Neurocomputing 99 (2013) 111–123

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Dynamical regimes and learning properties of evolved Boolean networks

Stefano Benedettini [a,*], Marco Villani [b,c], Andrea Roli [a], Roberto Serra [b,c], Mattia Manfroni [a], Antonio Gagliardi [b], Carlo Pinciroli [d], Mauro Birattari [d,1]

[a] DEIS Alma Mater Studiorum Università di Bologna Campus of Cesena, Via Venezia 52, I-47521 Cesena, Italy
[b] Faculty of Mathematical, Physical and Natural Sciences, Università di Modena e Reggio Emilia, Viale A. Allegri 9, 42121 Reggio Emilia, Italy
[c] European Centre for Living Technology, Ca' Minich, S. Marco 2940, 30124 Venezia, Italy
[d] IRIDIA, Université libre de Bruxelles, 50, Av. F. Roosevelt, CP 194/6 B-1050 Brussels, Belgium

ARTICLE INFO

Article history:
Received 7 October 2011
Received in revised form
14 February 2012
Accepted 3 May 2012
Communicated by B. Apolloni
Available online 3 July 2012

Keywords:
Boolean networks
Machine learning
Metaheuristics
Density classification problem
State-controlled Boolean network

ABSTRACT

Boolean networks (BNs) have been mainly considered as genetic regulatory network models and are the subject of notable works in complex systems biology literature. Nevertheless, in spite of their similarities with neural networks, their potential as learning systems has not yet been fully investigated and exploited. In this work, we show that by employing metaheuristic methods we can train BNs to deal with two notable tasks, namely, the problem of controlling the BN's trajectory to match a set of requirements and the density classification problem. These tasks represent two important categories of problems in machine learning. The former is an example of the problems in which a dynamical system has to be designed such that its dynamics satisfies given requirements. The latter one is a representative task in classification. We also analyse the performance of the optimisation techniques as a function of the characteristics of the networks and the objective function and we show that the learning process could influence and be influenced by the BNs' dynamical condition.

1. Introduction

Models of neural networks can be roughly divided into two classes, i.e., those where there is a directional flow of activation, such as feed-forward layered networks [1], and those which are truly dynamical systems, like, for example, those proposed by Elman [2] and by Hopfield [3].

This is particularly clear in the case of the Boolean Hopfield model, whose attractors are fixed points (in the usual case with symmetric synaptic weights). Another well-known Boolean network model is that of Random Boolean Networks (briefly, RBNs), which display a much richer dynamics than that of the symmetric Hopfield model. The attractors of finite RBNs are cycles, so the dynamics are fairly trivial; however, it has been possible to introduce a notion of ordered vs. disordered attractors, which represents the analogue (in a finite discrete system) of the distinction between regular and chaotic attractors in continuous systems.

Interestingly, this distinction holds for many properties usually associated to continuous chaotic systems, as, for example, the stability of dynamical attractors with respect to small perturbations: in the case of ordered systems, small perturbations usually die out, while in disordered ones they tend to grow. In RBNs it has been observed that ordered systems usually have fairly regular basins on attraction, so that two nearby states often evolve to the same attractor, while in disordered systems they often go to different attractors. This behaviour is reminiscent of the "butterfly effect" and this provides a reason why disordered RBNs are often called "chaotic" (in spite of the fact that, since the attractors are cycles, the term "pseudo-chaotic" would be more appropriate). The reason for this choice of terms is the following: a deterministic discrete system composed by a finite number of nodes N, each node taking one of M possible values, owns a finite number of different states, and, evidently, sooner or later reaches an already visited state: from that moment on the system starts to repeat the same sequence of states. Nevertheless, the period of a cycle can range from 1 to M^N, and for large systems the maximum value is so high that such a cycle could be covered only in a period of time greater than the age of the universe. For any purposes, a system owning cycles that long is called "pseudo-chaotic", or simply "chaotic" [4].

It turns out that the value of the so-called Derrida parameter ξ [4], which is the discrete analogue of the Lyapunov exponent of continuous dynamical systems, determines whether a given

* Corresponding author. Tel.: +39 0547 339210; fax: +39 0547 339208.
E-mail addresses: s.benedettini@unibo.it (S. Benedettini), marco.villani@unimore.it (M. Villani), andrea.roli@unibo.it (A. Roli), rserra@unimore.it (R. Serra), manfroni2@alice.it (M. Manfroni), cpinciro@ulb.ac.be (C. Pinciroli), mbiro@ulb.ac.be (M. Birattari).

[1] Mauro Birattari acknowledges support from the F.R.S.-FNRS of Belgium's Wallonia-Brussels Federation.

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.05.023

family of RBNs tends to display ordered or chaotic behaviours—$\xi < 1$ corresponding to ordered networks and $\xi > 1$ to chaotic ones. Particular interest has been raised by those networks which are in a critical state with ξ equal to (or close to) 1 , i.e., an intermediate state between order and chaos.

It has been proposed in the past that biological systems should operate in critical states (or close to the boundary between ordered and disordered regions, slightly into the ordered region), on the basis of heuristic arguments which can be summarised as follows. Biological systems need a certain level of stability, in order not to be disrupted by fluctuations which can take place either in the system or in the environment, and they need at the same time to provide flexible responses to changes in the environment. While a chaotic system would be poor at satisfying the first need, a system deeply in the ordered region would fail to meet the second requirement. A critical system should allow for an optimal trade-off between the two, therefore natural evolution should drive biological systems towards critical states [4].

These very same reasons should hold as well for an artificial learning system. A question may arise as to what are the conditions for critical networks to outperform ordered and chaotic ones. A sound, theory-based approach to the design of effective learning networks could try to answer this question. Nevertheless, such a theory is still missing. In this work, we make a first step towards it: we investigate whether and under what conditions the network's dynamical regime influences the learning performance, as long as static environments (i.e., not changing in time) are concerned. Future work will be aimed at investigating the case of changing environments.

Another reason of interest on learning Boolean networks comes from progress in optimisation methods. Note that there have been some attempts in the past to devise learning algorithms for RBNs, which have met limited success [5,6]. However, recent advances in the development of effective metaheuristics offer new tools to tackle the problem of devising RBNs which are able to perform well in difficult tasks through learning by examples. In this paper, we propose a principled approach for training Boolean networks and we show effective performance in some selected tasks.

This work is structured as follows: Section 2 provides a brief summary of the main concepts related to Boolean networks; Section 3 illustrates the optimisation method we apply to train our networks, whereas Sections 4 and 5 describe the two applications we focus on, namely, the problem of controlling the networks' trajectory to reach a target and the Density Classification Problem. In Section 6 we draw our conclusions and indicate some promising directions for future work.

2. Boolean networks

Boolean networks (BNs) have been introduced by Kauffman [7,4] as a genetic regulatory network model. BNs have been proven to reproduce very important phenomena in genetics and they have also received considerable attention in the research communities on complex systems [8,4]. A BN is a discrete-state and discrete-time dynamical system whose structure is defined by a directed graph of N nodes, each associated to a Boolean variable x_i, $i=1, \ldots, N$, and a Boolean function $f_i(x_{i_1}, \ldots, x_{i_{K_i}})$, where K_i is the number of inputs of node i. The arguments of the Boolean function f_i are the values of the nodes whose outgoing arcs are connected to node i (see Fig. 1a). The state of the system at time t, $t \in \mathbb{N}$, is defined by the array of the N Boolean variable values at time t: $s(t) \equiv (x_1(t), \ldots, x_N(t))$. The most studied BN models are characterised by *synchronous* dynamics—i.e., nodes update their states at the same instant—and *deterministic* functions (see Fig. 1(b)). However, many variants exist, including asynchronous and probabilistic update rules [9]. BN models' dynamics can be studied by means of classic dynamical system methods [10,11], hence the usage of concepts such as state (or phase) space, trajectories, attractors and basins of attraction. BNs can exhibit complex dynamics and some special ensembles have been deeply investigated, such as that of random BNs. Recent advances in this research field, along with efficient mathematical and experimental methods and tools for analysing BN dynamics, can be mainly found in works addressing issues in genetic regulatory networks or investigating properties of BN models [12–15]. A special category of BNs that has received particular attention is that of RBNs, which can capture relevant phenomena in genetic and cellular mechanisms and complex systems in general. RBNs are usually generated by choosing at random K inputs per node and by defining the Boolean functions by assigning to each entry of the truth tables a 1 with probability p and a 0 with probability $1-p$. Parameter p is called *bias*. Depending on the values of K and p the dynamics of RBNs is called either *ordered* or *chaotic*. In the first case, the majority of nodes in the attractor is frozen and any moderate-size perturbation is rapidly dampened and the network returns to its original attractor. Conversely, in chaotic dynamics, attractor cycles are very long and the system is extremely sensitive to small perturbations: slightly different initial states lead to divergent trajectories in the state space. RBNs temporal evolution undergo a second order phase transition between order and chaos, governed by the following relation between K and p: $K_c = [2p_c(1-p_c)]^{-1}$, where the subscript c denotes the critical values [16]. Networks along the critical line show equilibrium between robustness and adaptiveness [12]; for this reason they are supposed to be plausible models of the living systems organisation. Recent results support the view that biological genetic regulatory networks operate close to the critical region [15,17,18].

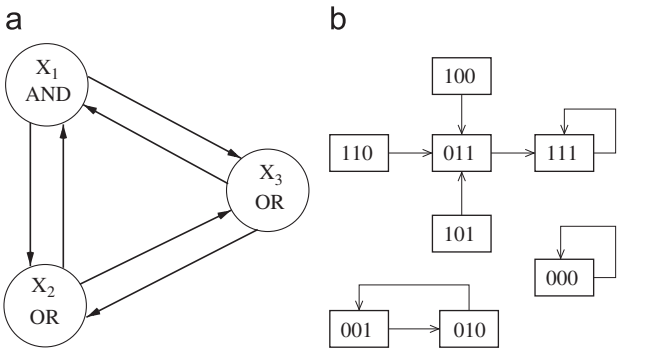

Fig. 1. An example of a BN with three nodes (a) and its corresponding state space under synchronous and deterministic update (b). The network has three attractors: two fixed points, (0,0,0) and (1,1,1), and a cycle of period 2, $\langle (0,0,1),(0,1,0) \rangle$.

3. Training Boolean networks by metaheuristics

BNs have been mainly considered as genetic regulatory network models, enabling researchers to achieve prominent results in the field of complex systems biology [19,20,12,9]. Nevertheless, in spite of their similarities with neural networks, their potential as learning systems has not yet been fully investigated and exploited. In this section, we first summarise the works in the literature that concern training or automatic designing BNs (Section 3.1) and then, in Section 3.2, we illustrate our method to tackle two notable applications: the problem of controlling the BN's trajectory to reach a target state (Section 4) and the density classification problem (Section 5).

We wish to remark that Section 3.2 contains basic definitions and concepts from metaheuristics and learning theory; the novice to these fields is encouraged to carefully read this section, whilst the expert is advised to at least skim through it to familiarise with the terminology we use throughout the paper.

3.1. Related work

The first work concerning BNs as learning systems has been presented by Patarnello and Carnevali [5] who trained a feed-forward Boolean network to perform binary additions. The training algorithm used was Simulated Annealing [21]. A study on the automatic design of BNs appeared the same year and was proposed by Kauffman [22]. The goal of the work was to generate networks whose attractors matched a prescribed target state. The algorithm proposed is a sort of genetic algorithm with only a mutation operator (no crossover) that could either randomly rewire a connection or flip a bit in a function's truth table, and extreme selection pressure: once a fitter individual was found it would replace the whole population. The process is analogous to a stochastic ascent local search. Lemke et al. extend this scenario [23] in that they require a network to match a target cycle. In this work a full-fledged genetic algorithm (with crossover) is employed. Another evolutionary approach is adopted by Esmaeili and Jacob in [24], where they require a population of RBNs to maximise a fitness function defined by a combination of several feature like network sensitivity, number of attractors and basin entropy. In their algorithm, a network can undergo changes in both functions and topology. Notably, mutation operator was allowed to add or delete a node. Their study is limited to networks of small size ($N \leq 10$). Several works addressing evolution of robust BNs have been proposed by Drossel and others. In these works, robustness is intended as the capacity of a network to return to the same attractor after a random flip occurs in one of its nodes. Various search strategies have been employed and will be outlined in the following. In [25] the authors applied a stochastic ascent (called "adaptive walk" in the paper) to networks with canalising functions; the move operator could rewire a connection or replace a function with a canalising one. The next three contributions revisited and extended this last paper, with the same goals of finding robust networks. Mihaljev [26], instead of a local search, proposes a genetic algorithm whose mutation operator is the move procedure described above. Fretter [27] studies the dynamical properties of evolved networks with *any* functions, not only canalising ones. Szejika [28] extends her previous work and this time investigates the behaviour of evolved networks with Boolean threshold functions. Another genetic algorithm was proposed by Roli et al. to design network with prescribed attractor length [29,56]. In a work by Espinosa-Soto and Wagner [30], populations of random Boolean threshold networks (a special case of RBNs) are evolved, by means of a genetic algorithm, to investigate the relationship between modularity and evolvability of genetic regulatory networks. The actual algorithm utilised is a simple genetic algorithm with constant size population, no crossover and a mutation operator capable of modifying edge weights in the topology graph. In a more recent work, Benedettini et al. [31] employed an Iterated Local Search metaheuristic to automatically design networks with maximally dissimilar attractors. Finally, we mention a work in which probabilistic BNs are trained so as to learn the equality function [6].

In summary, the techniques proposed in the literature to train a BN belong to either of two families: local search (stochastic ascent and variants, Simulated Annealing, etc.) and genetic algorithms (with variants in the genetic operators). The methods used are quite simple and might not be effective in tackling hard learning tasks. Indeed, the goals addressed in the literature are mainly concerned with investigating phenomena in evolutionary biology, rather than tasks in machine intelligence. We believe that the recent advances in engineering stochastic local search can be fruitfully applied also in the task of training BNs. In this paper, we show that advanced local search strategies, as well as algorithm engineering and analysis, enable us to train BNs for accomplishing difficult learning tasks.

In the next section we present our training methodology which, by means of an optimisation algorithm, modifies the structure of a RBN. For this reason, in the following we refer to the objects manipulated by the optimisation algorithm with the more general acronym "BNs", remarking in such a way that they are no longer RBNs.

3.2. Methods

In this work, we adopt the supervised learning paradigm. We suppose that it is possible to define an objective function that evaluates the performance of a network with respect to a given task to be accomplished. Besides the objective function, the learning process requires that some parameters of the system can be subjected to variations according to a learning algorithm. Since there are no dedicated algorithms for general BNs, we formulate the learning process into an optimisation problem. A prominent example of this approach is evolutionary robotics, in which evolutionary computation techniques are used for designing robots controlled by neural networks [32]. In this perspective, the learning process of a BN can be modelled as a combinatorial optimisation problem by properly defining the set of decision variables, constraints and the objective function. In this work, we employ metaheuristics to tackle this optimisation problem. Metaheuristics (a.k.a. stochastic local search techniques [33]) are general search strategies upon which a specific algorithm for solving an optimisation problem can be designed [34]. Examples of metaheuristics are Simulated Annealing, Tabu Search, Iterated Local Search, Ant Colony Optimisation and Genetic Algorithms. In our approach, which is illustrated in Fig. 2, the metaheuristic algorithm manipulates the decision variables which encode the Boolean functions of a BN. A complete assignment to those variables defines an instance of a BN. This network is then simulated and evaluated according to the specific target requirements. A specific software component is devoted to evaluate the BN and returns an objective function value to the metaheuristic algorithm that, in turn, proceeds with the search. Therefore, the evaluation provides the feedback used in the learning process.

Many successful metaheuristics are based on the iterative perturbation of a current candidate solution; notable instances of this scheme are Simulated Annealing and Tabu Search. The search process starts from an initial candidate solution and iteratively produces new candidates by (slightly) perturbing the current one, until some termination criterion is met (e.g., the

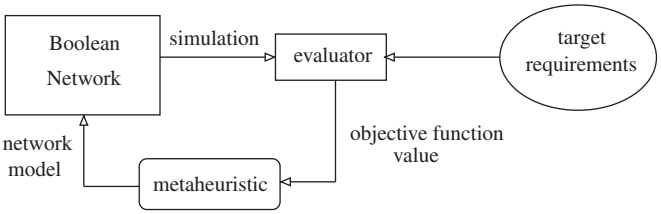

Fig. 2. Scheme of the process for training a BN. The BN is simulated and its behaviour is compared to the desired one defined by specific requirements. The evaluator component provides a feedback to the metaheuristic, which manipulates the BN parameters.

computation time limit is reached). Different search strategies can be defined by instantiating the two basic choices in the scheme, i.e., the generation and the choice of the next possible candidate solution and the acceptance criterion. These search strategies have been extended and improved, for example by adding advanced exploitation and exploration strategies [35].

Another prominent family of metaheuristics is that of genetic algorithms (GAs), which belong to the broad class of evolutionary computation techniques [32]. The general scheme of GAs is illustrated in Algorithm 1. The algorithm iteratively generates a new population of candidate solutions by applying operators such as selection, mutation and recombination to the current population. This search strategy actually performs a biased sampling of the search space; the parameters of the probabilistic model used for sampling are iteratively adapted in order to concentrate the search in promising areas of the search space.

Algorithm 1. Genetic algorithm high-level scheme.

1: $P \leftarrow$ GenerateInitialPopulation()
2: Evaluate(P)
3: **while** termination conditions not met **do**
4: $P' \leftarrow$ Recombine(P)
5: $P'' \leftarrow$ Mutate(P')
6: Evaluate(P'')
7: $P \leftarrow$ Select(P'',P)
8: **end while**

In the experiments that will be described in the following, a specific metaheuristic has been used, namely Iterated Local Search (ILS), which extends the basic perturbative search. ILS is a well-known algorithmic framework, illustrated in Algorithm 2, successfully applied to many hard combinatorial optimisation problems [36,37]. ILS makes it possible to combine the efficiency of local search with the capability of escaping from the basin of attraction of local optima. ILS applies an embedded stochastic local search method (line 6) to an initial solution until it finds a local optimum; then it perturbs the solution (line 5) and it restarts local search.

Algorithm 2. Iterated Local Search high-level framework.

1: INPUT: A LOCAL SEARCH
2: $s \leftarrow$ generateInitialSolution ()
3: $s^* \leftarrow$ localSearch (s)
4: **while** termination conditions not met **do**
5: $s' \leftarrow$ perturbation (s_{best})
6: $s'_{ls} \leftarrow$ localSearch (s')
7: $s^* \leftarrow$ acceptanceCriterion (s^*,s'_{ls})
8: **end while**
9: **return** s^*

In this work we implemented the following choices to instantiate the ILS framework.

Acceptance criterion: accept a new solution if it is better than the current best one.

Perturbation: for each node function a random flip in the truth table is performed. This choice makes ILS not too close to random restart, while keeping the perturbation computationally fast and easy to implement. As a drawback, local search moves can undo such perturbation, albeit unlikely.

The last component to be defined is the embedded local search procedure. We opted for Stochastic Descent (SD), a very basic search strategy, which, despite its simplicity, proved to be very effective. SD is a problem-independent local search algorithm whose pseudo-code is shown in Algorithm 3.

Algorithm 3. Stochastic descent.

1: INPUT: A SOLUTION s, AN OBJECTIVE FUNCTION F, a neighbour definition $\mathcal{N}$
2: $s_{best} \leftarrow s$
3: $v \leftarrow \mathcal{N}(s_{best})$
4: **repeat**
5: randomly pick a neighbour $s_0 \in N$ without replacement
6: **if** $F(s_0) < F(s_{best})$ **then**
7: $s_{best} \leftarrow s_0$
8: $v \leftarrow \mathcal{N}(s_{best})$
9: **else**
10: $v \leftarrow v \backslash s_0$
11: **end if**
12: **until** timeout **or** $v = \emptyset$
13: **return** s_{best}

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

Search state: a search state is a BN.

Initial solution: a random BN with defined N, K and bias p.

Neighbour definition: this component defines the modification, or *move*, performed on the current solution. For this experiment, first we randomly choose a node function, then we flip a bit in its truth table. The pair ⟨node, truth table position⟩ is uniformly sampled without replacement.

The objective function depends on the specific task to be accomplished, therefore it will be described separately for each case study presented. In the problems that will be discussed, the goal of the search is to minimise the objective function, which can thus be considered as an error function.

4. Target state-controlled Boolean networks

In this section, we describe the experiments in which we train a BN in such a way that some requirements on its trajectory are fulfilled. The problem of designing a dynamical system such that its trajectory in the state space satisfies specific constraints is a typical control problem. In the case of BNs, which in the general case exhibit complex dynamics, this task is not trivial for an automatic procedure because an assignment of Boolean functions must be found such that the resulting BN dynamics fulfils the requirements. This problem has been chosen with the aim of assessing the effectiveness of our approach. In fact, for a BN with N nodes and K inputs per node, the whole search space has a cardinality of 2^{KN}.

In this work, we are concerned with tasks in which a target state must be reached, subject to additional constraints. Given are an initial state s_0, a target state $\hat{s}$ and a number of network simulation steps T. The goal is to design a BN such that the trajectory in the state space with origin in s_0 reaches the target state $\hat{s}$ in one of the following conditions:

1. the target state $\hat{s}$ is reached in a number of steps less than or equal to T;
2. the target state $\hat{s}$ is reached for the first time at step t, such that $t \in [z,T]$, $0 < z < T$, where z is a parameter of the problem;
3. as point 2, but with the additional requirement that the target state is a fixed point.

4.1. Experimental setting

The BNs used for this task have $N=100$ nodes and $K=3$ distinct inputs per node (no self-connections). The connections of the networks are randomly generated. The initial Boolean functions

are also randomly generated with bias $p \in \{0.5, 0.788675, 0.85\}$. We recall that, for RBNs, the Lyapunov exponent λ can be analytically calculated as follows: $\lambda = \log \xi = \log [2p(1-p)K]$ [38, equation 27], where ξ is the Derrida parameter. Such bias values, if substituted in the previous equation, correspond to chaotic ($\lambda > 0$), critical ($\lambda = 0$) and ordered ($\lambda < 0$) regimes, respectively. For each value of p, 30 RBNs have been generated. The initial state s_0 of each BN is generated according to a uniform distribution over the whole state space. The target state $\hat{s}$ is likewise randomly generated.

The metaheuristic search used for designing the BN is ILS, which has been described in Section 3.2. Since the goal, in all the three cases, consists in reaching a given target state, the neighbourhood of the current solution can be sampled with a heuristic bias, trying to focus the search on promising neighbours of the current solution. To this aim, the node function to be changed is chosen among the ones corresponding to nodes whose values do not match the target state. The rationale behind this heuristic is that of "repair algorithms", in which local search moves only affect the parts of the current solution that contribute to increasing the objective function (to be minimised).

For each experiment, 100 000 iterations of the optimisation algorithm have been executed. Each iteration corresponds to a simulation of the respective BN trajectory lasting T steps, with $T = 1000$ (i.e., 1000 BN state updates).

4.1.1. Task 1: reaching a target state

The goal of this case study is to train a BN in such a way that its trajectory reaches a given a target state $\hat{s}$ at least once within the temporal left-open interval $]0,T]$. The evaluation of a BN is done on the basis of the evolution time step in which the BN presents the largest number of Boolean variables matching the target state. Let $u(t)$ be the function returning the number of Boolean variables matching the target state at each simulation step t, with t belonging to the left-open interval $]0,T]$; the objective function f_{task1}, to be minimised, can be described as follows:

$$f_{task1} = \min_{t \in]0,T]} \left(1 - \frac{u(t)}{N}\right).$$

To assess the robustness of the process, we compute the fraction of successful runs at each iteration of the algorithm, i.e., we estimate the *success probability at iteration t*, defined as the probability that a network with minimal objective function is found at generation $t' \leq t$. This kind of statistic is also known as *run length distribution* [33]. The results obtained are shown in Fig. 3. We first note that all the BNs with initial bias $p = 0.85$ reach the goal within 80 000 iterations of the optimisation algorithm. Also BNs initially in critical regime attain good performances, whereas only 10% of chaotic BNs reaches the goal. In order to explain this phenomenon, it is necessary to study the characteristics of the *search landscape*, as usually done in local search applications to combinatorial problems [33]. The search landscape is defined as a labelled graph whose vertices represent the search space states (in this case, the Boolean functions of the network), edges connect neighbouring states (in this case the topology of the graph is an hypercube, since we use a 1-Hamming neighbourhood) and vertex labels denote the objective function value associated to the search space states. It is commonly acknowledged that when the landscape is *smooth*, i.e., when neighbours have similar objective function values, local search is more effective than in the *rugged* case, in which the values across neighbouring states are characterised by high variance [33]. This property can be assessed by estimating the *autocorrelation* of the landscape. Smooth landscapes are characterised by high autocorrelation, while rugged ones have low autocorrelation [39]. The autocorrelation of a series $G = (g_1, \ldots, g_m)$ of objective

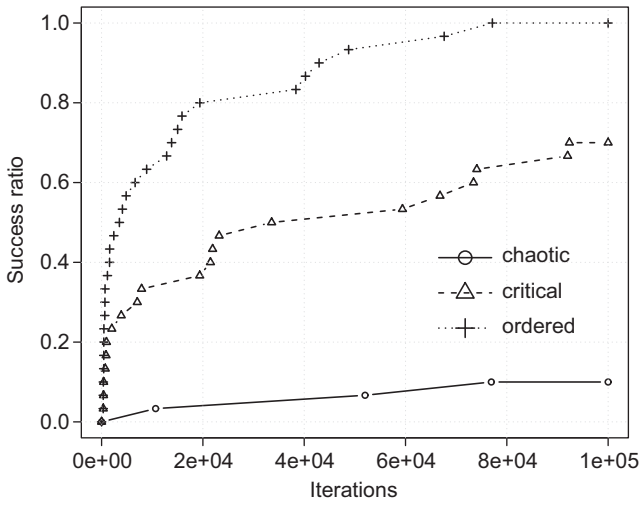

Fig. 3. Run length distribution related to task 1: reaching a target state.

function values is computed as

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

where $\overline{g}$ is the average value of the series. This definition refers to the autocorrelation of length one, i.e., that corresponding to series generated by sampling 1-Hamming neighbouring states in the search space. In optimisation problems, landscape autocorrelation obviously depends on the objective function and it is in general impossible to know in advance whether the neighbourhood structure combined with the objective function will produce a smooth, hence easy to be explored, landscape. However, in the case of BN design by metaheuristics, we can exploit a little piece of information concerning the type of tasks BNs are trained to perform. Indeed, the objective function is affected by BN dynamics, especially in those tasks in which it is a direct function of some properties of the BN trajectories in the state space. In particular, the dynamical characteristics of the initial BNs may have a strong impact on the effectiveness of local search, because they set the autocorrelation values of the landscape in which the search starts. BNs generated randomly with $p = 0.85$ are very likely to be ordered, therefore small changes in their Boolean functions correspond to small variations in the BN dynamics, hence small differences in the objective function values. Conversely, most of the BNs generated with $p = 0.5$ behave chaotically: slightly differing chaotic networks have a very different behaviour, similarly to what we have described in Section 2 concerning perturbations in the initial state. Therefore, the initial search landscape is very likely to be smooth in the case of ordered networks, whilst it is expected to be rather rugged for chaotic ones. Since critical BNs[2] are known to exhibit characteristics typical of ordered networks, but slightly perturbed towards chaotic ones [40], they are expected to induce a landscape with properties not dramatically differing from that of ordered ones. For each network's dynamic class we computed the empirical autocorrelation of 1000 time series obtained by collecting the objective function values along a random walk of 100 steps starting from 30 randomly generated initial candidate networks for each value of $p \in \{0.85, 0.788675, 0.5\}$. The boxplots in Fig. 4 summarise the statistics of the values of autocorrelation of the landscapes induced by the three BN dynamical regimes, respectively. Boxplots represent the main statistical features of a

[2] Generated with $p = 0.788675$.

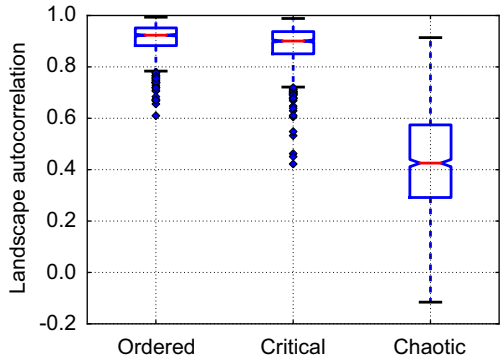

Fig. 4. Distribution of autocorrelation r of the landscapes corresponding to networks in different dynamical regimes.

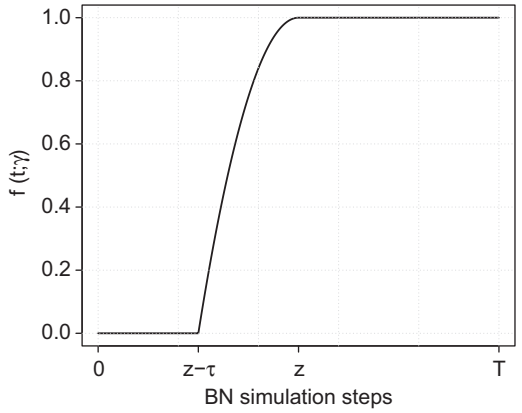

Fig. 5. Function $f(t;\gamma)$ used to assign a reward to partially successful BNs. In the figure, the function with $\gamma=2$ is plotted.

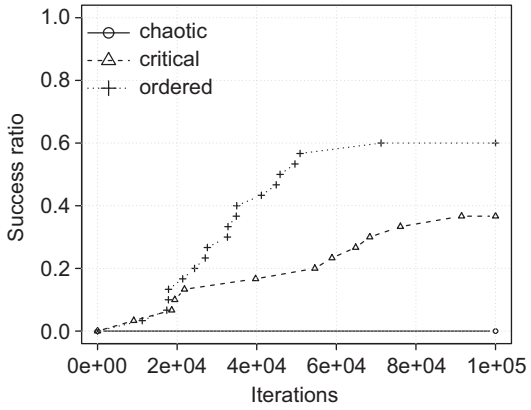

Fig. 6. Run length distribution related to task 2: reaching a target state within a given time window.

distribution in a compact way [41]. In particular, they show the median (central segment) and the 1st and 3rd quartiles (lower and upper sides of the box, respectively); sample minimum and maximum, along with outliers, are depicted as segments and points external to the box. We note that thelandscape corresponding to ordered and critical networks is highly correlated (median $r\approx 0.9$), whilst the one of chaotic networks is not (median $r\approx 0.4$). Critical BNs induce a correlated landscape, even though not as much correlated as the one induced by ordered ones. This result provides evidence to the hypothesis that initial networks' dynamical regime affects local search effectiveness. It is important to point out that BN dynamics might be less relevant when the objective function does not strongly depend upon BN trajectories in the state space. In fact, when this dependence is loose, we expect the impact of initial BN dynamics to be smaller, as we will see in Section 5.

4.1.2. Task 2: reaching a target state within a given time window

The goal of this second case study is to design a BN whose trajectory reaches a given a target state $\hat{s}$ at least once within the temporal interval $[z,T]$, but not before z, with $z\in]0,T]$. In this case, the objective function should be defined with care. Indeed, it is important that the function not only reaches its minimum if the constraint on the trajectory is reached, but it should also guide the search toward the satisfaction of such constraint. Therefore, we assign a certain reward also to those BNs whose trajectory reaches a state either almost congruent to the target one or a certain number of simulation steps τ before z. To implement this reward rule, we define a family of functions $f(t;\gamma)$ on the interval $[0,T]$ as follows:

$$f(t;\gamma)=\begin{cases}0, & t<z-\tau,\\ 1-\left|\dfrac{t-z}{\tau}\right|^{\gamma}, & z-\tau\le t\le z,\\ 1, & t>z.\end{cases}$$

The function $f(t;\gamma)$ is plotted in Fig. 5: note that BN states before z can be rewarded in several ways, depending on the value of parameters γ and τ. Let $u(t)$ be the function returning the number of nodes matching the target state at each simulation step t, with $t\in]0,T]$, the objective function f_{task2} can be described as follows:

$$f_{task2}=\begin{cases}1 & \text{if } \dfrac{u(t)}{N}=1,\ t\in]0,z-\tau],\\ 1-f(t;\gamma)\dfrac{u(t)}{N} & \text{otherwise.}\end{cases}\qquad(1)$$

Note that this objective function does not reward at all those BNs whose trajectory reaches the target state before $z-\tau$.

In Fig. 6, we show the results attained with $z=500$ and adopting the following parameter setting: $\tau=10$ and $\gamma=2$. Analogous results have been obtained with $z=50$ and $z=100$, the case of $z=500$ being the hardest for the learning process. In addition, different values of τ and γ have been tested ($\tau\in\{10,20,50,100\}$ and $\gamma\in\{0.5,1,2\}$) and no statistically significant difference was observed.[3]

The performances attained in this task are qualitatively the same as for the previous case study, even if the overall performance is lower with respect to the previous case. This is not surprising, as this task is rather more difficult than the previous one. Also for this task we computed the empirical autocorrelation of the search landscape and we obtained the same qualitative results as in the previous case.

4.1.3. Task 3: reaching a fixed point target state within a given time window

The goal of this third case study is the same as the previous one, with a further constraint: when a BN trajectory reaches the target state, then such state must be kept. In other words, the target state must be a BN fixed point. As in the previous case study, we assign a certain reward also to those BNs whose trajectory reaches a state either almost congruent to the target one or a certain number of simulation steps τ before z. At each network evaluation, let $z'\in[z-\tau,T]$ be the simulation step corresponding to the state with the largest number of Boolean

[3] We applied both χ^2 test and Fisher's test [42] to the success percentages and the null hypothesis (i.e., equal distributions) could not be rejected.

variables congruent to the target state. To verify that BN state in z' is a fixed point of the BN, it is enough to check if the state at $z'+1$ is equal to the one in z'. If this occurs, then we can assert that the BN trajectory has reached a fixed point. This statement is valid because we are considering BNs with deterministic dynamics and synchronous state updates.

Analysing the requirements, two different main features can be noticed: first of all, the network must reach the target state, but not before $z-\tau$. To evaluate this aspect, we can use the same objective function as in the previous case study, which is defined by Eq. (1). The second issue consists in making the target state a fixed point for the BN. A way to merge these two aspects is to define an objective function f_{task3} based on a weighted mean, as follows:

$$f_{task3} = \begin{cases} 1 & \text{if } \dfrac{u(t)}{N} = 1,\ t \in]0, z-\tau], \\ \alpha x(z') + (1-\alpha) y(z') & \text{otherwise,} \end{cases}$$

where $x(z')$ is defined by Eq. (1) and $y(z')$ is a function that compares the BN states in z' and $z'+1$, returning the ratio between number of not congruent Boolean variables and the total number of BN nodes. Thus, when $y(z')=0$, the BN state in z' represents a fixed point for the BN.

Different values for α account for different relative importance between reaching the target state and keeping it. We tried several values of this parameter ($\alpha \in \{0.25, 0.5, 0.75\}$) to estimate its impact on the optimisation process. We noticed that small values of α (i.e., $\alpha \leq 0.5$) lead to a slightly better performance than the one attained with $\alpha = 0.75$. In Fig. 7, we show the results obtained with $\alpha = 0.5$ and other parameters set to the same values as the results showed for the previous case study. We can note that the overall performance is better than in the previous case. We conjecture that the behaviour of the local search is positively affected by the introduction of the objective function component accounting for the fixed point constraint. In fact, once a BN is tuned such that a fixed point is reached, it is not hard to further change the Boolean functions so as to match the target state. This conjecture finds an independent support in a recent work in evolutionary robotics [43].

These experiments show that it is possible to train BNs to reach a given target state, while fulfilling additional requirements on the trajectory. The relevant point in these experiments is that the automatic procedure which realises the learning process is able to find a successful assignment of Boolean functions to the nodes by exploring a very large search space, thus showing that it is possible to control the dynamics of a BN. In addition, we also explained the reason why the performance attained starting with ordered and critical BNs is better than the one starting with chaotic networks in terms of autocorrelation of the search landscape.

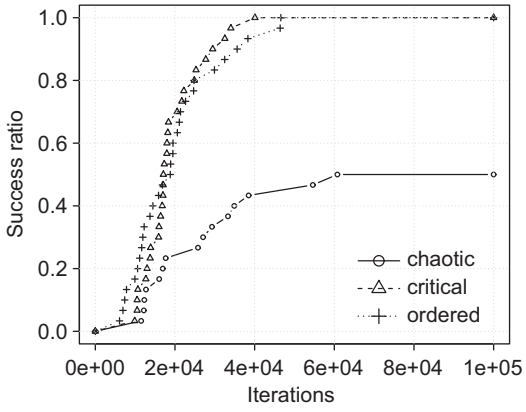

Fig. 7. Run length distribution related to task 3: reaching a fixed point target state within a given time window.

In the next section, a more complex task will be discussed in which a BN must tackle a classification problem.

5. Density classification problem

The Density Classification Problem (DCP), also known as Density Classification Task, first introduced by Packard, is a simple counting problem [44] born within the area of cellular automata (CA), as paradigmatic example of a problem hardly solvable for decentralised systems. Informally, it requires that a binary CA (or more generally a discrete dynamical system—DDS) recognise whether an initial binary string contains more 0 s or more 1 s. In its original formulation, the nodes (or cells) are arranged in a one-dimensional torus and can interact only with the neighbouring ones. The problem is designing simple rules, governing the dynamics of each node, in such a way that the system is driven to a uniform state consisting of all 1 s, if the initial configuration contains more 1 s, or all 0 s otherwise. In other words, the convergence of the DDS should decide whether the initial *density* of 1 s is greater or lower than 1/2.

Although the assignment might look trivial, it is a challenging problem and it is known to have no exact solution in the case of deterministic one-dimensional CA [45]. This difficulty stems from the impossibility to centralise the information or to use counting techniques: the convergence to a global uniform state must be obtained by using only local decisions, i.e., by using just the information available within the close neighbours of a node. Given these difficulties, various modifications to the classical problem have been proposed, including stochastic CA, CA with memory, CA with different rules succeeding in time (see [46] and references cited therein). Interestingly, some authors directly investigated the dichotomy between the local nature of the CA and the global requirements of the related DCP by allowing the presence of long-range connections within the links of the otherwise local neighbourhood [47–50]. In particular, it can be shown that the simple *majority rule* applied on random topologies outperforms all human or artificially evolved rules running on an ordered lattice [49,50]; a performance gap that increases with the number of nodes [50]. The majority rule states that the value of a CA cell at time $t+1$ is 0 (resp. 1) if the majority of its neighbours has value 0 (resp. 1) at time t.

These last two cited studies demonstrate that RBNs can effectively deal with the DCP. Our aim in this section is demonstrating that learning RBNs are flexible objects, able to attain a performance comparable to a hard-to-match benchmark such as the majority rule. Therefore, we do not use extremely large neighbourhoods or network sizes, but rather we focus our attention to the learning process itself, leaving scaling issues to further work.

In order to define the learning processes, we divide the nodes of a BN into three (possibly overlapping) groups: input nodes, output nodes and hidden nodes. This setting nevertheless does not completely specify the overall learning scheme: in fact, input nodes could maintain the initial values or could be involved in the usual BN dynamics, output nodes could have or not have feedbacks on the hidden/input nodes. Moreover, it is not clear what is the influence on the final attractors of the initial conditions of hidden and output nodes. A possibility, explored in the previous studies [51,52,56], consists in partitioning network nodes into input, hidden and output nodes. In this setting, the value of input nodes is externally imposed and does not change during network evolution, whereas hidden and output nodes are driven, as usual, by their transition function. However, in [51,56] it is also shown

that different initial settings of hidden and output nodes typically lead to different attractors, making the analysis of the network's answer difficult. For the DCP we opt for an easier choice: we establish that (a) all network nodes are input nodes, (b) all nodes are output nodes, and (c) there are no hidden nodes. In such a way, no node has a characterisation different from the usual one, the initial conditions are well defined and the correct answers can be identified with the two vectors composed by "all 0" and "all 1" values. Finally, and coherently with the Boolean nature of BNs, in order to correctly interpret oscillating asymptotic states it is enough to compute the time averages node per node, assigning "0" to the averages lower than 0.5 and "1" otherwise.

In this paper we use two groups of RBNs having, respectively, 11 and 21 nodes (odd numbers, as usual in the DCP, in order to avoid ambiguous situations where 0 s and 1 s are equally present; moreover we wished to keep computation times manageable), each with connectivity $K=3$: this choice makes the formation and detection of local majorities possible. We create a training and a testing set for each $N \in \{11, 21\}$, assembled in order to uniformly sample the whole range of the density possibilities in initial condition vectors. Thus, in the training set, if N is the number of nodes, we have N vectors having one component set to 1 and the others set to 0, N vectors having two components set to 1 and the others set to 0, etc., up to N vectors having $N-1$ components set to 1 and the other one set to 0. To that, we add N vectors having all components set to 1 and N vectors having all components set to 0, for a total of $N(N+1)$ examples. This last addition emphasises the importance of giving a correct answer when the example coincide with one of the targets. The test set is similar, but lacks the first two and the last two series of the training set, in order to avoid useless reiterations of fitness evaluation. Therefore, the set is composed of $N(N-3)$ samples.

In order to employ a BN as a classifier, we need to specify some kind of procedure, a *classification function*, that maps an example to a class. In the following we detail such procedure. A network assigns examples to classes by following these steps. Let us call these classes σ_0 (majority of zeroes) and σ_1 (majority of ones). Given an example s, a network is evolved up to an attractor starting from initial condition s; the temporal average of the attractor is computed and then the resulting vector is binarised with threshold 0.5, that is, values are rounded to the nearest integer[4]: a binary vector is so obtained. If this vector contains more zeroes than ones (respectively, more ones than zeroes) the network assigns s to σ_0 class (resp., σ_1 class).

5.1. Experimental setting

We carried out our experiments on networks consisting of classical RBNs with constant input connectivity $K=3$ and $N \in \{11, 21\}$. BNs are labelled as *critical*, *ordered* or *chaotic* according to the function bias values used to generate them. Details on network parameters are given in the following:

Critical ensemble: for each number of nodes $N \in \{11, 21\}$ and for each function bias $p \in \{0.211324, 0.788675\}$ we generated 50 networks for a total of 200 RBNs.

Ordered ensemble: for each number of nodes $N \in \{11, 21\}$ and for each function bias $p \in \{0.15, 0.85\}$ we generated 50 networks for a total of 200 RBNs.

Chaotic ensemble: for each number of nodes $N \in \{11, 21\}$ we generated 100 networks with function bias $p=0.5$ for a total of 200 RBNs.

These networks are the initial solutions in our local search algorithm and will be collectively referred to as *initial set*.

The local search used for training the BNs is ILS, described in Section 3.2. After preliminary experiments on a randomly selected subset of networks, we determined the termination criterion for the local search, which is also the only parameter to configure. We decided to stop our ILS algorithm after 150 000 networks have been evaluated. Within this limit, we observed that the local search reaches stagnation. At each improvement over the previous *incumbent*—i.e., the currently best found solution—we recorded the node functions of the new best solution.[5]

The objective function evaluates a BN classifier on the training examples. The objective function is to be minimised and has values in the $[0, 1]$ interval. The evaluation process is remarkably similar to the definition of classification function given at the end of paragraph 5, up to the computation of the binary vector: we initialise the network to be evaluated with an example s, we evolve it to an attractor A, we compute the temporal average of A and binarise it with threshold 0.5 obtaining a Boolean vector v. The contribution of $s \in \{0, 1\}^N$ to the objective function is the following: if s belongs to σ_0 (resp., σ_1) the objective function value is $w_h(v)/N$ (resp., $(N-w_h(v))/N$), where $w_h(v)$ is the Hamming weight of vector v—i.e., the number of '1' entries—and N is the network size. The contributes of all training examples are added up and divided by the number of examples seen. Notice that this definition entails that if the objective function is 0 then the BN classifier correctly classifies all examples in the training set. The converse, however, is not true: a BN might correctly classify all training examples even if its objective function is greater than zero.

In order to measure the classifying capabilities of the optimised networks, we compared them to BNs with a very specific structure. We thus generated a new ensemble, labelled *benchmark set*, whose networks have random topology, input connectivity equal to 3 and whose node functions are all equal to the Boolean majority function on three inputs. The benchmark set contains 100 BNs with $N=11$ nodes and 100 BNs with $N=21$ nodes.

5.2. Results

In this section we outline the analysis performed on data gathered from the experiments. We perform two kinds of analysis. Firstly, we assess the classification error of our optimised networks and we compare it with the BNs in the benchmark set (Section 5.2.1). Secondly, we analyse the network generated during the search process and show to what extent selected network features are affected by the optimisation algorithm (Section 5.2.2). Finally, we compare the performance of our ILS with genetic algorithms (Section 5.2.3).

5.2.1. Performance analysis

The performance of the optimised networks is measured by the *classification error*, that is, the fraction of misclassified examples.

Fig. 9 depicts the main statistical features of the distribution of the classification error on the test set. Each boxplot represents the statistics of the classification error attained in 50 independent runs (see Section 5.1), for each network category. These plots compare the networks in the benchmark set (leftmost boxplot) with the classifiers generated by our metaheuristic starting from networks in the critical (second boxplot), ordered (third boxplot) and chaotic (last boxplot) ensembles. Performances of optimised networks do not significantly differ if the local search starts from either ensemble, although the distribution of the error for the

[4] Values equal to 0.5 are rounded to 1.

[5] We recall that the local search move does not change network topology.

chaotic ensemble has the lowest minimum (for $N=11$ it ties the minimum on the ordered networks). We analysed the autocorrelation of the search landscapes, as done in Section 4. Since in this case the classification error appears similarly distributed across ordered, critical and chaotic BNs, we do not expect high variance in landscapes autocorrelation. The boxplots showing the main statistics of this analysis are depicted in Fig. 8. As in the previous test case, the autocorrelation of the landscape corresponding to ordered and critical BNs is higher than that of chaotic BNs. Nevertheless, the median autocorrelation coefficient r is rather low, ranging in [0.4,0.6] across all dynamical regimes. In addition, both the difference among the medians is quite low and the extreme values span across wide, overlapping, ranges. This result explains why, in this case study, we do not observe a difference across network's dynamical regimes as striking as in the case discussed in Section 4. It is important to stress that the objective function used in this task only loosely depends upon the BN trajectories in the state space, as it is rather affected by BNs steady state behaviour.

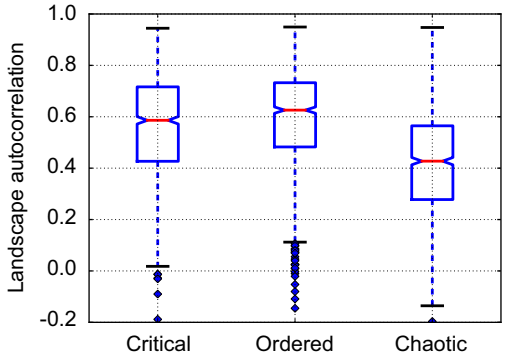

Fig. 8. Distribution of landscape autocorrelation r for RBNs with $N=21$ in different dynamical classes.

The remarkable performance gap between the benchmark and ILS observed when going from 11 to 21 nodes can be explained by the fact that, as previously noted, the majority rule cannot be improved and its classification performance increases with the number of nodes [45,50]. As a matter of fact, Fig. 9 shows that performances attained by ILS with both network sizes are similar (the median for $N=11$ is actually slightly lower).

It is also important to mention that even this optimisation scheme might be subject to *overtraining*. In this context, overtraining means that the network returned by our local search might not be the classifier that achieves the smallest classification error on the test set. Fig. 10 shows two typical examples of overtraining. These plots depict the classification error (y-axis) on both training and test sets attained by the *current best* network for every iteration of the local search (x-axis). As expected, the training error curve is non-increasing since our objective function approximates from above the classification error on the training set (see remarks at the end of Section 5.1). On the other hand, the test error curve has a global minimum before the local search reaches its optimal solution. Specifically, in Fig. 10(a) the classification error on the test set has a minimum at about iteration 10 000, while in Fig. 10(b) we have a minimum around iteration 3000.

For this reason, for each algorithm execution we kept track of all the networks generated whenever a new local optimum was found; that way we could choose the BN which minimises the classification error on the *test set*.

5.2.2. Analysis of learning process

In the following, we analyse statistical properties of the networks generated by the search process. Specifically, we compare the initial networks in each ensemble with the best classifiers

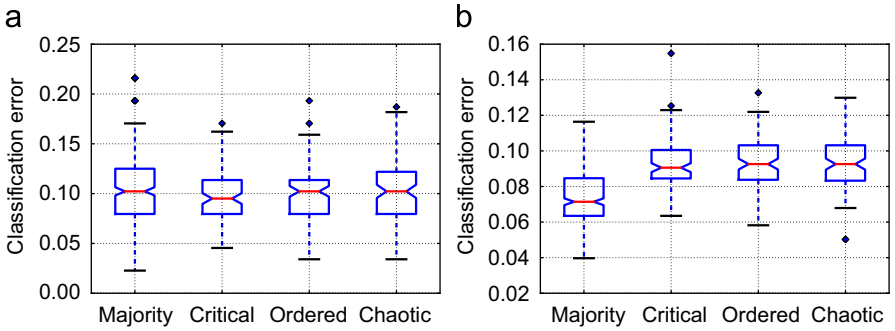

Fig. 9. Comparison of optimised networks against benchmark networks. Boxplots show distribution of classification error (fraction of misclassified examples) on the test set: (a) $N=11$; (b) $N=21$.

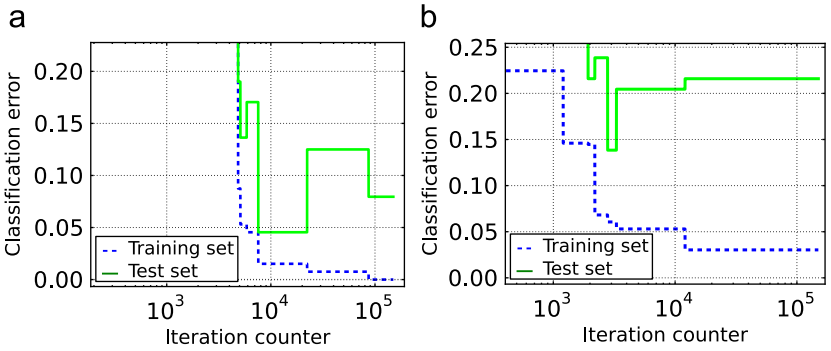

Fig. 10. Example of overtraining phenomenon on two BNs in the critical ensemble, $N=11$. The plots show the evolution of the classification error (y-axis) on training (solid line) and testing (dashed line) sets during the execution of our optimisation algorithm (x-axis shows the iteration counter). For clarity, we do not show the full range of iterations, but only a smaller interval: (a) Network 58; (b) Network 20.

returned by our local search, i.e., the best BNs that achieve the smallest error on the test set (henceforth referred to as "*optimised networks*").

The comparison is performed on the following network measures: number of attractors, average attractor period, average network *sensitivity* and *pattern distance*. In order to calculate the former two measures, we simulate a network up to an attractor starting from all the possible initial conditions.

The average network *sensitivity* (or network sensitivity for short) is a well-known measure often used in BN analyses [53,54]. Network sensitivity, determined according to the formulas in the previously cited papers, is defined as the average of node function sensitivities s_f. The sensitivity s_f of a K-variable Boolean function f measures how sensitive f is to a change in its inputs, and is calculated as follows. Let us define $s_f(x) = |\{x' | f(x') \neq f(x) \wedge d_H(x,x') = 1\}|$ where $x \in \{0,1\}^K$ and d_H is the Hamming distance. The sensitivity is thus $s_f = (1/2^K)\sum_{x \in \{0,1\}^K} s_f(x)$.

The *pattern distance* measures the average similarity of the node functions to the Boolean majority function and is computed as follows. Let us denote with $v_i \in \{0,1\}^8$ the truth table of the i-th node transition function; we compute the average over all nodes $\overline{v} = (1/N)\sum_{i=0}^{N} v_i$. We obtain a new binary vector $\pi \in \{0,1\}^N$ by rounding $\overline{v}$ to the nearest integer (0.5 is rounded to 1). We finally compute the Hamming distance of π to (0, 0, 0, 1, 0, 1, 1, 1), which is the truth table of the Boolean majority function of three inputs. Notice that a pattern distance equal to 0 does not imply that all network functions match the majority rule.

Sensitivities of the optimised networks have similar characteristics regardless of the dynamical regime of the initial networks (Fig. 11). This is a further explanation for the similarity between the distributions of the classification errors reported in Fig. 9. However, the distribution of the number of attractors of optimised networks and their average periods (not shown) just slightly differ. This observation needs more detailed considerations. Optimised networks show a sensitivity of about 1.3 regardless of both initial dynamical state *and* number of nodes. A sensitivity greater than one indicates a network with chaotic dynamics, as shown in [53], where an explicit relation between network sensitivity and Lyapunov exponent is given, and in [54] where the same conclusion is reached by a different argument. Nevertheless, a recent work has shown that networks which undergo an optimisation process could exhibit features from all dynamical classes [27]. This fact has also been confirmed in [31] and the actual results seem to provide further evidence for this property. These issues need more extended analyses and will be investigated in further work.

Fig. 12 illustrates the evolution of the pattern distance throughout the search process. The y-axis represents the pattern distance averaged over a network ensemble and the x-axis represents the iteration index. Each data point $(i,\overline{d})$ is obtained by calculating the pattern distance d on the incumbent solution at iteration i and averaging over all networks in the ensemble, thereby obtaining $\overline{d}$ (we show only two examples, since the curves of all the other cases are substantially identical). These plots further remark that optimised networks have similar characteristics. Specifically, they show that the search process is drawn towards networks with functions similar to the majority rule.

5.2.3. Comparison with genetic algorithms

Since the search landscape of the DCP is not highly correlated (see Fig. 8), a genetic algorithm (GA) could attain a better performance because of its capability of sampling wide areas of the search space. Therefore, we repeated the previous experiments by plugging a GA as the optimisation component inside the training algorithm. We tried a total of 12 different GAs which substantially differ in behaviour by selecting different recombination operators and parameters (see below). Furthermore, by these further tests, we can also address the question as to whether the results presented in Section 5.2 depend on the search algorithm used.

Coherently with all other experiments described in this paper, our genetic algorithm does not modify the network topology: all solutions in the populations share the same topology, which is

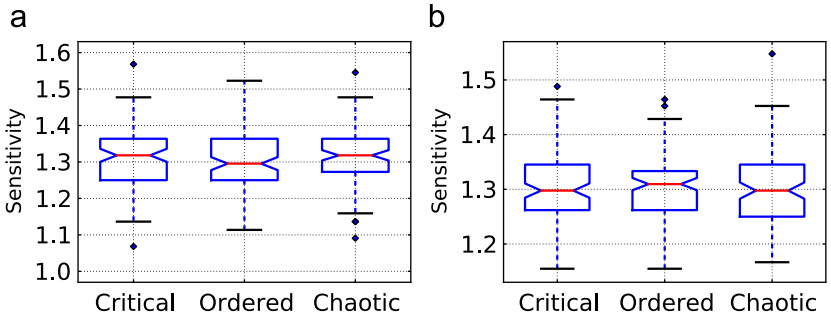

Fig. 11. Distribution of average network sensitivity for optimised networks: (a) $N=11$; (b) $N=21$.

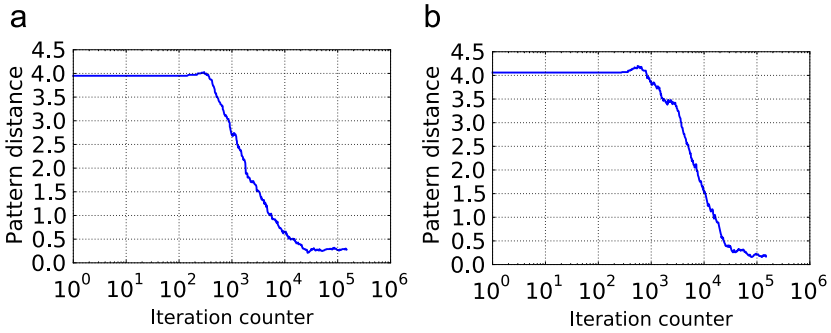

Fig. 12. Examples of typical pattern distance trend for optimised networks. Each data point represents the pattern distance value averaged on all optimised networks in (a) the critical ensemble ($N=11$) and (b) the chaotic ensemble ($N=21$).

Table 1
Summary of GA parameters.

Name	Values
Mutation operator	1-flip, 2-flip, node-function
Mutation probability	0.1, 0.2
Crossover probability	0.1, 0.9

kept fixed throughout the search procedure. The individuals of the candidate solution population in the GA are represented by genomes. Formally, a genome is a vector $\langle f_1, f_2, \ldots, f_N \rangle$ of N genes. A gene is a truth table $f_i \in \{0,1\}^8$ which defines the i-th node transition function. The offspring of the current population is built by generating new individuals by means of two genetic operators, i.e., crossover and mutation. The crossover operator is a standard two-parents one-point crossover. This operator is applied with probability p_{cross}. If crossover is not applied, the two parents are simply cloned. As for mutation, we experimented with two *ad hoc* operators. The first one, labelled *X-flip*, randomly picks X bits in a truth table and negates them. The second operator, labelled *node-function*, replaces a gene with another randomly chosen K-variable Boolean function.[6] Both operations are applied to all N genes with probability p_{mut}, meaning that, on average, $N \cdot p_{mut}$ genes are changed. We implemented a steady-state genetic algorithm solver with population size of 100 genomes and roulette wheel selection. Population overlap for the steady state is 50% meaning that at each generation an offspring of 50% population size genomes is generated and added to the current population. Afterwards, the current population is shrunk back to the initial size by removing the worst genomes.

We tested several algorithm configurations which differed in mutation operator (1-flip, 2-flip and node-function), mutation probability p_{mut} and crossover probability p_{cross} for a total of 12 configurations (see Table 1 for a summary). The reason for the choice of such extreme values for p_{cross} was to verify whether crossover introduced a too disruptive change in the network structures so as to degrade GA's performances. As for mutation operators are concerned, we chose 1- and 2-flip mutations because their effect on a BN is comparable to the application of (a small number of) local search moves. On the other hand, we chose an operator such as node-function because we wanted to test the effect of a more radical modification in a BN structure. For the sake of brevity, we will refer to a configuration with a triple (mutation operator, p_{mut}, p_{cross}). Each configuration was evaluated 30 times for each BN's dynamical class. For each algorithm evaluation, the initial population was initialised with 100 RBNs, all sharing the same topology, generated with appropriate bias depending on the dynamical class. We terminated each algorithm run after 150 000 objective function evaluations, the same termination condition used for ILS.

We performed a full factorial analysis of the configuration space and we subsequently applied a *Mann–Whitney test* [42], with a significance level equal to 0.05, to all pairs of algorithm configurations in order to obtain the best for each dynamical class of the initial networks. The test is able to identify two configuration, namely (node-function, 0.1, 0.9) and (node-function, 0.1, 0.1), that are significantly better than the others. This result holds true for all dynamical classes. Since these configurations are not distinguishable by the statistical test, we select the first one as the competitor against ILS.

[6] This last operator works only with topologies with constant in-degree, which is the case in our experiments.

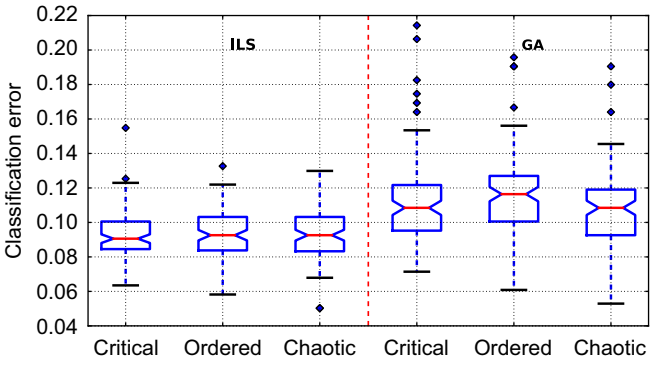

Fig. 13. Comparison between ILS and GA performance. Boxplots report the classification error distribution on the test set. Experimental setting consists of 300 RBNs with 21 nodes from each dynamical class.

Fig. 13 compares the performance of the selected GA configuration (node-function, 0.1, 0.9) (rightmost boxplots) with our ILS (leftmost boxplots) for all dynamical classes. The comparison was performed on the same set of 300 RBNs with 21 nodes introduced in Section 5.1.

First of all, we observe that the GA attains results which do not qualitatively differ from those obtained by ILS, because differences among the three dynamical classes are rather small. However, the overall performance of the GA is significantly lower[7] than that of ILS, since notches of boxplot associated to ILS do not overlap to those related to the GA [55]. This result might seem counterintuitive, given the low correlation of the search landscape. Nevertheless, we should remark that the values of autocorrelation we computed span across a wide range, therefore ILS can take advantage of its full neighbourhood exploration and find paths towards improving solutions which are not easily found by the sampling process of the GA. A detailed discussion of strengths and weaknesses of perturbative metaheuristic methods, such as ILS, and population-based ones is out of the scope of this work. However, we forward the interested reader to specialised literature [33].

6. Conclusions and future work

BNs have been mainly considered as genetic regulatory network models and are the subject of notable works in the complex systems biology literature. Nevertheless, in spite of their similarities with neural networks, their potential as learning systems has not yet been fully investigated and exploited.

In this work we use BNs as flexible objects, which can evolve by means of suitable optimisation processes, to deal with two notable issues: the problem of controlling the BN's trajectory to reach a target state and the density classification problem. The two tasks have different characteristics which have repercussions on the search itself: in fact, the initial dynamical regime could facilitate or slow down the learning process (as in the target state-controlled BN case study), or could have no particular consequences (as in the density classification problem). It is important to remark that the differences we observed in the performances are to be found in the autocorrelation of the search landscape, which depends on the objective function (plus the neighbourhood relation used in the local search), in turn affected by the BN dynamical class. If the objective function is directly influenced by the properties of the BN trajectory in the state space, then we expect the dynamical regime of initial BNs to impact search performance. When, instead, the objective function depends upon steady-state properties of the BNs, the dynamical regime is very likely to have a shallow influence on the autocorrelation of the landscape.

[7] With 95% confidence.

As a further observation, we can note that in both cases BNs successfully deal with the proposed challenges, revealing flexibility.

Critical BNs have been shown to outperform ordered and chaotic ones in terms of robustness and adaptiveness [12] and one might observe that, in this work, the optimisation processes starting from critical BNs do not show performances higher than those starting from different dynamical regimes. This is very likely to be a consequence of the static nature of the proposed tasks that have not time dependent assignments: in these cases, there are no particular reasons that can favour critical systems with respect more ordered or chaotic ones.

Further work will address the interaction between the neighbourhood definition (that is, the allowed moves of the metaheuristic algorithms) and the possibility of shaping the dynamical regime of these nets, by involving not only the functions expressed by each single networks' node, but also more global network properties as, for instance, their connectivity distribution, modularity and assortativity. Although control theory offers mathematical tools for steering simple engineered and natural systems towards a desired state, a framework to control complex self-organised systems is still lacking. BNs and the already present knowledge on their dynamical behaviour could help in this enterprise, and allow in such a way the design of specialised learning systems, able to dynamically shape their own learning capabilities in relation to the characteristics of the problem and of the search space.

References

[1] D.E. Rumelhart, J.L. McClelland, et al. (Eds.), Parallel Distributed Processing, Foundations, vol. 1, MIT Press, Cambridge, MA, 1986.
[2] J.L. Elman, Finding structure in time, Cogn. Sci. 14 (1990) 179–211.
[3] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. USA 79 (1982) 2554–2558.
[4] S. Kauffman, The Origins of Order: Self-Organization and Selection in Evolution, Oxford University Press, UK, 1993.
[5] S. Patarnello, P. Carnevali, Learning networks of neuron with Boolean logic, Europhys. Lett. 4 (1986) 503–508.
[6] M. Dorigo, Learning by probabilistic Boolean networks, in: Proceedings of World Congress on Computational Intelligence—IEEE International Conference on Neural Networks, Orlando, FL, 1994, pp. 887–891.
[7] S. Kauffman, Metabolic stability and epigenesis in randomly constructed genetic nets, J. Theoret. Biol. 22 (1969) 437–467.
[8] M. Aldana, S. Coppersmith, L. Kadanoff, Boolean dynamics with random couplings, in: E. Kaplan, J. Marsden, K. Sreenivasan (Eds.), Perspectives and Problems in Nonlinear Science. A Celebratory Volume in Honor of Lawrence Sirovich, Springer Applied Mathematical Sciences Series, Springer, Heidelberg, Germany, 2003.
[9] I. Shmulevich, E. Dougherty, Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks, SIAM, Philadelphia, PA, 2009.
[10] Y. Bar-Yam, Dynamics of Complex Systems, Studies in Nonlinearity, Addison-Wesley, Reading, MA, 1997.
[11] R. Serra, G. Zanarini, Complex Systems and Cognitive Processes, Springer, Heidelberg, Germany, 1990.
[12] M. Aldana, E. Balleza, S. Kauffman, O. Resendiz, Robustness and evolvability in genetic regulatory networks, J. Theoret. Biol. 245 (2007) 433–448.
[13] C. Fretter, B. Drossel, Response of Boolean networks to perturbations, Eur. Phys. J. B 62 (2008) 365–371.
[14] A. Ribeiro, S. Kauffman, J. Lloyd-Price, B. Samuelsson, J. Socolar, Mutual information in random Boolean models of regulatory networks, Phys. Rev. E 77 (2008) 011901:1–10.
[15] R. Serra, M. Villani, A. Graudenzi, S. Kauffman, Why a simple model of genetic regulatory networks describes the distribution of avalanches in gene expression data, J. Theoret. Biol. 246 (2007) 449–460.
[16] B. Derrida, Y. Pomeau, Random networks of automata: a simple annealed approximation, Europhys. Lett. 1 (1986) 45–49.
[17] I. Shmulevich, S. Kauffman, M. Aldana, Eukaryotic cells are dynamically ordered or critical but not chaotic, Proc. Natl. Acad. Sci. USA 102 (2005) 13439–13444.
[18] E. Balleza, E. Alvarez-Buylla, A. Chaos, S. Kauffman, I. Shmulevich, M. Aldana, Critical dynamics in genetic regulatory networks: examples from four kingdoms, PLoS ONE 3 (2008) e2456.
[19] R. Serra, M. Villani, A. Semeria, Genetic network models and statistical properties of gene expression data in knock-out experiments, J. Theoret. Biol. 227 (2004) 149–157.
[20] R. Serra, M. Villani, A. Barbieri, S. Kauffman, A. Colacci, On the dynamics of random Boolean networks subject to noise: attractors, ergodic sets and cell types, J. Theoret. Biol. 265 (2010) 185–193.
[21] S. Kirkpartick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[22] S. Kauffman, Adaptive automata based on Darwinian selection, Physica D 22 (1986) 68–82.
[23] N. Lemke, J. Mombach, B. Bodmann, A numerical investigation of adaptation in populations of random Boolean networks, Physica A 301 (2001) 589–600.
[24] A. Esmaeili, C. Jacob, Evolution of discrete gene regulatory models, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO '08, ACM, New York, NY, USA, 2008, pp. 307–314.
[25] A. Szejka, B. Drossel, Evolution of canalizing Boolean networks, Eur. Phys. J. B 56 (2007) 373–380.
[26] T. Mihaljev, B. Drossel, Evolution of a population of random Boolean networks, Eur. Phys. J. B Condensed Matter Complex Syst. 67 (2009) 259–267.
[27] C. Fretter, A. Szejka, B. Drossel, Perturbation propagation in random and evolved Boolean networks, New J. Phys. 11 (2009) 033005:1–13.
[28] A. Szejka, B. Drossel, Evolution of Boolean networks under selection for a robust response to external inputs yields an extensive neutral space, Phys. Rev. E 81 (2010) 021908:1–9.
[29] A. Roli, C. Arcaroli, M. Lazzarini, S. Benedettini, Boolean networks design by genetic algorithms, in: M. Villani, S. Cagnoni (Eds.), Proceedings of CEEI 2009—Workshop on Complexity, Evolution and Emergent Intelligence, Reggio Emilia, Italy, 2009.
[30] C. Espinosa-Soto, A. Wagner, Specialization can drive the evolution of modularity, PLoS Comput. Biol. 6 (2010), e1000719+.
[31] S. Benedettini, A. Roli, R. Serra, M. Villani, Stochastic local search to automatically design Boolean networks with maximally distant attractors, in: C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, G. Yannakakis (Eds.), Applications of Evolutionary Computation, Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2011.
[32] S. Nolfi, D. Floreano, Evolutionary Robotics, The MIT Press, Cambridge, MA, 2000.
[33] H. Hoos, T. Stützle, Stochastic Local Search: Foundations and Applications, Morgan Kaufmann Publishers, San Francisco, CA, 2005.
[34] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Comput. Surv. 35 (2003) 268–308.
[35] C. Blum, M.J. Blesa Aguilera, A. Roli, M. Sampels (Eds.), Hybrid Metaheuristics—An Emerging Approach to Optimization, Studies in Computational Intelligence, vol. 114, Springer, 2008.
[36] M. Chiarandini, T. Stützle, An application of iterated local search to graph coloring problem, in: D.S. Johnson, A. Mehrotra, M. Trick (Eds.), Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, pp. 112–125.
[37] H. Lourenço, O. Martin, T. Stützle, Iterated local search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 57, Springer, New York, NY, 2003, pp. 320–353.
[38] B. Luque, R. Solé, Lyapunov exponents in random Boolean networks, Physica A Stat. Mech. Appl. 284 (2000) 33–45.
[39] W. Hordijk, A measure of landscapes, Evol. Comput. 4 (1996) 335–360.
[40] A. Roli, S. Benedettini, R. Serra, M. Villani, Analysis of attractor distances in random Boolean networks, in: B. Apolloni, S. Bassis, A. Esposito, C. Morabito (Eds.), Neural Nets WIRN10 – Proceedings of the 20th Italian Workshop on Neural Nets, Frontiers in Artificial Intelligence and Applications, vol. 226, 2011, pp. 201–208. Also available as arXiv:1011.4682v1 [cs.NE].
[41] M. Frigge, D. Hoaglin, B. Iglewicz, Some implementations of the boxplot, Am. Stat. 43 (1989) 50–54.
[42] W.J. Conover, Practical Nonparametric Statistics, third edition, John Wiley Sons, 1999.
[43] J.C. Bongard, Spontaneous evolution of structural modularity in robot neural network controllers, in: N. Krasnogor, P.L. Lanzi (Eds.), 13th Annual Genetic and Evolutionary Computation Conference—GECCO 2011, ACM, 2011.
[44] N. Packard, Adaptation toward the edge of chaos, in: J. Kelso, A. Mandell, M. Shlesinger (Eds.), Dynamic Patterns in Complex Systems, World Scientific, Singapore, 1988, pp. 293–301.
[45] M. Land, R.K. Belew, No perfect two-state cellular automata for density classification exists, Phys. Rev. Lett. 74 (1995) 5148–5150.
[46] N. Fates, Stochastic cellular automata solve the density classification problem with an arbitrary precision, in: T. Schwentick, C. Dürr (Eds.), 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011), Leibniz International Proceedings in Informatics (LIPIcs), vol. 9, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2011, pp. 284–295.
[47] M. Tomassini, M. Giacobini, C. Darabos, Evolution of small-world networks of automata for computation, in: X. Yao, E. Burke, J. Lozano, J. Smith, J. Merelo-Guervós, J. Bullinaria, J. Rowe, P. Tino, A. Kabán, H.-P. Schwefel (Eds.), Proceedings of Parallel Problem Solving from Nature—PPSN 2004, Lecture Notes in Computer Science, vol. 3242, Springer, Heidelberg, Germany, 2004, pp. 672–681.
[48] D. Watts, Small Worlds: the Dynamics of Networks between Order and Randomness, Princeton University Press, Princeton, NJ, 1999.
[49] R. Serra, M. Villani, Perturbing the regular topology of cellular automata: implications for the dynamics, in: B. Chopard, M. Tomassini, S. Bandini (Eds.), Cellular Automata, 5th International Conference on Cellular Automata for Research and Industry—ACRI 2002, Lecture Notes in Computer Science, vol. 2493, Springer, Heidelberg, Germany, 2002, pp. 168–177.

[50] B. Mesot, C. Teuscher, Deducing local rules for solving global tasks with random Boolean networks, Physica D 211 (2005) 88–106.
[51] L. Ansaloni, M. Villani, R. Serra, Dynamical critical systems for information processing: a preliminary study, in: M. Villani, S. Cagnoni (Eds.), Proceedings of CEEI 2009—Workshop on Complexity, Evolution and Emergent Intelligence, Reggio Emilia, Italy, 2009.
[52] A. Roli, M. Manfroni, C. Pinciroli, M. Birattari, On the design of Boolean network robots, in: C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, G. Yannakakis (Eds.), Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 6624, Springer, Heidelberg, Germany, 2011, pp. 43–52.
[53] I. Shmulevich, S. Kauffman, Activities and sensitivities in Boolean network models, Phys. Rev. Lett. 93 (2004) 048701:1–10.
[54] J. Kesseli, P. Rämö, O. Yli-Harja, On spectral techniques in analysis of Boolean networks, Physica D Nonlinear Phenomena 206 (2005) 49–61.
[55] J.M. Chambers, W.S. Cleveland, B. Kleiner, P.A. Tukey, Graphical Methods for Data Analysis, Chapman and Hall, New York, 1983.
[56] M. Villani, S. Cagnoni (Eds.), Proceedings of CEEI 2009—Workshop on Complexity, Evolution and Emergent Intelligence, Reggio Emilia, Italy, 2009.

Stefano Benedettini is a Ph.D. student at *Alma Mater Studiorum* Università di Bologna (Italy) where he received his Bachelor and Master Degrees in Computer Science Engineering (2005). His main current research interests include application of metaheuristic algorithms to bioinformatics problems, complex systems and software engineering.

Andrea Roli received the Ph.D. degree in Computer Science and Electronic Engineering from *Alma Mater Studiorum* Università di Bologna, where he currently is a assistant professor. He teaches subjects in artificial intelligence, complex systems and computer science basics. His main current research interests include metaheuristics and complex systems, with applications to swarm intelligence, bioinformatics and genetic regulatory network models. Andrea Roli is a member of the steering committee of the Italian Association for Artificial Intelligence (AI*IA).

Mattia Manfroni is a software engineer and collaborates with *Alma Mater Studiorum* Università di Bologna (Italy) as a free researcher since September 2010. He received his Bachelor and Master degrees in Computer Science Engineering, both from Università di Bologna, in January 2008 and July 2010, respectively. His research interests are in artificial intelligence and swarm robotics.

Marco Villani received his honours degree in Physics in 1992; in 1993 his thesis obtained the "Franco Viaggi" awards of the SFI (the Italian Physic Society). He worked at ENEA and then at the Environmental Research Centre of Montacatini S.p.A (Montedison Group); from 2005 is a researcher and professor in Engineering and Computer Science at the University of Modena and Reggio Emilia, where he also leads the Modelling and Simulation Laboratory. Winner of the best paper awards at ACRI2006 and ECCS2010, the two major European conferences, respectively, on cellular automata and complex systems, Villani applies complex systems concepts in areas that require strong interdisciplinary interactions.

Roberto Serra graduated in Physics at *Alma Mater Studiorum* Università di Bologna, and later performed research activities in the industrial groups Eni and Montedison, where he served as a director of the Environmental Research Centre until 2003. Since 2004 he is a full professor of Computer Science and Engineering at the Modena and Reggio Emilia University. His research interests concern several aspects of the dynamics of complex systems, paying particular attention to biological and social systems, and to the dynamical approach to Artificial Intelligence. He published more than 130 papers in international journals and refereed conference proceedings, and is a co-author of four books. He has been responsible of several research projects, funded by companies, by the Italian Ministry for Scientific Research (Miur), by the National Research Council (CNR) and by the European Union. He has served as a member of the program committee of several international conferences, and has delivered various invited talks and seminars. He recently chaired two international conferences, one on Artificial Life and Evolutionary Computation (Venice, 2008) and one on Artificial Intelligence (Reggio Emilia, 2009). Roberto Serra has also been president of AI*IA (Associazione Italiana per l'Intelligenza Artificiale) and is currently chairman of the Science Board of the European Centre for Living Technologies.

Antonio Gagliardi started out as an analyst and programmer for TELECOMSpA and TecnostMael SpA Olivetti Group. In 1998 he began his activities as a lecturer for computer courses in public and private education; he worked for Modena Formazione on courses funded by the European Community (2001–2003). In 2007 he graduated in Hypermedia Communications at Parma University, followed by a post-graduate degree in Economics and Complex Systems at the University of Modena and Reggio Emilia. Currently he works as a lab technician and collaborates with the University of Modena and Reggio Emilia. His research interests are in machine learning and Boolean networks.

Carlo Pinciroli is a Ph.D. student at IRIDIA, CoDE, Université Libre de Bruxelles in Belgium. Before joining IRIDIA, in 2005 he obtained a Master's degree in Computer Engineering at Politecnico di Milano, Milan, Italy and a second Master's degree in Computer Science at University of Illinois at Chicago, IL, USA. In 2007 he also obtained a Diplôme d'études approfondies from the Université Libre de Bruxelles. The focus of his research is computer simulation and swarm robotics.

Mauro Birattari received his Master's degree in Electronic Engineering from Politecnico di Milano, Italy, in 1997; and his Doctoral degree in Information Technologies from Université Libre de Bruxelles, Belgium, in 2004. He is currently with IRIDIA, Université Libre de Bruxelles, as a research associate of the fund for scientific research F.R.S.-FNRS of Belgium's French Community. Dr. Birattari co-authored about 100 peer-reviewed scientific publications in the field of computational intelligence. Dr. Birattari is an associate editor for the journal Swarm Intelligence and an area editor for the journal Computers & Industrial Engineering.