\mathcal{N}_{i+1}$ whenever the search stagnates, that is whenever no further improving step is found for a neighborhood $\mathcal{N}_i$. If an improvement is obtained in $\mathcal{N}_i$, the search process switches back

to $\mathcal{N}_1$, from where the process is continued as previously described. It has been shown that variable neighborhood descent can considerably improve the performance of iterative improvement algorithms. Such improvements are both w.r.t. to the solution quality of the local optima reached, as well as w.r.t. the time required for finding (high-quality) solutions compared to using standard Iterative Improvement in large neighborhoods [33]. Figure 4.1 shows a VND algorithm outline.

```
procedure VND(π', N_1, N_2, ..., N_k)
    input: problem instance π' ∈ Π', neighbourhood relations N_1, N_2, ..., N_k
    output: solution ŝ ∈ S'(π') or ∅
    s := init(π');
    ŝ := s;
    i := 1;
    repeat
        find best candidate solution s' in neighbourhood N_i(s);
        if g(s') < g(s) then
            s := s';
            if f(s) < f(ŝ) then
                ŝ := s;
            end
            i := 1;
        else
            i = i + 1;
        end
    until i > k
    if ŝ ∈ S' then
        return ŝ
    else
        return ∅
    end
end VND
```

Figure 4.1: *Algorithm outline for Variable Neighborhood Descent for optimization problems*

A different approach to the idea of larger neighborhood concerns the selection of search steps from large neighborhoods efficiently, composing more complex steps from a number of steps in small and simple neighborhoods. The idea is exploited in *Variable Depth Search* (VDS) and in *Dynasearch*. The former can be seen as an iterative improvement method in which the local search steps are variable length

sequences of simpler search steps in a small neighborhood. The latter, differently from VDS, requires that the individual search steps that compose a complex step are mutually independent. Independence means that the individual search steps do not interfere with each other with respect to their effect on the objective function values and the feasibility of candidate solutions.

Randomized Iterative Improvement

Another idea to let the search out from local minima is to accept that in some cases the search process can perform worsening steps which can help to escape. One of the simplest ways of exploiting this idea is to extend iterative improvement algorithms such that they select, sometimes, a neighbor at random rather than an improving move. The alternation with a fixed frequency of such random walk steps and improvement steps can lead to situations in which the effect of a random selection are immediately undone by the subsequent improvement selection. In order to avoid the problem, it is possible to introduce a parameter $wp \in [0,1]$, called *walk probability* or *noise parameter*, that corresponds, at each step, to the probability of performing a random move rather than a improvement move. The resulting algorithm is called *Randomized Iterative Improvement* (RII). It differs from a iterative improvement just for two aspects: the step function, in which it determines probabilistically the step to be executed, and for the termination condition, because there is no more need to terminate as soon as a local optimum is encountered.

An advantageous consequence of the fact that arbitrarily long sequences of random steps can occur is that there is always a chance to escape from any local optima. However, RII is rarely applied in practical application, probably due to the fact that more complex SLS algorithms can often achieve better performances.

Simulated Annealing

Simulated Annealing (SA) is a SLS that explores a similar idea: the probability of accepting a worsening step should depend on the value of the objective function such that the worse a step is, the less likely it would be performed. This idea is the basis of a family of algorithms called *Probabilistic Iterative Improvement* (PII). In these techniques each step can be split into two stages: in the first, a neighbor of the current candidate solution is randomly chosen. In the second stage, an acceptance criterion is applied, according to a probability

distribution over neighboring candidate solutions based on their respective objective function value. The acceptance criterion, known as the Metropolitan condition, is shown in Figure 4.2.

$$p_{accept}(T, s, s') := \begin{cases} 1 & \text{if } f(s') \leq f(s) \\ \exp\left(\frac{f(s)-f(s')}{T}\right) & \text{otherwise} \end{cases}$$

Figure 4.2: *Acceptance criterion in simulated annealing*

The probability of performing worsening steps depends on the parameter T, also called *temperature*. The higher the temperature is, the more likely the algorithm is to accept even drastically worsening steps with relatively high probability. Maintaining the temperature constant during the search, we obtain a PII algorithm while the peculiarity of the SA is exactly the generalization of this idea: T can vary over the search process through a mechanism called *annealing schedule* or *cooling schedule* inspired to the physical annealing process. An annealing schedule is a function which determines, at each instant of time t, the respective value of temperature $T(t)$. There exist several ways to vary the temperature during the search, such as geometric, logarithmic or non-monotonic. The choice, obviously, strongly depends on the problem to be solved but the simple geometric cooling schedule has been shown to be quite efficient in many cases.

Simulated annealing is one of the most used algorithms. The reason can be the fact that it is simple to enhance it with other techniques such as a greedy initialization or neighborhood pruning. Another appealing reason can be the fact that, under certain conditions, the convergence of the algorithm can be proven (i.e., any arbitrary long trajectory in the search space is guaranteed to terminate in an optimal solution). The conditions are, however, extremely severe and typically not feasible in practical applications.

Tabu Search

A radically different approach to tackle the problem of escaping from local minima exploits information about the search history. *Tabu Search* (TS) is a general SLS method that systematically utilizes memory to guide the search process. In particular, we focus on the simplest and the most widely used technique, also known as *simple tabu search*. It uses a short-term memory in order to both avoid cycles

in the search trajectory and to escape from local minima. Typically, simple tabu search consists of a best improvement strategy to select the best neighbor of the current candidate solution. In a local minimum, this can lead to a worsening step or to a *plateau* step (i.e., a step which does not lead to any change in the objective function value). In order to prevent the search process to immediately return to a previously visited neighbor, TS keeps track of recent visited solutions and forbids them. Often, memorizing solutions is not convenient in terms of performance, therefore *moves*, i.e. solutions components, are stored instead. However, storing moves instead of solutions could forbid improving, not yet visited candidate solutions. For this reason, many TS algorithms use an *aspiration criterion*, which specifies the conditions under which a tabu restriction is overridden, thereby including the otherwise forbidden solution. A commonly used aspiration criterion is to allow solutions which have improving evaluation with respect to the currently-known best solution.

The most important parameter of TS is the the duration (in search steps) for which the tabu restrictions is applied, called *tabu tenure*. If tabu tenure is too small, search stagnation may occur. If it is too large, the search trajectory is restricted and good solutions may be missed. More complex version of the algorithm exist, that try to find a tabu tenure optimal value in different ways or extend the memory to form of intermediate-term or long-term memory.

Tabu search algorithms have been successfully applied to several combinatorial problems. Often, beyond the tabu tenure value, a careful choice of the neighborhood definition is crucial for the performance of these algorithms.

Dynamic Local Search

The previous techniques exploit two main strategies to guide the search process out from local optima: the first allows worsening steps while the second concerns changing the neighborhood structure during the search. We can account into the latter the variable neighborhood search algorithms, which increase the neighborhood size, and the tabu search methods, which instead change the neighborhood structure forbidding certain movements.

A further idea is to modify the search space with the aim of making unexplored areas more desirable. In order to do that, the SLS algorithms collectively called *Dynamic Local Search* (DLS) methods, change the objective function dynamically whenever a local optimum

is encountered. In such a way, the search landscape gradually changes and further improvement steps become possible. This can be achieved by associating *penalty weights* with solution features, typically solution components. Whenever the search process gets stuck in a local minima, the penalty of some solution components are increased and the objective function is modified so as to take into account these penalties (Figure 4.3). This mechanism leads to a degradation of the current solution evaluation until it reaches a value worse than the evaluation of a neighboring candidate solution. At that point, improving moves become available.

$$g'(\pi', s) := g(\pi, s) + \sum_{i \in SC(\pi', s)} penalty(i),$$

Figure 4.3: *Dynamic local search objective function with penalties*

Iterated Local Search

So far, the techniques we discussed in this Chapter could be characterized as *simple* SLS methods, in the sense that they perform only one type of search step. The performance of the search process can be often significantly improved combining various different types of search steps. The result is a series of *hybrid* SLS algorithms that can be seen as a combination of simpler SLS techniques. One of the simplest and most used hybrid algorithm explores a very intuitive idea to address the problem of escaping from local optima: it utilizes two types of SLS steps, the first for reaching as efficiently as possible a local optima and the other for effectively escaping from the local optima. This idea is the basis of *Iterated Local Search* (ILS) algorithm. Figure 4.4 shows an ILS algorithm outline.

Starting from an initial candidate solution, usually randomly selected in the search space, a local optima solution is obtained through an underlying local search procedure (*localSearch*). Then, at each iteration of the cycle the algorithm is divided in three stages: first, a perturbation is applied to the current candidate solution in order to obtain a new candidate solution (*perturb*). Then, starting from this new solution, the local search procedure leads the process to a new local optima solution. Finally, an acceptance criterion (*accept*) is applied in order to decide whether the new local optimum is accepted

```
procedure ILS(π′)
    input: problem instance π′ ∈ Π′
    output: solution ŝ ∈ S(π′) or ∅

    s := init(π′);
    s := localSearch(π′, s);
    ŝ := s;
    while not terminate(π′, s) do
        s′ := perturb(π′, s′);
        s″ := localSearch(π′, s′);
        if (f(s″) < f(ŝ)) then
            ŝ := s″;
        end
        s := accept(π′, s, s″);
    end
    if ŝ ∈ S′ then
        return ŝ
    else
        return ∅
    end
end ILS
```

Figure 4.4: *Iterated local search outline*

as new candidate solution from which start the new cycle of the algorithm. A variety of termination conditions can be used for deciding when the search process ends.

The three stages are therefore the core of the precess. A good balancing of these components is fundamental for achieving a good tradeoff between *intensification* and *diversification*. The former aspect has the goal of greedily improve solution quality by means of some guidance mechanism, for instance an objective function. The latter aims to prevent search stagnation trying to lead the search process to a good coverage of the space. Obviously, the *localSearch* procedure has strong impact on the performance. In general, more effective local search methods lead to better performing ILS algorithms [32]. The perturbation has the role of modifying the current candidate solution in such a way that it will not be immediately undone by the subsequent

local search. Finally, the acceptance criterion can also have a strong impact on the performance: for instance, a strong intensification of the search is obtained if the best of the two solutions s and s'' is always accepted.

ILS can be seen as a simple, yet powerful technique for extending basic SLS algorithms. Moreover ILS algorithms are typically easy to implement but, at the same time, they are today among the best approximate search methods for many combinatorial problems. Some example of ILS algorithms successfully applied to hard combinatorial optimization problems are [34, 35].

4.2.3 Population-Based Methods

The SLS methods discussed so far are all characterized by the fact that they consider a single candidate solution at each search step. An intuitive extension is to consider algorithms in which several individual candidate solutions are simultaneously maintained. This idea is the basis of the category of algorithms called population-based methods. The first clear advantage of using a population is the implicit search diversification feature which leads to an enhanced exploration capability of the search process.

A first population-based SLS is the *Ant Colony Optimisation* (ACO), a method inspired by aspects of the pheromone-based trail-following behavior of real ants. It was introduced by Marco Dorigo in [36] as a metaphor for solving hard computational problems. The peculiarity of ACO is that the individuals in a population (*ants*) interact with each other in a indirect way. This indirected, distributed, dynamic exchanging of information is performed via the so-called (artificial) *pheromone trails*. For a complete description of the ACO metaheuristic we refer to [37], since in the following we focus on a category of algorithm more related to this work, evolutionary algorithms.

Evolutionary Algorithms

The most prominent example of population-based methods is based on a direct interaction between the individuals within a population of solutions. These methods are called *Evolutionary Algorithms* (EAs). EAs are inspired by principles of Darwinian evolution theory. They are typically iterative methods that, starting with a set of candidate solutions (the initial population), repeatedly apply three genetic operators, *selection*, *mutation* and *recombination*. Through these opera-

tors, EAs apply the principles of evolution, leading to the development of species (solutions) that are better adapted for survival in a given environment. Specifically, using selection, mutation and recombination, the current population is (completely or partially) replaced by a new set of candidate solutions at each iteration. The populations generated in the course of the evolution process are called *generations*.

At each generation, the *selection* operator aims to chose, typically probabilistically, the candidate individuals either for the next generation or for the subsequent application of mutation and recombination. Usually, selection is performed ensuring that fitter individuals have a higher probability of being chosen. The fitness of a single individual is measured by means of a *fitness function* that is typically positively correlated with the objective function that quantifies the quality of a candidate solution. *Mutation* is a unary operator which aims to introduce small, and often random, modifications in a single individual. *Recombination* is an operator that generates one or more new individuals (the *offspring*), by combining components of two or more individuals of a *parents* population. The most used type of recombination operator is the *crossover*, inspired to biological mechanisms, which consists in combining slices of information belonging to different parent individuals in order to assemble new child individuals. Several kinds of crossover combination exist, that differ from each other for the number of parents involved in the recombination (*multi-parent crossover*), for the number of points in which the parent individuals are split (*single-point* vs *multi-point crossover*) or for the general schema of recombination (*uniform crossover*). One of the most commonly used recombination is the *one-point binary crossover* operator, whose schema is shown in Figure 4.5.

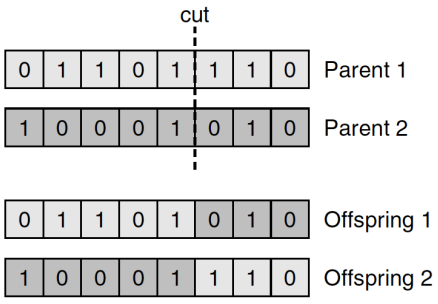


Figure 4.5: *One-point binary crossover*

The concepts introduced so far are the basis of all the EAs. The

most prominent example of Evolutionary Algorithms for combinatorial problem is the *Genetic Algorithm* (GA). GAs were introduced by John Holland in the 1970s to mimic features of natural evolution [38]. The general scheme of GAs can be outlined as in Figure 4.6.

```
P ← GenerateInitialPopulation()
Evaluate(P)
while termination conditions not met do
   P′ ← Recombine(P)
   P″ ← Mutate(P′)
   Evaluate(P″)
   P ← Select(P″,P)
end while
```

Figure 4.6: *Genetic algorithm outline*

The algorithm starts from an initial population, for instance generated at random, and iteratively evolves it by applying the operators previously described: mutation, recombination and selection. The function *Evaluate* computes the fitness of each individual of the population. One of the crucial aspects for the performance of a GA is the selection operator that has to bias the search toward the fittest solutions, i.e., those with the highest objective function value. Many selection schemes are possible, the most of which involve probabilistic choices (e.g. *roulette wheel selection* or *tournament selection*). However, it is often advantageous to use elitist strategies, which ensure that the best candidate solutions are always selected.

The first goal of genetic algorithms, and of the other evolutionary methods, is the search diversification through which the search process can cover promising regions of the search space. This feature can facilitate the process to reach high-quality solutions of a given optimization problem instance. Thanks to this property GAs have been successfully applied to many combinatorial problem [39, 38]. Obviously, several different version of GA can be obtained varying the three operator. In general the best setting for a good performance depends on the specific problem. A prominent extension to the simple GA is, however, the *steady-state* GA, in which the parent population is not completely replaced by the selection, but an arbitrary number of parents are maintained.

4.3 Analysis of Algorithms

One of the crucial aspects of the SLS algorithms is the analysis of both their behavior during the search process and their performance. Given the non-deterministic nature of such algorithms, a theoretical analysis of these features is often difficult to perform. This stems from the fact that theoretical results are typically hard to obtain, and even when they do exist, their practical applicability is often limited.

Given the fact that the knowledge about the behavior of stochastic techniques is often insufficient to perform a theoretical study, the analysis of the run-time behavior of SLS algorithms is often based on empirical methodologies. In particular, a series of hypothesis, computational experiments and observations is employed in order to obtain a model which can explain the phenomena and reveal practically relevant aspects of algorithmic behavior.

4.3.1 Run-Time Distributions

Applying local search methods to real world scenarios, the utility of a solutions typically depends on its quality as well as on the time required to obtain it. Therefore, SLS optimization algorithm evaluation should be based on a detailed knowledge and analysis of the solution probabilities $P_s(RT \leq t, SQ \leq q)$, where P_s is the probability to obtain a certain solution quality q within a given time t. This probability can be determined from the probability distributions of the random variables which characterize the solution quality and the run-time of a given algorithm. Formally we can state the following definition:

Given an optimization algorithm A for an optimization problem Π and a solvable problem instance $\pi \in \Pi$, let $P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q)$ denote the probability that A applied to π finds a solution of quality less than or equal to q in time less than or equal to t. The runtime distribution (RTD) *of A on π is the probability distribution of the bivariate random variable $(RT_{A,\pi}, SQ_{A,\pi})$, which is characterized by the run-time distribution function $rtd : \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow [0, 1]$ defined as $rtd(t, q) = P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q)$.*

The behavior of an optimization algorithm, and similarly for decision problem algorithms, applied to a given problem instance is completely described by the corresponding RTD. The knowledge of the runtime distribution allows one to easily compute other performance

measures about an algorithm, such as the mean time to find a solution, the median and its standard deviation. The converse, instead, does not hold because the RTD carries more information than these other measures. In fact, it enables the study of the behavior of algorithms in applications which involve more complex tradeoffs.

From the definition above, the runtime distribution for an optimization algorithm is a multivariate probability distribution. This kind of functions are often difficult to handle and manage. For this reason it is preferable to refer to the univariate distributions of the run-time required to obtain a given solution quality threshold. The resulting probability distributions are called *Qualified Run-Time Distributions* (QRTD). The qualified RTDs are marginal distributions of the bivariate RTD. In practice, they are usually used for studying the ability of an algorithm to find optimal or near-optimal solutions (if the optimal solution quality is known). Analyzing the QRTD for subsequent and tight values of solution quality threshold can give a detailed overview of the general behavior of the algorithm. An example of QRTD is shown in Figure 4.7.

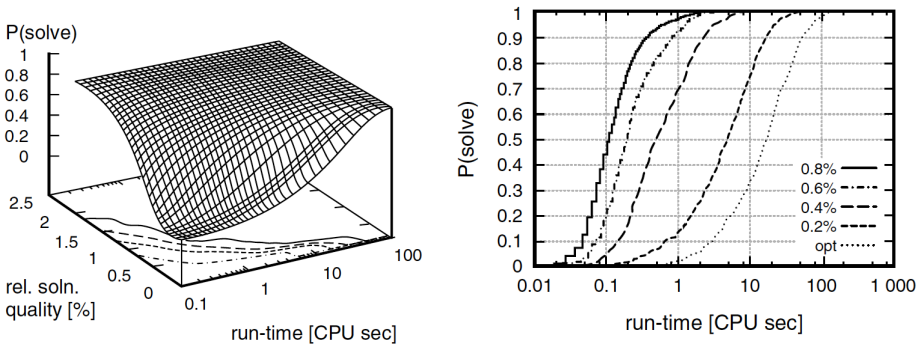


Figure 4.7: *Bivariate Run-time distributions for SLS algorithms applied to hard combinatorial optimization problem (left), corresponding univariate Qualified RTDs (right)*

Practically, it is often impossible to compute analytically the RTD of a SLS algorithm. Instead, the real RTDs characterizing the behavior of an algorithm, are typically approximated by empirical RTDs. For a given algorithm A on a given problem instance π, the empirical RTD can be obtained by performing k independent running of A on π and record its results (at each improvement the solution quality obtained and the time at which the improvement is achieved) during the search process. The RTD is calculated as the cumulative distribution function

associated with these observations. Formally, let *sq(t, j)* denote the quality of the best solution found in run j until time t, the cumulative empirical run-time distribution of A on π is defined by $Ps(RT \leq t', SQ \leq q') = \#\{j|sq(t', j) \leq q'\}/k$. Obviously, the more runs are executed, the better will be the empirical approximation of the true distribution.

The empirical study of SLS algorithms are often performed in order to compare different techniques, trying to determine a superiority of one of them with respect to the others. Thus, these studies are intended to show that, given two algorithms, one of them consistently gives a higher solution probability than the other. RTDs can be very useful for this purpose, since the previous qualitative sentence can be formally captured by the concept of probabilistically domination in the following way:

For an instance $\pi \in \Pi$ of an optimization problem Π and optimization SLSs A and B for Π, A probabilistically dominates B on π for solution quality less than or equal to q if, and only if, $\forall t : P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q) \geq P_s(RT_{B,\pi} \leq t, SQ_{B,\pi} \leq q)$ and $\exists t : P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q) > P_s(RT_{B,\pi} \leq t, SQ_{B,\pi} \leq q)$.
A probabilistically dominates B on π if, and only if, A probabilistically dominates B on π for arbitrary solution quality bounds q.

From the definition, one algorithm probabilistically dominates another if, and only if, the two qualified RTDs do not cross each other. This provides a graphical method to check the probabilistic domination. However, for a single problem instance, a probabilistic domination does not always hold. In these situations, in which a cross-over between the two RTD lines occurs, other statistical tests can be used to verify the performance differences (e.g., the Mann-Whitney U-test or the Wilcoxon test can be applied to determine whether the medians of two samples are equal, hence a rejection indicates significant performance differences) .

For a more detailed discussion about the properties of the RTDs in comparison and analysis of the SLS algorithms behavior we refer to Chapter 4 of [32].

4.3.2 Search Space Structure

The performance of SLS algorithms deeply depends on the properties and the structure of the search space. The study of these features

can allow to significantly improve or understand the behavior of such techniques.

We already introduced, in Section 4.2.1, the triple $L(\pi) = (\mathcal{S}(\pi), \mathcal{N}(\pi), f(\pi))$ representing the search process over a graph. This triple is called *search landscape*, or simply *landscape*, of the optimization problem instance π. The landscape makes it possible to define some global and local properties of the search space. Often, the detailed analysis of the landscape is based on local features, i.e. performed by classifying the search positions according to the topology of their local neighborhood.

One of the most relevant landscape features in terms of the impact on the SLS behavior are the local minimum. Clearly, in landscapes in which there are not local minima other than the global minima, even the simplest search algorithms should eventually reach the optimal solution. Therefore, the difficulty of solving a combinatorial problem can be attributed to the presence of large numbers of local minima in the corresponding landscapes. Another important feature which can have impact on the algorithm behavior is the distribution of the local minima within the search space. For instance, the local minima can be uniformly distributed across the entire landscape or clustered in some regions causing a large variability in localized local minima density.

The guidance mechanism of the search method through the landscape is given by the objective function. A crucial aspect in studying the performance of SLS algorithms is to verify the effectiveness of such guiding mechanism. A first aspect to check is whether there exists a relationship between the solution quality and the distance to an optimal solution. Ideally, the better the solution are evaluated, the closer they should be to the optimum. This aspect can be explored by the *Fitness-Distance Analysis* (FDA), which measures this relationship in terms of the correlation between the quality of solutions (on the basis of their objective function value) and their distance to the closest globally optimal solution.

However, one of the features that more deeply influences the behavior of SLS algorithms is the landscape *ruggedness*. In particular, a search landscape is called *smooth* when neighboring positions have similar objective function values, while in the *rugged* case, the values across neighboring states are characterized by high variance in the objective function evaluation. Intuitively, landscape ruggedness is related to the number of local optima: landscapes with a high density of local optima are typically rugged, while smooth landscapes have usually fewer local optima. From the relation between the number of local

optima in the landscape and the difficulty of solving the corresponding combinatorial problem, it is possible to state that more rugged landscapes are harder to search for SLS algorithms. The ruggedness property can be analyzed by estimating (in cases where it cannot be determined analytically) the *autocorrelation* of the landscape. Smooth landscapes are characterized by high autocorrelation, while rugged ones have low autocorrelation. One common approach is to measure the correlations between neighboring solutions by means of a uninformed random walk of m steps starting from a random initial position. The autocorrelation of a series $G = (g_1, \ldots, g_m)$ of objective function values is computed as

$$r = \frac{\sum_{k=1}^{m-1}(g_k - \bar{g}) \cdot (g_{k+1} - \bar{g})}{\sum_{k=1}^{m}(g_k - \bar{g})^2} \tag{4.1}$$

where $\bar{g}$ is the average value of the series. This definition refers to the autocorrelation of length one, i.e., that corresponding to the correlation between two points that are one step far in the random walk. One important aspect is that the starting point of the random walk has no influence on the information obtained from the trajectory. Hence, any random walk is representative of the entire search landscape. The autocorrelation analysis can be therefore useful to assess the differences between various neighborhood relations for a given problem or for studying the impact of parameter settings of an SLS algorithm on its behavior.

Finally, other important features of the search landscapes that can influence the SLS performances, are *plateaus*, *barriers* and *basins*. Plateau regions are simply regions of positions that are all at the same level. This kind of regions are dangerous for the search process which can stagnates within the plateau if the escape mechanisms are not effective. Not all the local minima or closed plateaus are equally hard to escape from. One factor connected with the difficulty of achieving an improvement in the given objective function f from a local minimum or closed plateau is the difference in f that needs to be overcome in order to reach a position at a lower level. This intuitive sentence is captured by the concept of *barrier*. Related to the notion of local minima depth and barrier height is the notion of a *basin*, which intuitively describes a region of positions at or below a given level. However, for a complete description of all these concepts, the analysis methods, and the subsequent measures that might be adopted in order to tune the algorithm performance, we refer to Chapter 5 of [32].

Chapter 5

Test Cases

In this Chapter we present the first robotic experiments of this work. The goal is both to acquire familiarity with robotic applications and to exploit the potential of the automatic controller design methodology introduced in Chapter 3. To these purposes we test the methodology on two basic tasks, analyzing the results obtained using a very simple search algorithm and trying to draw some important considerations with a view to more complex tasks. Section 5.1 contains an introduction to the goals of the Chapter, in particular the aspects that we want to analyze about the automatic design process. Section 5.2 introduces the general experimental settings. In Section 5.3 and 5.4 we present the two experiments, describing the task to be performed and all the specific settings chosen to achieve the target. Finally, in Section 5.5 we analyze the results obtained in both the experiments trying to draw some consideration about the relevant aspects regarding the methodology and the parameters that can affect it.

5.1 Introduction

Besides familiarizing with the robotics applications, the goal of the test cases and of the Chapter is to investigate some substantial aspects of the automatic design process, that we shall also call *training* w.r.t robotics applications. First, we want to provide evidences that with simple tasks to perform and search landscapes high autocorrelated (Section 4.3.2), the methodology can lead to very good results even using a simple SLS method such as the stochastic descent. Moreover, we want to highlight the differences in the methodology effectiveness when using different type of random Boolean networks.

Another issue we analyze is the link between the improvements

during the training and other properties of the networks: in particular, our hypothesis is that there is a correlation between the local search improvements and the trend of two quantities, that are the number of states visited by the networks inside their state space during their execution and the complexity of these networks. Analyzing the former aspect we want to show how the improvement moves in the search landscape correspond to particular effects on the dynamical features of the networks, especially on the number of states visited. The latter quantity also concerns aspects of the system dynamics providing a measure of their complexity. In this regard, we think that significant improvements of the objective function values correspond to an increase of the system complexity. All these questions are in some respect counterintuitive: the search algorithm and the network dynamics operate in two different spaces, the search space and the state space, that during the design process are not directly bound. The methodology works, in fact, only on the structure of the networks, modifying their Boolean functions, without directly deal with dynamical issues. Despite this, the fact that an indirect connection might emerge between the two space properties is very attractive and could be an interesting way to exploit for future improvements to the methodology.

Then, we check the robustness of the design process essentially by two methods: the first is the testing, through which we test the best networks obtained during the training over a series of different initial condition in order to verify if they can adapt their behavior to general situations. The second method are the RTDs, described in Section 4.3.1, that show the trend of the training phase over the time and depending on the given solution quality expected.

Finally, we show the role that the number of nodes of each network plays in the design phase. A wrong choice of this parameter can deeply affect the performance of the process: it is crucial to find the right tradeoff between the required computational capacity to perform the target task and a small search landscape which can help the search algorithm.

5.2 General settings

In this Section we present all the training settings shared by the two experiments. The methodology we adopt, that is the automatic design process described in Section 3.3.2 and illustrated in Figure 3.2, starts

from an initial random Boolean network. In order to achieve both the proposed tasks, we use as first attempt Boolean networks with 20 nodes, i.e., $N = 20$. The topology, i.e. the inputs, of the networks are randomly generated with $K = 3$, which means that each node has 3 ingoing arcs. Boolean functions are also randomly generated with a p, i.e. the probability to have an entry with 1 as output value in the Boolean function of a node, which varies. In particular, we generate 100 initial networks with $p = 0.5$ (with respect to the equation 2.1 these networks are in the chaotic regime), 100 networks with $p = 0.9$ (ordered regime) and 100 networks with a $p = 0.788675$ (the value which verifies the critical line equation). In the second part of each test case we test the same process also with initial Boolean networks with a number of nodes of 40 ($N = 40$) to verify how the results can change depending on this parameter.

During each cycle of the process, the Boolean network representing the current candidate solution is injected into the robot and connected through a suitable mapping to the robot's sensors and actuators. In this way the execution of the network, in particular its dynamics, controls the robot behavior, constantly perturbing the environment through the robot actuator and being perturbed by the environment in which the robot moves.

The robot employed for all the experiments presented in this work is the *e-puck*. The e-puck is a small circular robot designed with the purpose to be both a research and an educational tool in universities. It has many interesting features, but here we focus only on those necessary for our tasks. The robot is equipped with 8 infrared proximity sensors which have also the capability to perceive the amount of light acting as 8 light sensors. The disposition of these sensors is shown in Figure 5.1. Moreover, the e-puck can detect the color of the ground it is moving on thanks to 3 ground sensors located on the bottom of the chassis. Finally, it has 8 LED single color on the top and 2 motors that actuate the 2 wheels.

The specific mapping between the robot and the BN controller, depend on the task to be performed and, for this reason, it will be discussed in the Sections concerning each experiment.

Once the mapping is defined and the network is configured, a suitable software component simulates the experiment and evalutes the behavior of the robot, according to the specific target requirements. This step is typically executed in a simulated environment rather than on real robots. The reason is that the several iterations that are usually involved in the automatic methodology, could render the whole

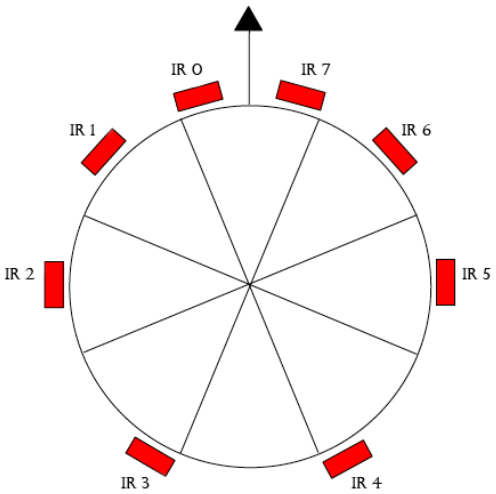


Figure 5.1: *Proximity sensors grouping*

process unfeasible in a reasonable amount of time, or w.r.t. other limited resources such as the battery charge of the robots. The simulated environment used for all the experiment performed during this work is an open source modular multi-robot simulator called ARGoS [31]. The simulation is time discrete, that is, for each instant of time (or *tick*), the robot perceives the environment through its sensors, selects the action to perform, and finally actuates it.

The performance evaluation is then passed back to the metaheuristic technique which can consequently modify the current solution and proceed with the search process. The optimization process works only on the Boolean functions of the nodes, leaving the initial topology unchanged, in order to not increase the problem complexity. The whole cycle is executed iteratively to gradually improve the behavior and finally obtain a solution which performs the target task. The local search method employed for the test cases is a basic SLS strategy, the Stochastic Descent (SD) whose outline is shown in Figure 5.2.

In our case, the search process starts from a Random BN generated as described above. The neighborhood relation, which defines the modification performed by the metaheuristic on the current solution in order to obtain a neighboring one, is defined by randomly choosing a node function and then flipping a random bit in its Boolean function. The pair formed by the node and the Boolean function position is uniformly sampled with replacement. The difference between the algorithm outline in Figure 5.2 and the version we adopt, is that

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 outline*

we accept a new candidate solution even if its evaluation is equal or slightly worse (by an *epsilon = 0.0001*) than the current one. This movements, called *sidewalks*, are intended to avoid the instant stagnation of the search process inside the plateau regions. By allowing these side movements, the search process can continue the exploration and, in some cases, escape from such regions.

For the first two experiments, in particular with the configuration of network with 20 nodes i.e. *N=20*, we run the optimization process for just 1000 iterations, given the triviality of the tasks to be performed. Notice that, with the configuration of network with 40 nodes, i.e. *N=40*, we increase the number of iterations to 2000 in order to obtain comparable final results.

5.3 Obstacle Avoidance

In this Section we describe the first robotic experiment of the thesis. In particular we focus on all the settings which completely define the training process for the obstacle avoidance task.

5.3.1 Task Definition

The target task of the experiment is a problem that has been often tackled in the past, mostly in the context of robotics: the *obstacle avoidance*. For our purposes, it simply consists of a robot that has to avoid any collision with the obstacles it detects along its path (no path planning issues are involved). Many branches of AI can be applied

to the problem, some of which employ automatic design approaches (e.g. in this field several works with neural networks and evolutionary algorithms have been proposed [27, 28, 43]) while others concern the application of specific algorithms (e.g. fuzzy logic [44], steering behavior model created by Craig Reynolds and generally applied to flocking [45]).

In order to completely define how the task learning process is performed according to our methodology, we need to describe its environment: it consists of a corridor of length *6.5* meters and width *0.5* meters with the exit in the origin of the coordinate system. The robot is placed within the corridor *6* meters far from the exit. During the experiment it has to advance along the corridor avoiding collisions and reach the exit. A representation of the environment is shown in Figure 5.3.

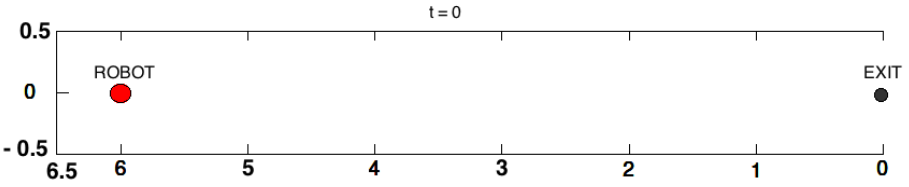


Figure 5.3: *Obstacle Avoidance corridor environment*

5.3.2 Robot setup

As mentioned in Section 3.3.1, the way in which the Boolean networks control the robot behavior is the following: we use a subset of the nodes as input nodes and some other as output nodes. The input nodes are connected to the robot sensors, while output nodes are connected to the actuators. Input nodes are nodes whose Boolean variable, at each instant of time, is not determined by BN dynamics (*frozen nodes*), but it is set according to the robot sensor values. On the other hand, output nodes are nodes whose Boolean variable, at each instant of time, is used to set actuator values. Thus, at each step, the values detected by the sensors are encoded in the input nodes, then a network update is performed, and finally the output node values are used in order to set the actuators. Hence, this mechanism requires a suitable mapping between value range of sensors and actuators, and Boolean values.

In this first experiment, with the aim of avoiding obstacle on their

path, the robot needs only proximity sensors and wheels in order to detect the obstacles and change the movement accordingly. The mapping is defined as follows: we use four frozen nodes to encode the eight proximity sensor states. Thus proximity sensors are gathered in pairs, and if at least one of them overtakes a chosen threshold (proximity sensor values are normalized between 0, which means no obstacles detected, and 1, that is obstacle very close, therefore we need a threshold beyond which we can consider the sensor on), the corresponding input node is set to 1. The groups of sensors are chosen to ensure that robots can detect obstacles from the four directions north-east, south-east, south-west and north-west.

Furthermore, we set two nodes as output nodes, each of which control a wheel of the robot. Thus the speed value can assume for each wheel only two values, that are ON or OFF. The complete mapping between perceptions/actions and node values is shown in Table 5.1 and Table 5.2.

Input node	**Node value**	**Proximity state**
x_0	0	No obstacle detected by IR0 or IR1
	1	Obstacle detected by IR0 or IR1
x_1	0	No obstacle detected by IR2 or IR3
	1	Obstacle detected by IR2 or IR3
x_2	0	No obstacle detected by IR4 or IR5
	1	Obstacle detected by IR4 or IR5
x_3	0	No obstacle detected by IR6 or IR7
	1	Obstacle detected by IR6 or IR7

Table 5.1: *Mapping between robot sensors and BN input nodes*

5.3.3 Evaluation

In this Section we focus on the last aspects that characterize the methodology proposed, that is number and nature of the different initial conditions (namely *trials*) whereby each network is tested, and the evaluation criteria used to assess the performance of each network:

- **Trials:** given the arena previously described and depicted in Figure 5.3, we test the behavior of each robot over 6 trials start-

Nodes		Actuators	
x_4	x_5	Left wheel	Right wheel
0	0	OFF	OFF
0	1	OFF	ON
1	0	ON	OFF
1	1	ON	ON

Table 5.2: *Mapping between robot actuators and BN output nodes*

ing from different initial conditions. The starting position remains the same, in particular the robot starts from the position *6,0,0* (i.e. 6 meters far from f the origin and the exit of the corridor). What makes the difference between the trials is the initial rotation, which assumes values in a range among *120°* and *240°* with steps of *24°*. Through these trials we evaluate the network over several heterogeneous situations in which it has to detect the corridor walls in different directions and react to avoid them. At the same time, we want limit the complexity of the trials. In fact, situations in which the robot might need to move away from the exit to avoid a wall could be too tricky for the robot to solve as well as for the experimenter to fairly assess them. In order to achieve this tradeoff between situation diversification and straightforward execution, we decided to start the robot with an initial rotation such that it is always turned toward the exit direction (hence the angle among 120° and 240°). During each trial the robot must move along the corridor, avoid the collision with walls and reach the exit before the time of the experiment has expired. We empirically estimate the experiment length in *120* seconds (the speed of the robot with both the wheels activated is *5* cm/s, hence ignoring noise it needs 120 seconds to cover the entire corridor). In cases in which, during this time the robot hits a wall, the trial is immediately stopped.

- **Evaluation criteria:** the performance evaluation of each network is very simple thanks to the trivial task to achieve and to the experiment setup. Given the configuration described, the performance value assigned to each trial is merely the final distance of the robot to the center of the coordinate system, that is the corridor exit. Thus, the more the robot advances avoid-

ing collision along the corridor during a trial, the shorter is the final distance to the exit, the better is the trial evaluation. The evaluation can be seen therefore as the measure of the error committed by the network. Once we have the performance in each of the 6 trials, we finally assess the network on the basis of the arithmetic mean of these values. The learning process is carried out minimizing this average error value.

5.4 Phototaxis

In this Section we describe the second experiment of the thesis. We focus on all the settings needed to completely define the training process for the phototaxis task. However for a more exhaustive discussion about the experimental settings of this specific task we refer to [29].

5.4.1 Task Definition

The target task of this second experiment is another robotics basic task: *phototaxis*. Phototaxis is a kind locomotory movement, that occurs when an organism moves in response to the stimulus of light. Phototaxis is called positive if the movement is in the direction of increasing light intensity and negative if the direction is opposite. In our experiment the robot has to perform a form of positive phototaxis, moving toward the light source. In order to completely define the task, we describe its environment: the robot is placed into a square arena with edge of 5 meters positioned at a distance of 4.5 meters to the light source. During the experiment it has to perceive the light stimulus and move towards the light source, which is placed on top of one of the vertices of the arena (in particular the one centered in the origin of the coordinate system). The environment described is illustrated in Figure 5.4.

5.4.2 Robot setup

In this second experiment, the robot needs to perceive the light in order to and move toward it. For this purpose, the devices involved in the BN-robot mapping are only light sensors and wheels which permit to the robot to sense the light and change consequently the direction. The mapping: we use four frozen nodes to encode the eight light sensor states. Like for the first experiment, light sensors are gathered in pairs. For each couple of sensors, it is sufficient that one overtakes a

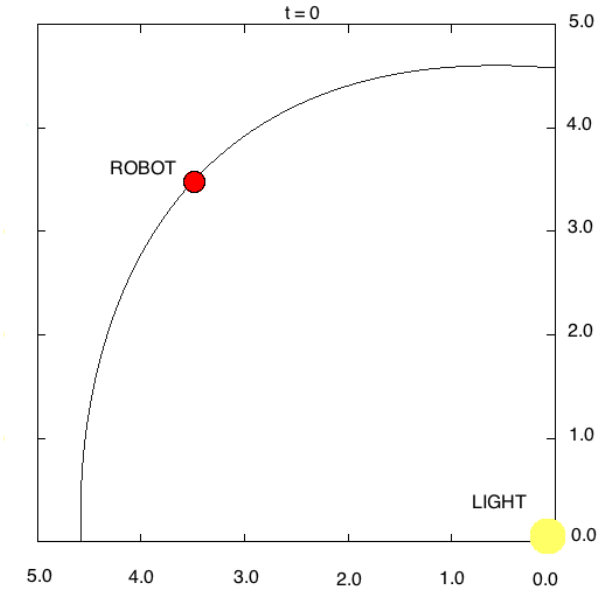


Figure 5.4: *Phototaxis arena environment*

chosen threshold of activation to make the corresponding input node assumes the value 1. The groups of sensors are chosen to ensure that robots can sense the light in the four directions north-east, south-east, south-west and north-west.

Moreover, we use two nodes as output nodes whose mapping is the same described for the first experiment and illustrated in Table 5.2. The Table 5.3 shows instead the input mapping between the Boolean network and the robot's light sensors.

5.4.3 Evaluation

The last aspects of the second experiment are the nature of the trials through which the behavior of each robot is tested and the evaluation criteria employed to assess the performance of the network:

- **Trials:** each robot is tested during the simulation of *10* different *trials*, which differ from each other for the initial conditions. In particular the robot is placed in *5 starting positions*, and for each of them *2 trials* are performed with 2 different initial orientations. The positions are chosen so that the robot starts *4.5* meters far from the origin of the coordinate system, on top of which is placed the light source. For this purpose, they are defined on a quarter of a circumference of radius *4.5* m centered in

Input node	Node value	Light state
x_0	0	No light sensed by IR0 or IR1
	1	Light sensed by IR0 or IR1
x_1	0	No light sensed by IR2 or IR3
	1	Light sensed by IR2 or IR3
x_2	0	No light sensed by IR4 or IR5
	1	Light sensed by IR4 or IR5
x_3	0	No light sensed by IR6 or IR7
	1	Light sensed by IR6 or IR7

Table 5.3: *Mapping between robot sensors and BN input nodes*

the origin. On this circle, we sample 5 points corresponding to 5 different angles with respect to the *z axis* (the normal axis to the plan on which the arena of Figure 5.4 is defined). More precisely, the angles range from *15°* to *75°* (with angles 0° e 90° the robot would be positioned on the edge of the arena) with steps of *15°*. The 10 initial rotation of the robot are instead chosen so that for each starting position, the corresponding two rotation are specular, i.e., differs from each other by 180°. During the trials, the robot has to sense the light and moves towards its, terminating the experiment as close as possible to its source. We empirically estimate that the amount of time required in order to properly perform such task in *120* seconds.

- **Evaluation criteria:** thanks to the experimental settings described so far, and to the trivial task, the performance evaluation of each network is very straightforward. Like for the obstacle avoidance, the performance value assigned to each trial is simply the final distance of the robot to the origin of the coordinate system, where the light source is placed. Therefore, the more a robot is able to narrowly perform phototaxis, the closer to the light it will be at the end of the experiment, obtaining a good evaluation. Again, given this criteria, the evaluation can be seen as the measure of the error committed by the network in each trial. The final performance of a network is computed, like in the first experiment, by means of the arithmetic average of the evaluations obtained during the 10 trials. The training process

try to iteratively minimize this value.

5.5 Results & Analysis

In this Section we present all the results obtained during the two experiments presented so far in the Chapter. Since the goal of the thesis is to investigate the most relevant properties of the training process, we focus on some relevant aspects in that direction, in particular concerning the learning process and the local search features.

The first thing we analyze are the final results obtained for each run executed. More precisely, in the plot reported below (Figure 5.5 (top) for phototaxis and Figure 5.5 (bottom) for obstacle avoidance), are depicted the the results of the best networks found at the end of each experiment. The plots exploit the expressiveness of *boxplots*. Boxplots represent the main statistical features of a distribution in a compact way, showing the median (central segment) and the 1st and 3rd quartiles (lower and upper side of the box, respectively). Minimum and maximum, along with outliers, are placed as segments and points external to the box.

The results are classified in different ensembles on the basis of the regime of the initial RBN from which the learning process starts. A consideration must be accounted about this classification, which we follow throughout the thesis: starting from a RBN in one of the three regimes, the training process works on the Boolean functions changing the network dynamic, and consequently its type of dynamical regime. For this reason, the final solution of each experiment can not be tightly expected to be in the corresponding dynamical regime.

Nevertheless, the networks generated by perturbing the initial random Boolean network maintain some properties of the initial dynamical regime. This is also proved by the results reported in the plots, which show the different success rate of the process when it starts from networks in different dynamical regimes. All the plots presented in the following exploit this assumption to classify the results.

The two plots, which depict the results after 1000 iterations of the process, show that phototaxis is the simplest task and all the three categories of initial RBN obtain a low error. This means that, except for few outliers with a higher error, the final networks are able to perform phototaxis and reach the light source. The boxplots concerning obstacle avoidance, show instead a different trend: the best results are achieved by the ordered networks, most of which reach the

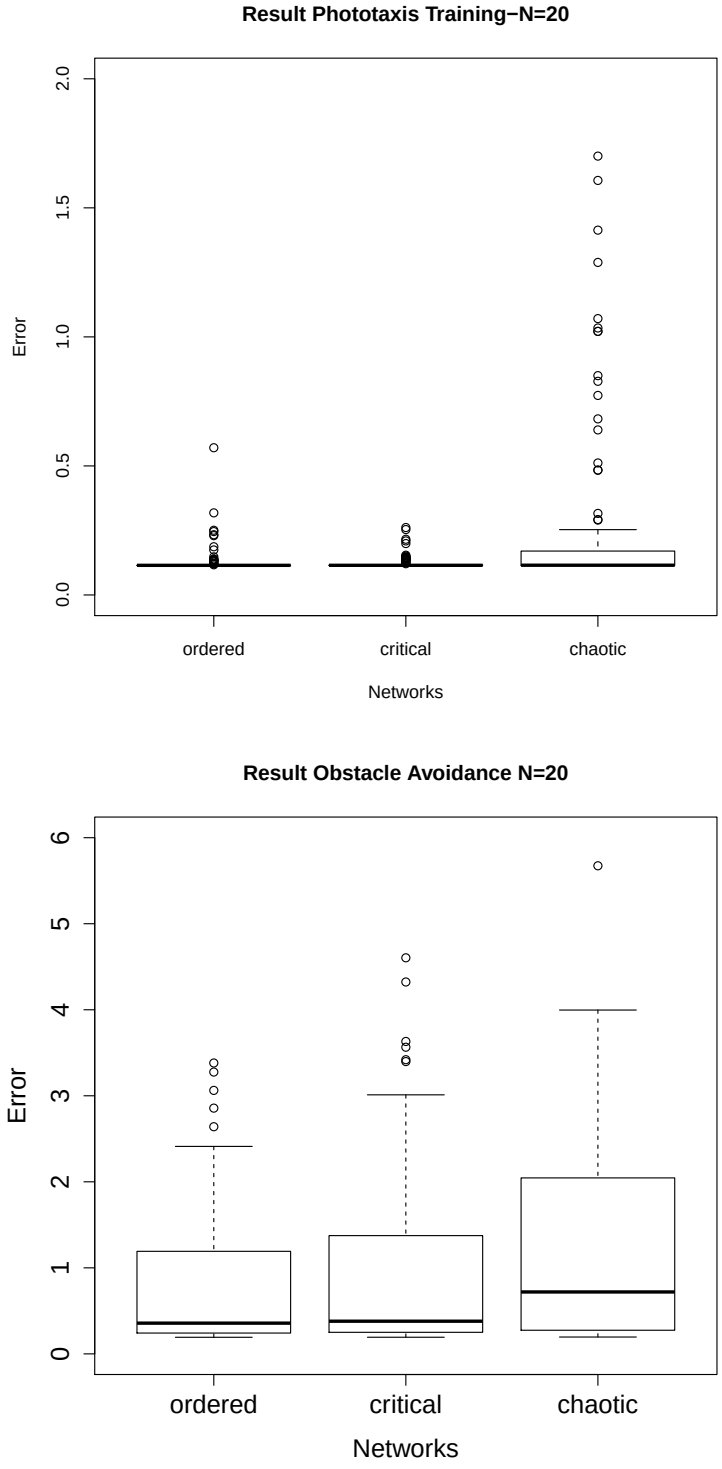


Figure 5.5: *Distributions of optimized network's error after 1000 iterations for phototaxis (top), and obstacle avoidance (bottom)*

target (if we choose an error threshold of 1 the success rate is ≈ 0.75). The worst results are obtained by the chaotic networks, whose success rate w.r.t. the same threshold is slightly over the 0.5 and the median is rather higher. The critical networks produce good results, even though not as much good as those achieved by the ordered ones.

Our hypothesis, to explain such difference between the results of the three types of networks, is that one of the reason can be the ruggedness of the search landscape (we have introduced the concept of ruggedness in Section 4.3.2). In fact, small perturbations in an ordered network's Boolean functions correspond to small variations in its BN dynamics, hence small differences in the objective function values. Conversely, slightly differing chaotic networks have a very different behavior, as discussed in Section 2.3. Thus, the initial search landscape is likely to be smooth in the case of ordered networks, whereas it is expected to be rugged for chaotic ones. Critical networks are expected to be in the middle of the two, with properties that do not excessively differs from the ordered ones. In order to verify the hypothesis the that initial networks dynamical regime affects local search effectiveness we need to compute a measure of the landscape ruggedness for each of the type of network.

The parameter that measure such property is the autocorrelation of the landscape, introduced in Section 4.3.2. For each networks dynamic class we compute the empirical autocorrelation obtained by collecting the objective function values along a random walk of 200 steps starting from 30 randomly generated initial candidate networks for each dynamical regime. For each random walk we calculate the value of autocorrelation according to the equation 4.1. The boxplots in Figure 5.6 summarize the statistics distribution of the values of autocorrelation of the landscapes induced by the three BN dynamical regimes (on the left the boxplots concerning the phototaxis and on the right the obstacle avoidance).

The first thing to notice in the plots is that the level of autocorrelation is high for both the experiments and, in general, for each class of network. This stems from the triviality of the task to be performed and leads to a high success rate for both the experiment. This result provide evidence to the fact that, for simple tasks and search landscape strongly autocorrelated, the automatic methodology proposed is very suitable, even using one of the simplest stochastic local search methods, in our cases the stochastic descent (Figure 5.2).

Concerning the plot of the phototaxis, the boxplots show slight differences between the three classes. However, these differences can not

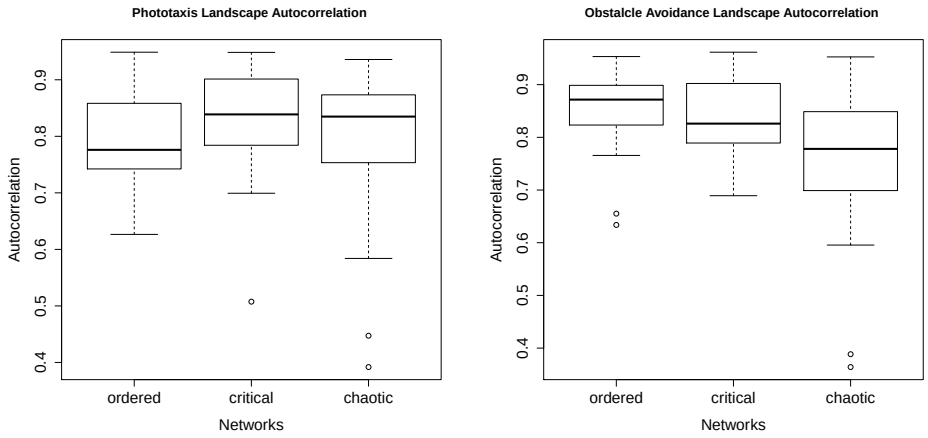


Figure 5.6: *Autocorrelation distribution of the landscapes corresponding to networks in different dynamical regimes. Phototaxis (top) and obstacle avoidance (bottom)*

be considered statistically significant. The situation changes focusing on the plots regarding the obstacle avoidance. Here the differences shown by the boxplots are more pronounced, specially between the ordered and the chaotic networks. In order to assert that the two distributions are significantly different, we use the Wilcoxon test, a statistical hypothesis test used when comparing two related samples on a single sample to assess whether their distribution means are equal. We perform the test for each pair of distribution and the results are reported in Table 5.4:

Networks		**Test Result**
Regime 1	Regime 2	p-value
Ordered	Chaotic	0.0008203
Ordered	Critical	0.2244
Critical	Chaotic	0.03997

Table 5.4: *Wilcoxon test results between initial network regimes*

Whereas usually is considered statistically relevant a *p-value* under *0.02* or *0.05*, these values provide an interesting overview: while the autocorrelation distribution of ordered and critical networks do not differ, the difference is quite relevant between critical and chaotic and

statistically significant between ordered and chaotic networks. These data support our hypothesis about the fact that initial networks dynamical regime affects local search effectiveness. Therefore, higher performance can be obtained where the landscape is smoother, i.e. typically with networks in ordered regime.

In order to support these hypothesis and analyze the robustness of the training process, we use the run-time distribution (RTD) method introduced in Section 4.3.1. RTD represent the probability to obtain a certain solution quality within a given time, or iteration. These distribution are computed, for each solution quality threshold, by counting the number of runs that reach the solution quality within a certain iteration, and repeating this procedure for a series of values of iterations. In our experiments, we use steps of *100 iterations* and *5 solution* quality thresholds (different between phototaxis and obstacle avoidance). Therefore, for each plot shown in Figure 5.7 we have 5 RTD curves, one for each error threshold. Furthermore, these plots highlight again the differences in the effectiveness of the training, during its progress, by starting from networks in different regimes. Figure 5.7 shows only the RTD curves concerning the obstacle avoidance, whereas they are the most interesting due to the higher complexity of the task to be performed.

The RTD trends confirm that the training process is faster when starting from networks in ordered regime and reach better final results. The critical networks have performance slightly worse but they do not differ significantly while with the chaotic the much lower slope of all the 5 curves proves that the training is slower and more complex.

Another interesting method to assess the robustness of the automatic process is the *testing* phase. It consists in testing the best solution achieved in each run over a series of trials, different from those used during the training phase, and measure its performance. Through this method, it is possible to realize the robustness of the training process, that is whether the resulting networks are able to generalize the task to be performed without fossilize on the training situations. One thing to point out is that the testing trials should be of the same nature of the training ones, in order to not introduce new variables that could distort the evaluation. The plot below (Figure 5.8) illustrates the results of the testing, classified according to the initial network regime, for the obstacle avoidance. They are obtained by testing each final network over 6 trials with different initial rotations w.r.t. the training ones, but chosen in the same range between 120 e 240 degrees). We do not show the results for phototaxis for

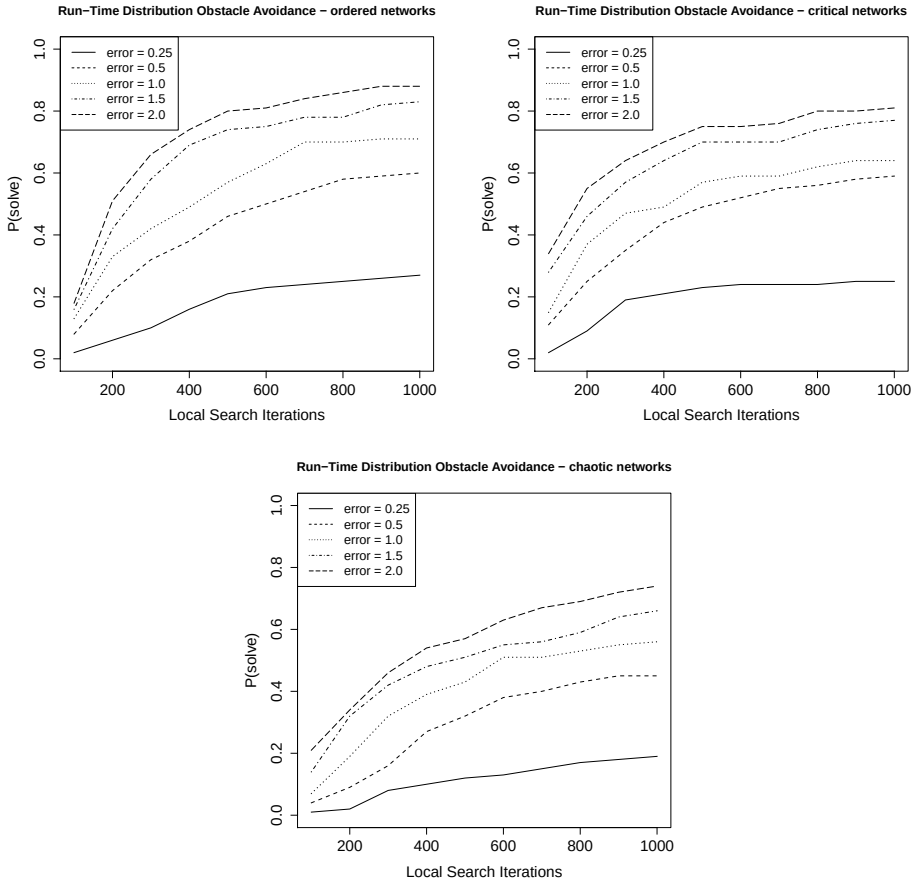


Figure 5.7: *Obstacle avoidance RTDs. Ordered (left), critical (right) and chaotic (center)*

reasons of brevity since they are very similar.

What emerges from the plot is that the results of the ordered and the critical networks remain good with a slight deterioration, proving that the most of the final networks of these runs are able to generalize the task. The worsening of the chaotic is more significant but, in general, the plot reveals a high degree of robustness of the process.

Given these results we want to focus now on the ordered networks, trying to figure out some aspect that can be the basis for their better performance w.r.t. the other network regimes. An interesting feature is the trend of the number of visited state by the network in its state space during the training process. Such property is an indirect yield of the optimization algorithm. In fact, our automatic design methodology works only on the Boolean functions, i.e. on the structure of

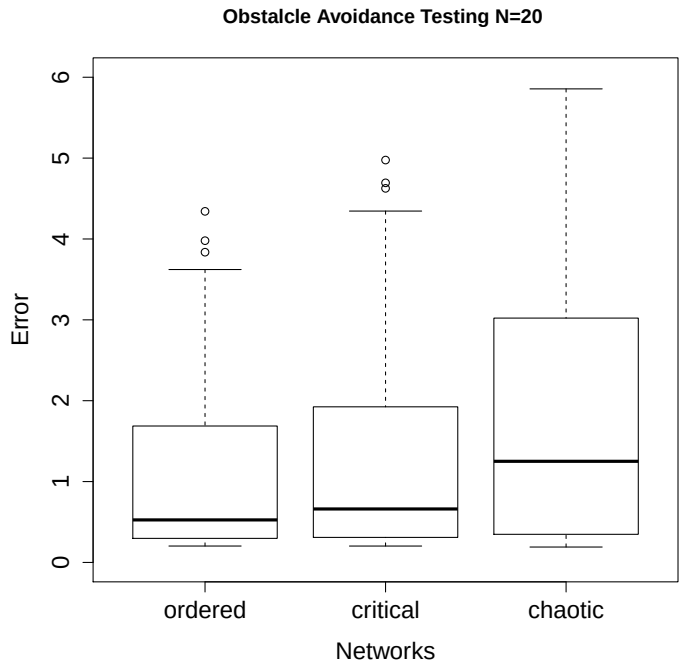


Figure 5.8: *Obstacle avoidance testing results*

the networks, without deal with space state concerns. The state space features that stem during the training process could be interesting in order to understand whether there exists a connection between the search process and the corresponding behavior of the candidate solutions inside their state space. In this context, by overlapping the trend of the number of visited state with the one of the objective function values for a typical case, we try to understand if the two quantities are related. Moreover, this method can reveal how the changes in the number of visited states affect the objective function trend. Figure 5.9 shows a typical case of overlapped trends during the obstacle avoidance training of an ordered network.

The plot suggests that the initial network visits a very limited number of states, as expected for an ordered network. Initially, the search algorithms try to quickly increase this number in order to have a wider number of possible dynamics and enhance consequently the probability to find a good solution. This mechanism can be seen as the method exploited by the search process to achieve the search diversification. Once a good solution has been found, we have a second phase in which the process tries to optimize this solution (intensification),

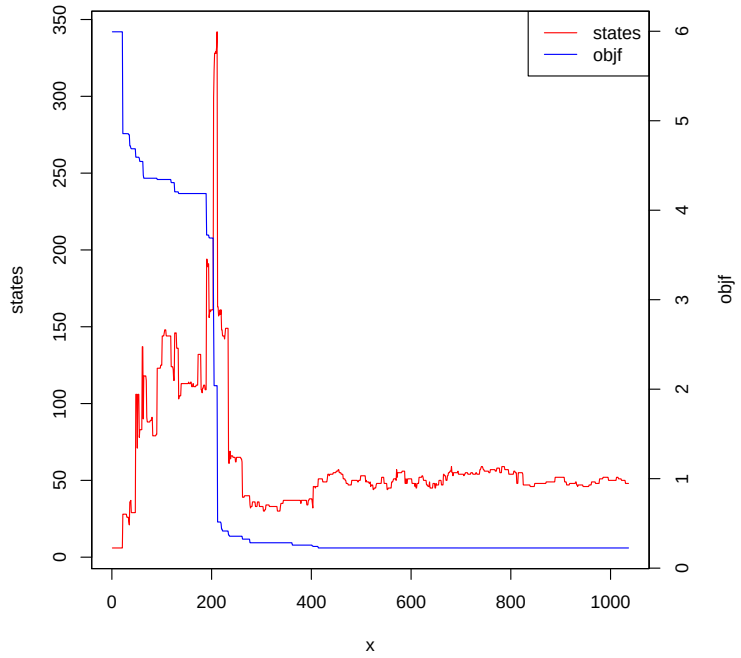


Figure 5.9: *Overlapped trends of visited states and objective function values during obstacle avoidance training (1000) iterations*

reducing and wrapping the visited states. In this phase the value of the objective function is subject to small changes. Finally, both the objective function and the number of states are stabilized, the latter in particular, on a number extremely low (≈ 50) with respect to the dimension of the state space (2^{20}). This means that the design process is able not only to achieve solution that perform the target task, but it shows also the ability to optimize the utilization of the state space. This trend is typical of the ordered network, while the other network have different behaviors inside the state space during the training. In particular, as Figure 5.10 highlights in a typical case, with chaotic networks the number of nodes steadily increases and never converges whereas the critical network remains between the other two.

The number of states assumed by the solutions during the training is not the only dimension that we can analyze. In particular, we want now to show the connection between the progress of the automatic design process and another dimension, that is the *statistical complexity* [40]. The statistical complexity is computed on the basis of two

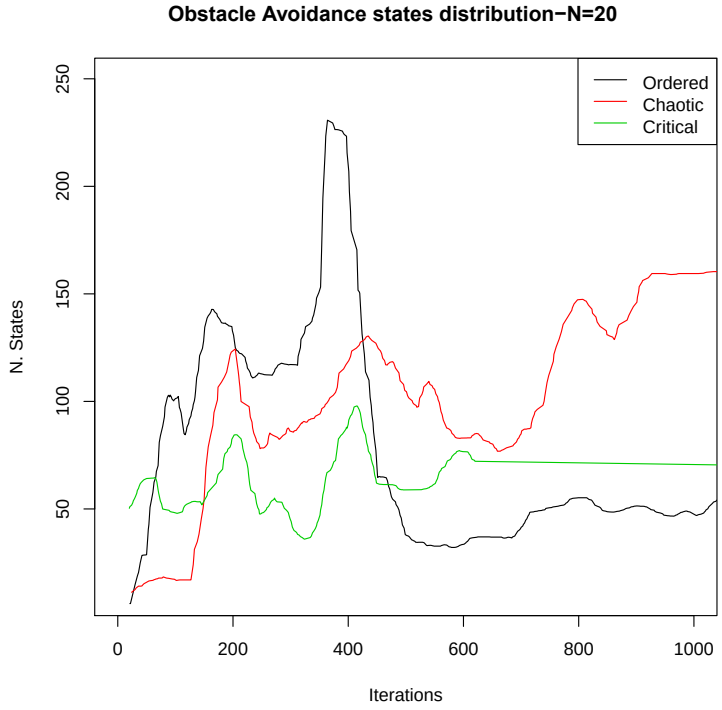


Figure 5.10: *Number of visited state's trends comparison between the three different network regimes during the training*

other quantities: *entropy* and *disequilibrium*. The former is the measure introduced by Shannon [41] which quantifies the expected value of the information contained in a random signal, or in a system. The entropy of a system with N accessible states with probabilities to be visited $p_1, \ldots, p_N$ is calculated as follows (binary alphabet used to encode the states):

$$H = -\sum_{i=1}^{N} p_i \log_2 p_i \tag{5.1}$$

The latter is a kind of distance from the actual system configuration to the equilibrium (the more states the system assumes, the further it is from the equilibrium). The disequilibrium is calculated as follows:

$$D = \sum_{i=1}^{N} (p_i - \frac{1}{N})^2 \tag{5.2}$$

The complexity is defined as the product between entropy and disequilibrium: $C = H \times D$. For our analysis we compute the complexity value of each candidate solution encountered during the search process, by measuring these quantities on the trajectory performed by the network over the state space. Therefore we have the complexity trend during the whole training.

What we want to prove is that there exists a correlation between the complexity trend and the significant variation of the objective function value. In particular, we are interested in highlight the fact that relevant improvements of the objective function correspond to a complexity increase. In order to do that, we exploit a measures employed in the signal processing discipline, the *cross-correlation* [42]. Cross-correlation is a measure of similarity of two waveforms as a function of a time-lag applied to one of them. Computing the discrete cross-correlation between the complexity and the objective function trends, we obtain for each run of the experiments a measure of the similarity of the two signal over a time series. One thing to remark is that these values refers to the measure of cross-correlation corresponding to a subset of the improvements, since we want to focus on the most relevant ones. To this end, for each run, we sample the values of the two trends at each improvement at least as significant as the tenth, sorting by relevance, of the run. Since we are interested in the simultaneous correlation in the two trend changes we focus, for each run, on the value of cross-correlation corresponding to $Lag = 0$. Finally, we plot all the values obtained through a boxplot that can reveal the distribution properties. Figure 5.11 shows the results both for phototaxis (top) and obstacle avoidance (bottom).

The plots show that, in general, there exists an anti-correlation rather clear between the two trends, which means that relevant improvements of the objective function correspond to a complexity increase. In particular the median value of correlation is equals to -0.6 ($r = -0.6$), and the distribution is compressed towards the value -1. Moreover, where the correlation is positive the value is low, except for few outliers, and therefore not very relevant.

This is an interesting result which could mean that the training process for a given task involves an increase of the complexity of the system, which leads to a greater learning ability. One attractive idea could be to exploit this trend to improve the search process that underlies the automatic training process, for instance by adding a term in the objective function which rewards candidate solutions with higher complexity.

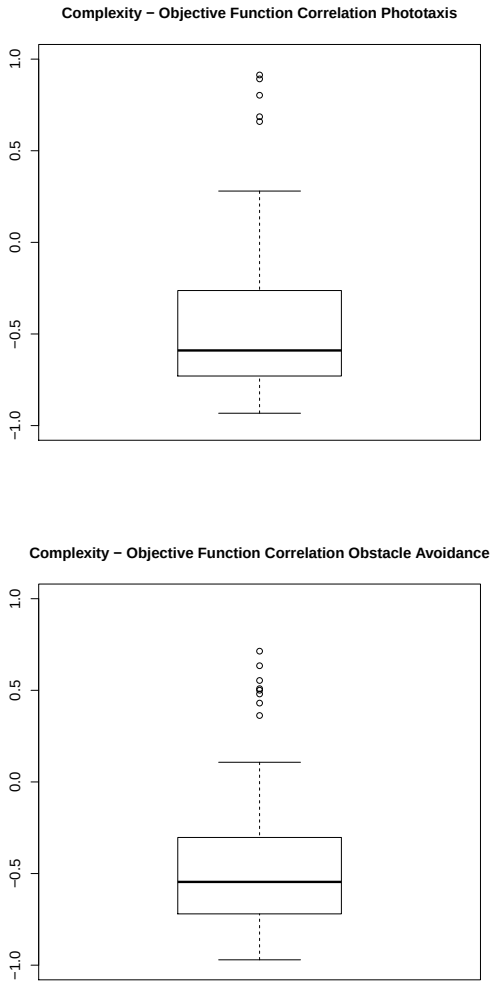


Figure 5.11: *Distribution of Lag 0 correlation between complexity and objective function values for phototaxis (left) and obstacle avoidance (right)*

Last thing we discuss about the analysis is how the number of nodes of the Boolean networks employed in the process affect the local search performance. For this purpose, we increase the number of nodes from 20 to 40 ($N = 40$) and we execute a new training process. The result expected is that with the same number of iteration used with 20 nodes, i.e. 1000, the local search algorithm could not be able to reach quality results as good as the previous with 20 nodes. The reason is that the search space and the state space of the networks increase their

dimension by a factor 2^{20}. Therefore the search algorithm moves in a much wider space and handles solutions with more variegate dynamics.

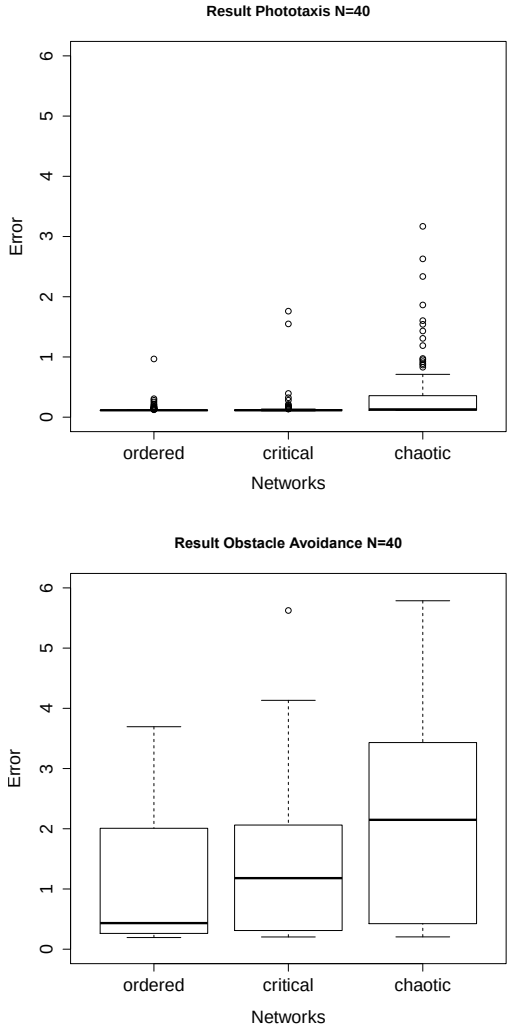


Figure 5.12: *Distributions of optimized network's error after 2000 iterations for phototaxis (top), and obstacle avoidance (bottom). N = 40*

Given these issues, we choose to increase the local search iterations to 2000 and analyze the results (Figure 5.12).

The boxplots show that the results are similar with respect to the ones with N = 20 for the ordered networks while are rather worse for the critical but specially for the chaotic ones. Once again this proves that the initial network dynamic regime affect the training process

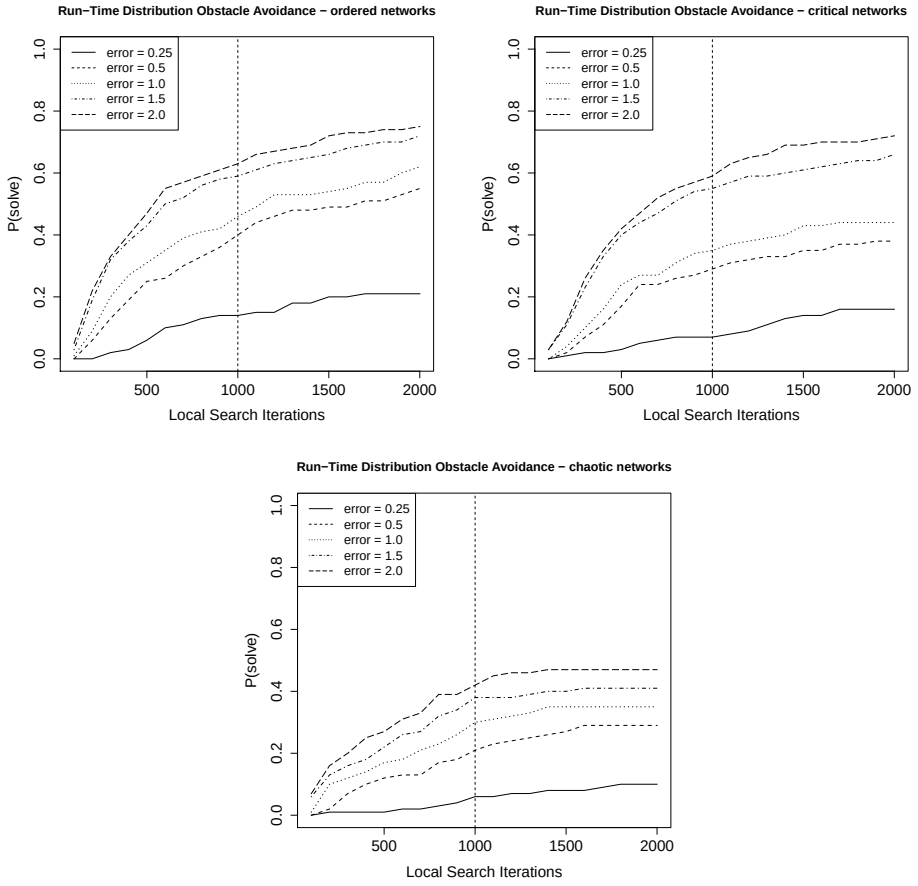


Figure 5.13: *Obstacle avoidance RTDs with N = 40. Ordered (left), critical (right) and chaotic (center)*

and the quality of the results. These results allow us to state that the number of nodes of the networks has relevant consequences on the training process performance. In other words, the number of nodes must be dimensioned with respect to the target task complexity so that to obtain the right tradeoff between the required computational capacity to perform it and a space not too wide which favors the search. The last plots we report, in Figure 5.13, are intended to support this argument, illustrating the obstacle avoidance RTDs over the 2000 iterations. In fact, the curves show how the process is slower, specially for the chaotic, compared to the training of 20 node networks.

Chapter 6

Dynamic Tasks Learning

In Chapter 5 we undertook two robotic experiments in order to validated our methodology on two simple target task and investigate some interesting aspects of the training process. In this Chapter we move to a much more complex target task: *dynamic task learning* or *task sequence learning*. The first goal is to verify the robustness of the training process in tackling the issue of design BN robotic programs for complex tasks, confirming the interesting features revealed in Chapter 5. The second goal is to improve the methodology by exploiting some advanced stochastic local search methods and compare their results.

6.1 Introduction

This chapter aims at testing our methodology on a complex target task in which the training process has to tackle the issue of designing BN networks with some advanced dynamical characteristics such as a form of state memorization.

Beside the validation, the chapter aims at improving the training process by means of some advanced stochastic local search methods. Analyzing the different results we focus on the properties of such methods that are shown to be more effective on the performance. In particular, we want to support the hypothesis that a restart mechanism, such as the perturbation performed in the iterated local search methods, can profoundly improve the results. In general, our speculation is that the process should alternate deep intensification of the search in order to reach a local minima, to diversification which leads the search to visit other regions in order to escape from the local minima. The simple algorithm employed for the test cases lacks with respect to the second aspect, since when it reaches a local minima it has no

mechanism that can lead the search out of it.

Finally, we focus on the properties of the training process already considered for the test cases: the relation between the improvements of the objective function and some properties of the networks, in particular in their state space, such as the number of visited state during the execution and the system complexity. The goal is to confirm the considerations drawn for the test cases: the first is that the search needs to explore regions with network characterized by a rich dynamic in order to significantly improve the solution and subsequently optimize it by reducing the space used in the state space. The second is that there exists a correlation between the relevant changes of the objective function and the complexity of the network trajectories.

6.2 Task Definition

Sequence learning is one of the most prevalent form of human and animal learning. It is a fundamental part of everyday human activities that range from reasoning to language, from common skills and abilities to complex problem solving. Sequence learning is a relevant component in many domains and scientific disciplines, such as planning, reasoning, robotics, natural language processing, speech recognition, adaptive control, time series prediction, financial engineering, DNA sequencing. Clearly, each of these different perspectives leads to a different approach in tackling the sequence learning problem. These approaches, stemming from various task domains, deal with differently formulated sequential learning problems and different aspects of sequence learning.

With some simplifications we can characterize the different formulation of the sequence learning in four main categories [46], that can be formally defined as follow (in each case, $1 \leq i \leq j < \infty$):

- *Sequence prediction:* $s_i, s_{i+1}, ..., s_j \rightarrow s_{j+1}$. That is, given s_i, s_{i+1}, ... , s_j, we want to predict s_{j+1}. When $i = 1$, we make predictions based on all the previously seen elements of the sequence. When $i = j$, we make predictions based on only the immediately preceding element.

- *Sequence generation:* $s_i, s_{i+1}, ..., s_j \rightarrow s_{j+1}$. That is, given s_i, s_{i+1}, ... , s_j, we want to generate s_{j+1}. (Clearly, sequence prediction and generation are essentially the same.)

- *Sequence recognition:* $s_i, s_{i+1}, ..., s_j \rightarrow$ *yes or no*. That is, given s_i, s_{i+1}, ... , s_j, we want to determine whether this subsequence is legitimate.

- *Sequential decision making* (that is sequence generation through actions): $s_i, s_{i+1}, ..., s_j; s_G \rightarrow a_j$. That is, given $s_i, s_{i+1}, ..., s_j$ and the goal state s_G, we want to choose an action a_j at time j that will likely lead to s_G in the future. In particular, in the *trajectory based* version we have $s_i, s_{i+1}, ..., s_j; s_{j+1} \rightarrow a_j$, that is, given $s_i, s_{i+1}, ..., s_j$ and the desired next state s_{j+1}, we want to choose an action a_j at time j that will likely lead to s_{j+1} in the next step. From these definition is noticeable that the sequential decision making formulation is the most general and it subsumes all the others.

Sequence learning is clearly a difficult task, due to the fact that forms of memory structures are needed. Several techniques exist to tackle the problem, including recurrent neural networks, hidden Markov model, dynamic programming, reinforcement learning, evolutionary computational models and so on. However, even if the sequence learning has been often under-estimated in AI and machine learning [46], it is still an open problem in many disciplines.

The target task of the last robotic experiment is a form of sequence learning. More precisely, the robot has to learn to recognize a sequence of colors, by performing certain actions. The environment of the task is a straight corridor of length *7* meters and width *0.5* meters with the exit in the origin of the coordinate system (which is the coordinate system provided by the ARGoS simulator used to simulate the experiment [31]). Along the corridor, the ground is composed of three different color: the white color is the background while the black and the gray colors are the symbols" that can compose a sequence. In particular, the colors are disposed in such way to realize a striped ground, with stripes *0.5* meters wide orthogonally arranged with respect to the corridor direction. The stripe pattern is realized by alternating a white stripe with a black or gray stripe. In this environment the robot, placed within the corridor *6.5* meters far from the exit on the white color, has to move along it, turning the LEDs on when it encounters a black or gray in the right sequence and keeping the LEDs off when the color is not in the right sequence. The sequence of task to perform, in our case, is a cyclic repetition of black and gray. On the background color, i.e. the white, the robot has to behave as well as if it was on a wrong color of the sequence, that is keeping its

LEDs off. An illustration of the environment described is shown in Figure 6.1. The sequence of colors on the floor is just an example of the possible combinations that can be proposed to the robot on its path.

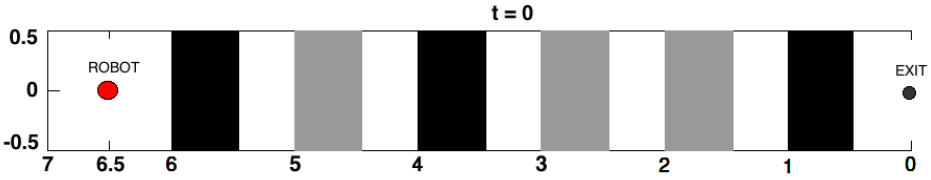


Figure 6.1: *Sequence learning task corridor environment*

The task described can be seen as a form of dynamic task, in which the robot, i.e. the network that drive the robot, needs to take decisions dynamically about which task to perform next, on the basis of the tasks it performed in the past. In other words, the network needs to exploit some form of memory in order to know which task has been already performed (turning the LEDs on), and consequently which is the next task (the next color in the right sequence). This complex task can easily be ascribed to the category of sequential decision making discussed, in particular, to the trajectory based version. In fact, the task of the robot can be seen as a sequence of decisions to make, one for each color it encounters, which consist in turning the LEDs on on the right color and keeping the LEDs off on the wrong, i.e. the color not in sequence.

6.3 Settings

Once the task to be performed is defined, we discuss the experimental settings required to completely define the training process. In the following of this Section we present the setup for the Boolean networks and for the robot, and finally we describe the evaluation methods. The last component of the training, i.e. the stochastic local search methods employed, will be discussed in Section 6.4, so as to discuss one of the most prominent aspects of the thesis in depth.

6.3.1 BN & robot setup

Again the process starts from an initial random Boolean network. In order to achieve the last task, much more difficult with respect to the

test cases, we choose as first attempt to increase the number of nodes to 30, i.e., $N = 30$. The topology, i.e. the inputs, of the networks are randomly generated with $K = 3$, which means that each node has 3 ingoing arcs. Boolean functions are also randomly generated with a p, i.e. the probability to have an entry with 1 as output value in the Boolean function of a node, which varies. In particular, unlike in the test cases, we generate 30 initial networks with $p = 0.9$ (ordered regime) and 30 networks with a $p = 0.788675$ (the value which verifies the critical line equation 2.1). This choice of reducing the number of networks is due to the increased number of process iterations required to reach good results. Hence also the choice of dismissing the chaotic networks, proved to be the least relevant in the test cases, is intended to economize computational and time resources.

The network are used then as the robot program by defining a suitable mapping between the robot sensors and the network input nodes and between the actuators and the subset of output nodes. In this last experiment, in which the robot needs to move along a corridor, detect the colors of the ground and turn the LEDs on when it encounters the right color of the sequence, the devices employed are: the ground sensor, able to detect the color the robot is moving on, the wheel motors and the LEDs actuator. We include in the mapping also the proximity sensors because moving in the corridor the robot might need, if its trajectory is not straightforward, to avoid the impact with the walls. The mapping is defined as follow: four frozen nodes encode the eight proximity sensor states (this mapping is the same as that used for the obstacle avoidance in Section 5.3.2). The color detected by the ground sensor is encoded in two input nodes since we need to encode just three possible colors (white, black and gray). Finally, we connect two output nodes to the wheel motors, that hence can assume only the value ON or OFF, and one output node that trigger the eight LEDs behavior (the robot does not need to use single LED separately). The complete mapping between perceptions/actions and node values is shown in Table 6.1, Table 6.2, Table 6.3, Table 6.4.

6.3.2 Evaluation

Once we have defined the mapping between the robot and the BN program, we execute the BN in such way that its dynamic controls the robot behavior. The experiment is then simulated over a series of trials, which differ for the initial condition, and the BN is evaluated according to the specific target requirements. In order to completely

Input node	Node value	Proximity state
x_0	0	No obstacle detected by IR0 or IR1
	1	Obstacle detected by IR0 or IR1
x_1	0	No obstacle detected by IR2 or IR3
	1	Obstacle detected by IR2 or IR3
x_2	0	No obstacle detected by IR4 or IR5
	1	Obstacle detected by IR4 or IR5
x_3	0	No obstacle detected by IR6 or IR7
	1	Obstacle detected by IR6 or IR7

Table 6.1: *Mapping between robot proximity sensors and BN input nodes*

Input nodes	Node value	Color
x_4	0	BLACK
x_5	0	
x_4	0	GRAY
x_5	1	
x_4	1	WHITE
x_5	1	

Table 6.2: *Mapping between robot ground sensor and BN input nodes*

Nodes		Actuators	
x_6	x_7	Left wheel	Right wheel
0	0	OFF	OFF
0	1	OFF	ON
1	0	ON	OFF
1	1	ON	ON

Table 6.3: *Mapping between robot wheel actuators and BN output nodes*

define this process we describe the nature of the trials and the evalu-

Node	Actuators
x_8	LEDs
0	OFF
1	ON

Table 6.4: *Mapping between robot LED actuator and BN output node*

ation criteria:

- **Trials:** in the corridor arena described in Section 6.2, the robot behavior is tested over *10 different trials*. At the beginning of each of these trial, the robot is placed in the centre of the corridor *6.5* meters far from the exit (*6.5,0,0*) and with an initial rotation of *180°*, i.e. perfectly directed towards the exit and the origin of the coordinate system. Thus, differently from the test cases, the robot's initial position and rotation remain the same in all the trials. This choice aims to focus the process only on the sequence learning, without increase the complexity by involving in the experiment other tasks, such as the collision avoidance. However the robot is not prevented to move towards the walls as long as it succeeds in avoiding them thanks to the proximity sensors. What differs between the 10 trials is the pattern of colors placed on the ground of the arena. More precisely, for each trial we realize a different sequence of length 6 with color black and gray, ignoring the white stripes between them. In the arena depicted in Figure 6.1 is shown one of the possible sequences realized. During each trial, the robot has to advance along the corridor and choose the right action to perform for each color it encounters during its movement. In case of error in these choices, the trial is immediately stopped. The errors that a robot can make are listed below (in the definition we use the term *colored* to identify black or gray stripes):

  - crossing, with the LEDs on, the colored stripe not in the next sequence position. More precisely, we consider an error when the robot keeps the LEDs on for a fraction of the total time it spends on the color, greater or equal than a given *threshold*. We set this threshold to a fraction of *1/10* of the total time on the color.

  - crossing, with the LEDs off, the colored stripe in the next sequence position. Again we decide whether an error occurs according to a time fraction threshold. In this case the robot has to keep the LEDs on, on the right color, at least for a time fraction of *2/3* of the total. Otherwise it is considered an error.
  - keeping the LEDs on for a fraction of time equals or greater than *1/10* of the total time it spends on a white stripe.
  - colliding with a wall.

  Given these definitions, we consider that the robot recognize a color in the right sequence if, and only if, it spends at least 2/3 of the total time on the color with the LEDs on. Conversely, we consider a color correctly not recognized when the time with the LEDs on is less than 1/10 of the total. The choice of the first rule aims at reaching a good tradeoff between a strict policy that do not allow the robot to make wrong choices and a relaxed line that grant few errors that might be due, for instance, to the network dynamic delay. The network dynamic, in fact, might take more than one tick to propagate a decision to the output from the instant of time in which an input changes. Employing a too strict policy in such cases, the network would be unfairly penalized. This second restriction, instead, is required in order to obtain results with visually different behaviors on color in sequence or not in sequence. Summarizing, for each of the 10 trails, the robot starts with the same initial position and rotation but with different colored pattern on the floor. During the trials it has to advance along the corridor towards the exit by recognizing the right sequence of colors (a cyclic repetition of black and gray) and, if necessary, avoiding collision with walls. Otherwise, in all the cases listed above, the robot is immediately stopped in its current position and the trial ends. We empirically estimate the amount of time required in order to cover the entire corridor in *130* seconds. Like in the other experiments, the simulation is time discrete, which means that each second a certain number of steps (or *ticks*) are performed. During each tick, the robot perceives the environment through its sensors, selects the action to perform, and finally actuates it.

- **Evaluation criteria:** given the configuration described, the performance value assigned to each trial, that is the objective

function to be minimized by the training process, is defined as follow:

$$E = 1 - (\alpha \cdot (1 - d) + (1 - \alpha) \cdot \frac{t_r}{t_{tot}}) \tag{6.1}$$

In the equation 6.1, d is the normalized distance of the robot to the center of the coordinate system (the corridor exit) at the end of the trial. The term t_r t_{tot} represents the fraction of the total time of the trial, in which the robot has a good behavior, in terms of the state of the LEDs. In other words, t_{tot} is the total length of the trial, in terms of the number of ticks elapsed before the termination (due either to the expired time or an error occurred among those listed above). t_r, instead, counts the number of ticks during which the robot maintains the LEDs in the right state: off when it is on the color not in sequence and on the white color, and on when it is on the color in sequence. This fraction is particularly useful to distinguish between network that commit an error at the same distance from the exit: this situation, in fact, is very likely due to restrictive policy employed. Thanks to this term we can therefore provide higher rewards to networks that have better performance on the color than networks that reach the same distance but with worse performance. The term has value 1, when a robot perfectly actuates its LEDs during its life". Note that the performance defined as in equation 6.1 allows the experimenter to favor the final distance term with respect to the one of the LEDs performance changing the value of α. For our experiments we use an $\alpha = 0.8$ in order to favor the term of the final distance of the robot to the exit, since this term is the one that best fits the performance measure concerning the sequence recognition. The other term is useful, as mentioned, for the discrimination between networks with similar behavior.

Once we have the performance in each trial, the final evaluation assigned to a network is given by the worst case, which means the worst performance obtained in the 10 trials. This choice, different from the one made for the other experiments, aims at not suffering the likely variability of the 10 performance measures. The different colored patterns, in fact, are likely to cause large differences in the 10 evaluations of the same network. The worst case performance selection, differently from the average, is intended to limit this problem and finally provide networks

with good behavior in all the situations.

6.4 SLS Techniques

In this Section we describe, with respect to the sequence learning task, the aspect of the methodology that is the heart of the thesis: the metaheuristic techniques employed for the automatic design of BN robot controllers. In particular, for this experiment we use four different SLS methods in order to exploit their different properties and better understand the features that can be useful for this kind of problems. In the following we will describe each of the techniques, while for the results and the final comparison we refer to the Section 6.5.

SD

The first algorithm employed is the one we used for the two test cases, i.e. the Stochastic Descent (SD), which despite its simplicity proved to be very effective. A general outline of the algorithm is depicted in Figure 5.2. In our case, the initial solution is randomly generated according to the settings described in Section 6.3. The neighborhood relation is defined, like in the test cases, by randomly choosing a node and then flipping a random bit in its Boolean function. The pair formed by the node and the Boolean function position is uniformly sampled with replacement, which means that once a neighbor has been sampled and rejected we do not remove it from the neighbors ensemble. The consequence is that the neighbors ensemble will never be empty and the algorithm will be executed until the timeout expiration. Again, we accept a new candidate solution even if its evaluation is equal or slightly worse (by an *epsilon = 0.0001*) than the current one. These sidewalks aim to prevent the instant stagnation of the search process inside the plateau regions and to enhance the exploration capacity of the algorithm.

ILS+SD

The second version of algorithm used for the training process is an hybrid technique which exploit the straightforward stochastic descent along with a mechanism for effectively escaping from the local minima in which the SD can get stuck: iterated local search. The idea is therefore to intensify the search with the SD until it reaches a local

minima, and then apply a perturbation to the current optimal solution to diversify the search and hopefully reach other improvements. The outilne of the algorithm is the one illustrated in Figure 4.4. The internal local search procedure is the stochastic descent described above with a simple variation: we allow the sidewalks but if the local search can not find an improvement solution within a certain number of steps, we assume that the search is stocked. In such cases, we stop the search procedure to exploit the ILS perturbation mechanism in order to escape from the local minima. The perturbation method is chosen so that it is not too close to a random restart, while keeping the perturbation computationally fast and easy to implement. It consists of performing a random flip in the Boolean function of each node. The acceptance criterion is the same employed for the internal SD: accept a new candidate solution if it is better, equal or slightly worse than the current best one. Again this choice aims to achieve a substantial search diversification, since the new cycle of perturbation and local search will start from the new current candidate solution.

ILS+VND

As discussed in Chapter 4, there exist several methods designed to help a local search procedure to escape from a local optima. Another method we employ, beside the ILS perturbations, is based on the idea of changing the neighborhood relationship whenever the search gets stuck. This idea is at the basis of the Variable Neighborhood Descent (VND) algorithm, whose outline is shown in Figure 4.1. In particular, in our solution, the neighborhood relations are characterized by the number of random flip performed to obtain from the current solution the neighboring one. The algorithm starts with neighborhood N_1, that is the simple neighborhood defined by randomly choosing a node and flipping a random bit in its Boolean function, like in the SD. Whenever no further improvements are found for a certain number of steps, the algorithm continues the search with the neighborhood N_2, which means performing two random flips at a time and so on until the last neighborhood relation defined k. If the search can not still find further improvements within the maximum number of iterations, the search is stopped. During this process, whereas an improvement solution is found, the neighborhood relation is switched back to N_1. Once again we decide to allow sidewalks in order to move the search process as much as possible but without, in these cases, switch the neighborhood back to N_1. The algorithm described is employed as the

local search procedure inside the ILS mechanism in order to exploit its properties inside a method that allow an higher search diversification. ILS, in fact, can be very effective in escaping from deep local minima in which the VND mechanism is not enough. The ILS perturbation and the acceptance criterion remain the same of the ILS+SD.

GA

Another prominent category of metaheuristics, are the Genetic Algorithm (GA) introduced in Section 4.2.3. The general scheme of GAs is illustrated in Figure 4.6. For this work, we implement a GA through the open source library EALib [47], which provides the basic primitives to develop an evolutionary algorithm. More precisely, the algorithm employed is a simple GA, with a two parents single-point crossover, mutation that consists in flip a randomly chosen bit in the Boolean function of a random node and elitism. The selection is performed by a roulette wheel: the probability of each individual of being chosen to reproduce in the next generation is proportional to its fitness value (objective function value).

6.5 Results & Analysis

In this Section we present the results obtained in the sequence learning training. First, in Section 6.5.1, we discuss the results comparison between the different stochastic local search techniques employed. Finally, in Section 6.5.2, we analyze other prominent aspects emerging from the results. This analysis aims to confirm the results obtained in the test cases, such as the correlation between the network dynamics complexity and the objective function improvements.

6.5.1 Algorithms comparison

In order to explore the potential of improving our methodology by means of advanced stochastic local search techniques, we analyze the results of the four algorithms described in Section 6.4 in the training for the complex sequence learning task. Below we briefly describe the experimental settings for each technique:

- **SD:** the algorithm has been described in Section 6.4. We execute *60* independent experiments and each of them corresponds to an initial different BN (randomly generated as described in

Section 6.3). As mentioned, the algorithm samples a neighbor from the neighbors ensemble with replacement. This means that the termination condition is simply the chosen timeout. In particular, we execute for each experiment a number of iteration equal to *100000*.

- **ILS+SD:** the version exploits the simplicity of the SD search procedure along with the potential of a restart mechanism typical of the ILS. More precisely, the version employed is characterized by a number of external cycles, i.e. of restarts, of *10 iterations*. The internal local search procedure is the same SD described above with a cutoff of *50000 iteration*. As mentioned in Section 6.4 , the unique variation is that the local search is stopped also if a number of iteration without an improvement is reached. For this version we use a non-improvements timeout of *20000 iterations*. We run *60* independent experiment.

- **ILS+VND:** this second version of ILS, whose external iterations are also *10*, makes use of a different local search procedure based on the basic idea of the VND. The algorithm passes from a neighborhood to the next (i.e. from a number of flips to the next, as explained in Section 6.4) whenever the maximum non-improving steps are achieved, in our version *5000 iterations*. If a new improvement is found the algorithm switches back to the first neighborhood and the non-improvements counter is set to *0*. Five neighborhood relations are defined, from *1 flip* to *5 flips*. Whether the non-improving timeout is reached with the last neighborhood, the local search procedure is stopped. However, the general timeout of *50000* iterations also stands. Like for the other methods, *60* independent experiment are executed.

- **GA:** the GA is a simple GA with a population size of *20 individuals*. The *elitism* is set to *2*, which means that the best 2 individuals of the current population are automatically chosen for the next generation. This ensures that the best candidate solutions are always selected. Again we execute *60* experiments corresponding to an initial different BN generated as described in Section 6.3. Each of these random BN represent an individual in the initial population. The others *19* are also randomly generated by the library primitives [47]. Concerning the genetic operators, the mutation consists in flip a randomly chosen bit with probability $p_{mutation} = 0.02$ while the single-point crossover

has probability $p_{crossover} = 0.1$. The selection is a proportional selection performed by a roulette wheel.

Given these experimental settings, the first thing we present is the final error distribution comparison of the 60 independent runs of each algorithm. These results are shown in Figure 6.2. Each boxplot corresponds to the 60 final error distribution of a different algorithm.

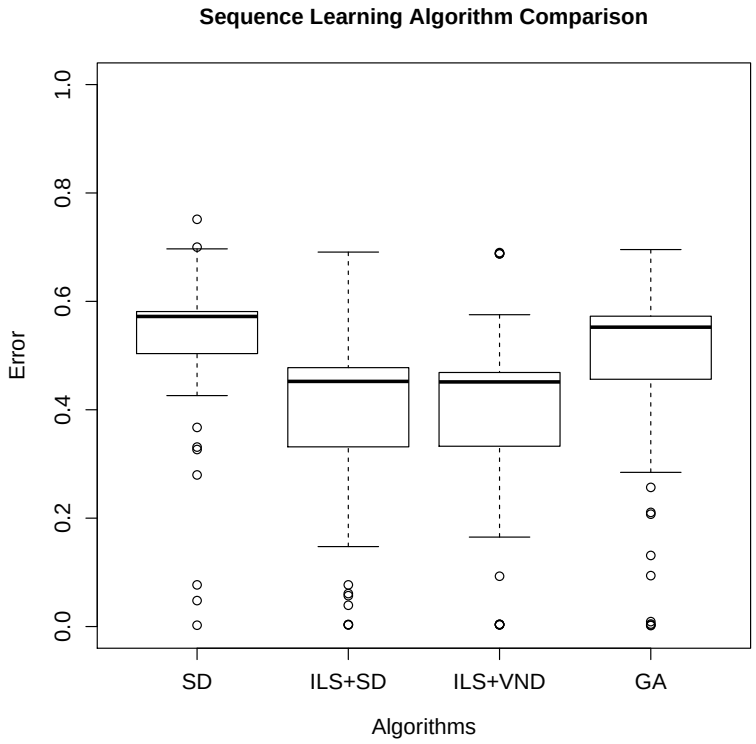


Figure 6.2: *Sequence learning task corridor environment*

In general, the boxplots confirm the complexity of the task, since the errors are mostly distributed on high values. One thing that must be said is that the results under the value of 0.3 are all working, which means that the robots are able to perform the task correctly. The variability between 0 and 0.3 is due only to a different speed of the robots along the corridor. Robot with an evaluation of 0.3 are slightly slower than those with a lower error, and hence they are not able to reach perfectly the corridor exit, even if their behavior is correct in terms of sequence recognition. For this reason we report below in Figure 6.3 a histogram which shows the number of results of each

algorithm in the range *0.0 - 0.3*, i.e. the number of final results able to perform the target task.

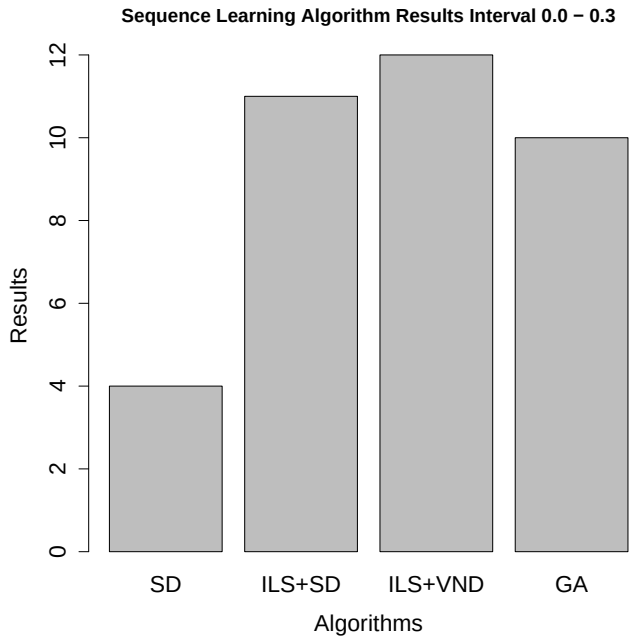


Figure 6.3: *Final results in range 0 - 0.3 for each algorithm*

The histogram supports the information emerged from the boxplots: the worst algorithm is the simple SD, whose median error is just under *0.6* and the number of working results, just *4*, is the lowest. The GA errors distribution is slightly better with a number of good results that is up to *10*. The GA feature which probably leads to the advantage with respect to the SD is the implicit search diversification through which the search process cover more extensive regions of the search space. However, the two best algorithms are the ILS+SD and the ILS+VND. The distributions of the final error are vary similar, with a median slightly over *0.4*, and with *11* and *12* working solutions respectively. This results confirm the deep effectiveness of the search diversification which helps to escape from the local minima and explore different regions of the search landscape. In the former, i.e. ILS+SD, the diversification mechanism is exploited by the ILS method while in the latter, that is ILS+VND, also the internal local search procedure alternates intensification, during the *5000* possible non-improving steps, to diversification, when the neighborhood relation changes in favor of a more perturbative one. In order to sta-

tistically test the differences in the error distributions we execute the wilcoxon test for each couple of algorithms. The results are reported in Table 6.5.

Algorithms		**Test Result**
Algorithm 1	Algorithm 2	p-value
SD	GA	0.01004
SD	ILS+SD	1.427e-07
SD	ILS+VND	7.562e-08
ILS+SD	ILS+VND	0.5637
ILS+SD	GA	0.0002003
ILS+VND	GA	5.371e-05

Table 6.5: *Wilcoxon test results between algorithms*

These results confirm the observations: a significative difference stands between the SD and the other three methods. The two methods which make use of the ILS have significantly better performance also with respect to the GA. Then, between the two techniques, there is no significative statistical difference, even if the ILS+VND is confirmed to be slightly better. To further corroborate these information, we present in the following the histogram (Figure 6.4) which illustrates the number of results, for each algorithm, in the range between 0 and 0.5. The plot provides evidence to the fact that the two algorithm with the ILS are able to generally reduce the error, leading almost all the runs to a final error under the 0.5. The GA and the SD are instead rather worse, succeeding in such exploit in less than the half of runs for the former and in just 14 runs for the latter.

Up to now, we have compared the algorithm results after their entire execution. This is not completely fair, since the algorithms have different experimental settings and they could execute different number of iterations. Moreover, the iterations can have a different nature: for example an iterations of the GA is completely different with respect to the iterations of the other three methods, since each GA iteration handle a population of candidate solutions rather than a single solution. For these reasons, to fairly compare the algorithm behavior another method has to be employed. The method is the RTDs, already used in Section 5.5, which shows the probability distribution that an algorithm finds a solution of a given quality within a certain

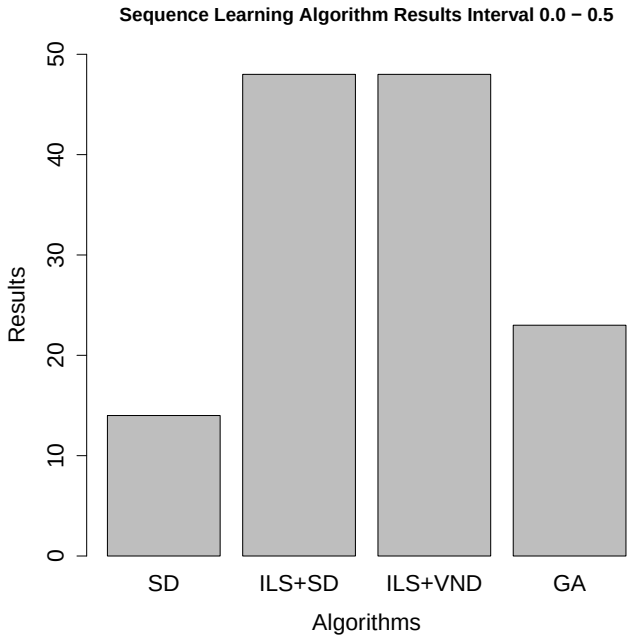


Figure 6.4: *Final results in range 0 - 0.5 for each algorithm*

time, or a certain number of iterations. Through this method we can compare the algorithm behavior in the same period of time with respect to more error thresholds. We decide to carry out our comparison over a frame of *100000 iterations*, choosing *5* different *error thresholds*. For the GA comparison, we consider an iteration each individual evaluated during a generation. This means that, with a population of 20 individuals, each generation is considered as 20 iterations. The RTD comparison is shown in Figure 6.5.

The y axis is scaled is in the range between 0 and 0.4 since the target task complexity, as mentioned, has led to a deterioration of performance. What the plots show is that, over the 100000 initial iterations the worst behaved algorithm is the GA, since its distributions are always under the distribution of the other algorithms. The simple SD is rather better, but again the two techniques exploiting the ILS show the best behavior also during the initial 100000 iterations. Between ILS+SD and ILS+VND there are no significative differences, since the latter has a slight advantage concerning the lowest thresholds while the situation is reversed for the highest error thresholds. In order to clearly show the differences between the algorithm behaviors we report another series of plots (Figure 6.6), each one corresponding

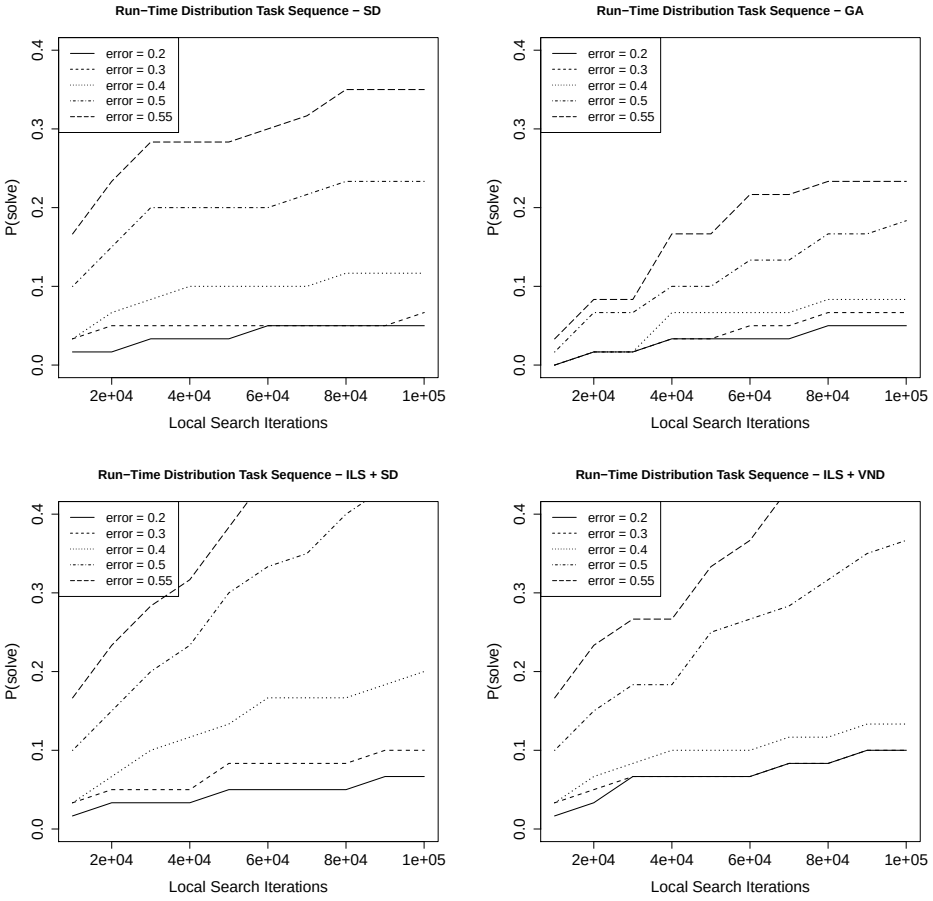


Figure 6.5: *Algorithm comparison through RTDs. 100000 iterations, 5 error thresholds. SD (top-left), GA (top-right) ILS+SD (bottom-left) ILS+VND (bottom-right)*

to a different error threshold with a line representing the trend of each algorithm.

These results confirm the effectiveness of the perturbation mechanism exploited by the ILS method, and in general of the diversification of the search alternated with a deep intensification. In fact, from the plots emerges that the difference between the simple SD and the ILS methods arises mostly after the 50000 iterations, that is the point at which, roughly, the first perturbation is performed. This can be a starting point for future improvements aimed at exploiting more effective diversification methods. Moreover, other mechanism can be employed to improve also the intensification phase, for instance with a tabu search that can help the search to do not get stuck inside cycles.

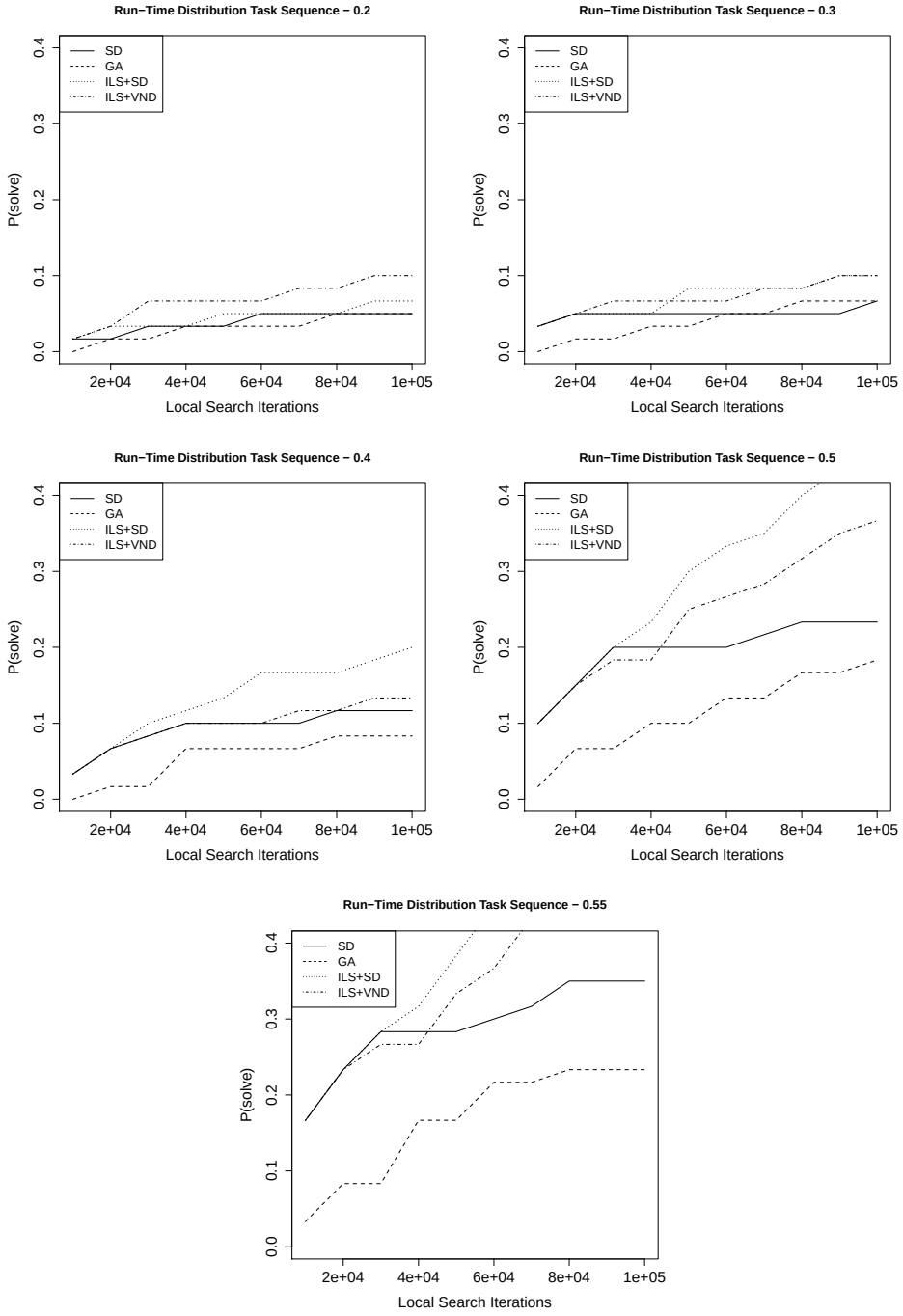


Figure 6.6: *Algorithm comparison through RTDs grouped on the basis of the error threshold. 100000 iterations, 5 error thresholds: 0.2 (top-left), 0.3 (top-right) 0.4 (bottom-left) 0.5 (bottom-right) 0.55 (center)*

A last statistic which can be useful is the time of execution of each algorithm. In fact, the result distributions shown in Figure 6.2 stem from algorithms with different number of iterations, and in some cases, like between the trajectory-based algorithms and the GA, the iterations have heterogeneous execution times. For these reason we show in Table 6.6 the time stats concerning all the algorithms employed, distinguishing between the execution over 100000 iterations, used for the RTDs comparison, and the entire run.

Algorithm	**milliseconds**		
	1 Iteration	**100000 iterations**	**Entire run**
SD	700	7e07	7e07
ILS+SD	700	7e07	3.5e08
ILS+VND	700	7e07	3.15e08
GA	550	5.5e07	2.2e08

Table 6.6: *Time stats comparison*

The Table shows that the GA iterations are in general faster, probably thanks to the internal optimization of the open-source library used for its implementation [47]. Moreover, since the single iteration time is the same for the other three methods, the interesting value is the last, concerning the total run execution. Apart from the simple SD executed for just 100000 iterations, there is a difference between the ILS+SD and the ILS+VND in terms of mean number of iterations executed and, consequently, in terms of total execution time. This is due to the experimental settings, in particular to the non-improving counter setting. In the former method, in fact, the non-improving timeout is probably excessively high and this leads the algorithm to reach the timeout of 50000 in each local search procedure. In the latter, instead, the changing neighborhood mechanism leads, in general, to a earlier termination and consequently a lower number of total iterations executed.

6.5.2 Analysis

After the algorithm result comparison we want to focus on the properties of the results in order to confirm the observations carried out for the test cases. More precisely, we want to analyze the trend of the

two quantities already considered in the test cases, during the training process. Before the analysis we need to make some preliminary remarks: the complexity of the target task have huge effects on the data collection. In fact, the high number of iterations needed to reach good results, and the limited number of final networks able to perform the task, significantly limit the data to be analyzed. For these reasons, in the following we analyze some typical representative cases, trying to draw considerations and conjectures.

The first quantity we analyze is the number of visited states by the network in their state space during the training process. Like for the test cases, our hypothesis is that there exist a connection between the local search improvements and the trend of the number of states visited by the networks inside their state space during their execution. In order to investigate this aspect we report two plots (Figure 6.7) about two different working network training, in which the objective function trend is overlapped to the trend of the umber of visited state.

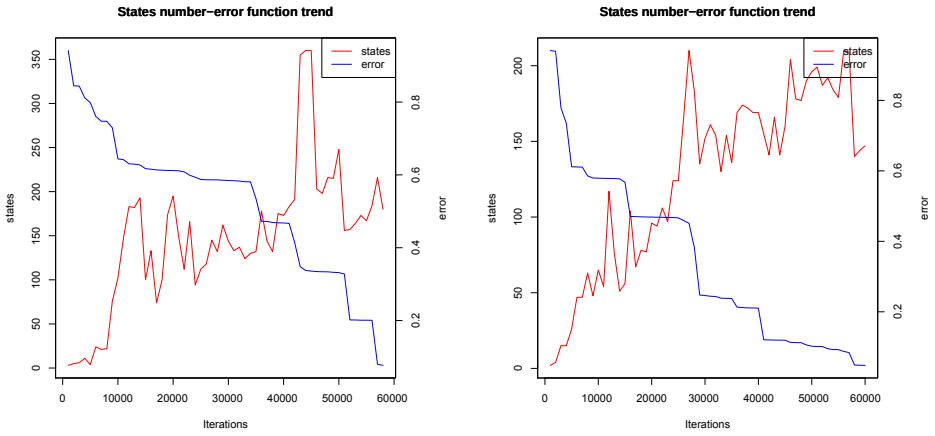


Figure 6.7: *Overlapped trends of visited states and objective function values during task sequence training of 60000 iterations. Two different networks*

The execution is limited to 60000 iteration in order to limit the memory resource needed to store the entire trajectories of the network during the training. The plots are very interesting because they show that the learning process seems to be incremental, i.e. the robots learn to recognize a color after the other and finally to generalize to a cyclic sequence. Moreover, the moments in which the objective function value significantly decrease its value, correspond to peaks in the number of state trend. Again we can speculate that there ex-

ists a parallel between the search space and the state space and the search process needs to explore in regions of the landscape in which the networks have a high number of visited states in order to obtain an improvement (*exploration* phase). Subsequently the search optimizes this results by wrapping and grouping the states (*exploitation* phase). This second phase affects the objective function value with small changes, probably due to the optimization and generalization which stem from the described grouping. This trend is repeated for all the improvements, but the most significative for our task is the improvement that leads the objective function under the threshold of 0.4. The reason is that the networks with an evaluation greater than 0.4 are able to recognize at most a single occurrence of the sequence, while under the 0.4 they start to recognize the cyclic sequence. Thus, we can see this improvement as the moment in which the process fully generalize the task. Interestingly, in both the two plots, this moment corresponds to the highest peak of the number of visited states. This is another attractive feature that in the test cases could not be such evident, due to the simplicity of the target tasks. Such feature supports our hypothesis that the design process needs to explore landscape areas where the solution are richer, in terms of dynamic properties, to find a solution which can perform a given task. Moreover it seems that the more the task is complex, the richer must be the dynamics in order to find a solution, which will be subsequently grouped and optimized again.

Last thing we want to highlight about the plots is that, in general, the number of states visited during the whole process is rather limited, slightly greater then for the simpler test case tasks. This is another important property of our methodology, which seems able to economize the resources and the state space. For a detailed description of the final internal organization of the networks and their dynamical properties that give raise to features, like the memory needed to solve this complex task, we refer to the work [29].

The second quantity we want to put in relation with the learning trend is the network dynamic complexity. The definition of statistical complexity is the same given in Section 5.5. Like for the test cases we compute the cross-correlation between complexity values and the objective function values, focusing on the topical moment, i.e. on the most relevant improvements. To this end, we sample the values of the two trends at each improvement at least as significant as the tenth, sorting by relevance, of the run. Then, we focus on the value of cross-correlation corresponding to $Lag = 0$, since we are interested in the

simultaneous changes of the two trends. The values obtained for the two networks under examination are reported in Table 6.7.

Network	**Cross-correlation at Lag = 0**
Network 1	-0.8808433
Network 2	-0.7077384

Table 6.7: *Cross-correlation between complexity and objective function at Lag 0 for the two networks analyzed*

Two values are clearly not enough to provide evidence, but can be stated that they support our first analysis: in general, there exists an anti-correlation between the two quantities, which means that relevant reductions of the objective function correspond to a complexity increase. As discussed for the test cases, after a deeper analysis of such data in order to verify that these results statistically stand, these information could be used to improve the methodology. The search method, for instance, could be designed so as to favor solutions with higher complexity in order to lead the process to quicker improvements.

Conclusion

In this thesis we employed an automatic design methodology in order to synthesize BN-based programs for robots able to perform a given task. Treating the design problem as a search problem, the methodology is composed of two main components: the robot program model and the metaheuristic technique used for the design process. Concerning the robot program model we have used the Boolean networks, whose dynamics is suitable to realize complex behaviors notwithstanding the simplicity of the model. Moreover the dynamical behavior can be studied and analyzed. For the second component we have utilized the stochastic local search methods.

The design process has been treated as an optimization combinatorial problem. More precisely, the optimization algorithm works on the Boolean functions of nodes, considered as the set of decision variables, in order to find the assignment which maximizes the robot performance.

Starting from these definitions, we have employed the methodology on two test cases whose target tasks are very basic robotic task: phototaxis and obstacle avoidance. These experiments have aimed at revealing some prominent and attractive aspects of the learning process thanks to the huge amount of data collected. In particular, we have proved the robustness of the methodology in tackling these experiments showing, for example, the excellent results obtained with the simplest stochastic technique and the ability of generalizing the task. Moreover, we have studied some properties of the search landscape, such as the landscape autocorrelation, that can affect the result quality. In this context, we have shown that the dynamical properties of the initial solution, i.e. the initial RBN, can impact the performance of the search process. Initial solutions which lead to more uncorrelated landscapes, i.e. networks in chaotic regime, cause a deterioration of the search performance. Furthermore, we have proved how the choice of the number of nodes of the BNs can deeply affect the performance of the process: it is crucial to find the right tradeoff between the re-

quired computational capacity for the target task and a small search landscape which can help the search algorithm.

An attractive study carried out concerns the link between the improvements during the training and the properties of the networks: we have shown that relevant improvements in the search landscape correspond to particular effects on the dynamical features of the networks, especially on the number of states visited. Precisely, the search method needs to move towards networks with rich dynamics in order to find a good solution and subsequently, with slight improvements, it optimizes this solution by grouping the states and generalizing the task. Furthermore, we have provided evidence that there exists a significative anti-correlation between the trend of the relevant objective function improvements and the complexity of the network dynamical behaviors. These studies show that, even if the search algorithm and the network dynamics operate in two different spaces not directly bound, during the learning process a relation is, in some respect, established.

Finally, the methodology has been employed for the dynamic task learning task, more complex due to the form of memory required to be performed. This last experiment has aimed at confirming in a complex scenario the interesting results obtained in the test cases and proposing important improvements for the methodology. After demonstrating the lack of the simple stochastic descent in tackling the task, we have proposed different techniques with improved features of search diversification, such as the iterated local search. Then, we have also refined the search intensification phase by utilizing techniques of variable neighborhoods. Finally, from the results comparison and the depth analysis, we have drawn important consideration about the best mechanisms that can improve the methodology and help to achieve a good tradeoff between intensification and diversification.

The results achieved during the work leave open several possible future developments. The first will be the employment of the methodology for complex target tasks composed by single subtasks, assessing how this affect the process. For instance, the last experiment undertook, i.e. the dynamic task learning, can be enhanced with obstacle avoidance issues or random walking in a wider arena. Then, studying in depth the correlation between the search process and the dynamic features of the networks, the methodology can be improved by exploiting this property: an idea can be the insertion of a new term in the objective function the leads the search towards solutions with certain dynamical characteristics (e.g. a higher number of visited states in

order to achieve a solution which can perform the target task). Moreover, other parameters cold be subject to automatic design, such as the number of nodes and the topology of the networks. The former aspect, especially, has been proved to have a strong effect on the design performance. Other studies should reveal the relation between the task complexity and the number of nodes required in order to solve it.

Another future work will be the employment of other stochastic local search methods in order to improve the methodology performance. This goal can be reached by optimizing the search behavior both concerning the intensification and the diversification. An idea could be, for instance, to avoid loops in the search movements by exploiting techniques which utilize information about the search history (e.g. tabu search).

Finally, an attractive idea could be to divide the automatic design process in the two stages: in the first, an algorithm (e.g. ant colony optimization) can set up the optimal topology and during the second another method can optimizes the Boolean function assignments in order to solve the given task.

Bibliography

[1] Kauffman, S. *Metabolic stability and epigenesis in randomly constructed genetic nets* Journal of Theoretical Biology 22, 437467 (1969)

[2] C. Gershenson. *Introduction to random boolean networks* CoRR,nlin.AO/0408006, 2004.

[3] S. Benedettini et al. *Learning Boolean Networks* Elsevier Editorial System(tm) for Neurocomputing

[4] B. Mesot, C. Teuscher. *Deducing local rules for solving global tasks with random Boolean networks* Physica D 211 (2005) 88106.

[5] S. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution* Oxford University Press, UK, 1993.

[6] Aldana, M., Balleza, E., Kauffman, S., Resendiz, O. *Robustness and evolvability in genetic regulatory networks* Journal of Theoretical Biology 245 (2007) 433-448.

[7] Ribeiro, A., Kauffman, S., Lloyd-Price, J., Samuelsson, B., Socolar, J. *Mutual information in random Boolean models of regulatory networks* Physical Review E 77(011901) (2008).

[8] Nykter, M., Price, N., Aldana, M., Ramsey, S., Kauffman, S., Hood, L., Yli-Harja, O., Shmulevich, I. *Gene expression dynamics in the macrophage exhibit criticality.* Proceedings of the National Academy of Sciences, USA. Volume 105. (2008) 1897-1900

[9] Kauffman, S. A. *Investigations* Oxford University Press (2000).

[10] C.G. Langton. *Computation at the edge of chaos: phase transitions and emergent computation.* Phys. D, 42(1-3):12-37, 1990.

[11] B. Derrida and G. Weisbuch. *Evolution of overlaps between configurations in random Boolean networks.* Journal de physique, 47(8):1297-1303, 1986.

[12] Derrida, B. and Pomeau, Y. *Random networks of automata: A simple annealed approximation* Europhys. Lett., 1(2):45-49. (1986).

[13] S.A. Kauffman and R.G. Smith. *Adaptive automata based on Darwinian selection.* Physica D: Nonlinear Phenomena, 22(1-3):68-82, 1986.

[14] Roli, A., Arcaroli, C., Lazzarini, M., Benedettini. *Boolean networks design by genetic algorithms.* In: Villani, M., Cagnoni, S. (eds.) Proceedings of CEEI 2009 - Workshop on complexity, evolution and emergent intelligence. Reggio Emilia, Italy (2009), http://www.aixia09.unimore.it/index.php/workshops/64

[15] Stefano Benedettini et al. *Learning Boolean Networks*

[16] Andrea Roli et al. *Boolean network robotics: a proof of concept*

[17] Mattia Manfroni. *Towards Boolean network design for robotics applications*

[18] A. M. Turing. *On computable numbers, with an application to the entscheidungsproblem* Proceedings of the London Mathematical Society, 42(2), 230-265 1936

[19] A. Newell and H. A. Simon. *The logic theory machine: A complex information processing system* IRE Transactions of information theory, 1956.

[20] A. Newell and H. A. Simon. *Computer science as empirical enquiry: Symbols and search* Communications of the ACM, 1976.

[21] R. A. Brooks. *A robust layered control system for a mobile robot.* IEEE Journal of Robotics and Automation, 1986.

[22] H. Maturana and F. J. Varela. *Autopoiesis and Cognition: The Realization of the Living. volume 42 of Boston Studies in the Philosophy of Science.* D. Reidel Publishing Company, Dordrecht, The Netherlands, 1980.

[23] T. Quick et al. *Evolving Embodied Genetic Regulatory Network-Driven Control Systems.* Department of Computer Science University College London Gower Street, London WC1E 6BT, U.K.

[24] Ansaloni, L., Villani, M., Serra. *Dynamical critical systems for information processing: a preliminary study* Villani, M., Cagnoni, S. (eds.) proceedings of CEEI 2009 - Workshop on complex- ity, evolution and emergent intelligence. Reggio Emilia, Italy (2009)

[25] Miller JF, Job D, Vassilev VK. *Principles in the evolutionary design of digital circuits part I. Genetic Programming and Evolvable Machines* (2000) 1: 735.

[26] A. M. Turing. *Computing machinery and intelligence* Mind 49: 433460 (1950)

[27] D. Floreano, L. Keller. *Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection*

[28] S. Nolfi, D. Floreano, O. Miglino, F. Mondada. *how to evolve autonomous robots: different approaches in evolutionary robotics*

[29] M. Amaducci *Design of Boolean network robots for dynamics tasks*

[30] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. C. Zuerey, D. Floreano, and A. Martinoli. *The epuck, a Robot Designed for Education in Engineering.* In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, volume 1, pages 59 - 65, Portugal, 2009. IPCB: Instituto Politecnico de Castelo Branco

[31] C. Pinciroli et al. *ARGoS: a Modular, Multi-Engine Simulator for Heterogeneous Swarm Robotics*

[32] H.H. Hoos and T. Stutzle. *Stochastic local search: Foundations and applications* Morgan Kaufmann, 2005.

[33] P. Hansen and N. Mladenovi. *An introduction to variable neighborhood search.* In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, pages 433458. Kluwer Academic Publishers, Boston, MA, USA, 1999.

[34] M. Chiarandini, T. Stutzle. *An application of iterated local search to graph coloring problem* D. S. Johnson, A.Mehrotra, M. Trick (Eds.), Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, pp. 112125.

[35] H. Lourenco, O. Martin, T. Stutzle. *Iterated local search* F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, volume 57 of International Series in Operations Research Management Science, Springer, New York, NY, 2003, pp.

[36] M. Dorigo, V. Maniezzo, A. Colorni. *Distributed Optimization by Ant Colonies* actes de la premire confrence europenne sur la vie artificielle, Paris, France, Elsevier Publishing, 134-142, 1991.

[37] M. Dorigo and T. Stutzle. *Ant Colony Optimization* MIT Press, Cambridge, MA, USA, 2004.

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

[39] Goldberg, D. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, Reading, MA (1989).

[40] Ricardo Lopez-Ruiz, Hector Mancini, Xavier Calbet. *A Statistical Measure of Complexity.*

[41] Shannon, C.E., Weaver, W. *The Mathematical Theory of Communication.* University of Illinois Press, Urbana, Illinois (1949).

[42] *http://en.wikipedia.org/wiki/Cross-correlation.*

[43] O. Miglino H. H. Lund S. Nolfi. *Evolving Mobile Robots in Simulated and Real Environments.*

[44] *http://aass.oru.se/Agora/FLAR/HFC/home.html*

[45] Craig Reynolds. *Steering Behaviors For Autonomous Characters, Boids Web Page, Flocks, Herds, and Schools: A Distributed Behavioral Model.* 1987 Siggraph Paper

[46] Ron Sun. *Introduction to sequence learning* CECS Departement, University of Missouri Columbia, MO 65211, USA

[47] Martin Kreutz, Bernhard Sendhoff and Christian Igel. *EALib: A C++ class library for evolutionary algorithms* Institut fur Neuroinformatik, Ruhr-Universitat Bochum