Genetic Evolution of the Topology and Weight Distribution of Neural Networks

Vittorio Maniezzo

Abstract— **This paper proposes a system based on a parallel genetic algorithm with enhanced encoding and operational abilities. The system, used to evolve feedforward artificial neural networks, has been applied to two widely different problem areas: Boolean function learning and robot control. It is shown that the good results obtained in both cases are due to two factors: first, the enhanced exploration abilities provided by the search-space reducing evolution of both coding granularity and network topology, and, second, the enhanced exploitational abilities due to a recently proposed cooperative local optimizing genetic operator.**

I. Introduction

GENETIC ALGORITHMS (GA's) for training and designing artificial neural networks (ANN's) have proved to be a useful integration, when not a viable alternative to more common algorithms, such as backpropagation. Genetically evolved ANN's have often been tested in the literature on a standard problem test suite relating to Boolean function learning, originally proposed by Rumelhart [48] to evaluate the effectiveness of backpropagation. Several investigations, [31], [51], [55], obtained comparable results on those problems using genetically evolved nets.

Gradient-descent learning algorithms, i.e., learning algorithms that modify the network parameters following the gradient of an error function, have difficulties in learning the topology (number of nodes and their connections) of the net whose weight distribution they optimize [58]. In contrast, encouraging results have been obtained using GA's to design both network topology and the associated weighted connections, with applications to animat control [44], face recognition [25], and other problems [38], [39], [56].

Search in the space of topologies is of extreme interest because it relieves the network designer of the burden of identifying the network structure that will allow a problem to be solved. The structure must otherwise be designed using one of the following three approaches: a difficult *a priori* analysis of the potentialities of a net with a given topology; the use of oversized networks that can result in enormous search spaces for the learning technique, making it far slower and less effective than it could be; and finally, the use of computationally prohibitive construction-destruction algorithms.

The main computational advantage of evolving the network's topology is the possibility to autonomously identify minimal search spaces containing reachable solutions to the specified problem. The dimension of search spaces can be further affected by another net coding feature: the bit length of weight coding; i.e., the coding *granularity*. In the system presented in this paper, coding granularity is a parameter that evolves concurrently to the net structure.

Granularity is a coding parameter that has always been specified in the design phase and whose computational impact is at least as great as that of network topology. Given a function to be learned, having therefore fixed the adaptive landscape (which is the primary determinant of the "ease" of evolvability), too coarse a granularity may prevent the existance of a suitable solution, while too fine a granularity may lead to huge search spaces, retarding the speed of learning (see also [2] for related results).

In this paper, it is shown how searching over the space of granularities can lead to network coding advantages that are synergistic with those due to searching in the space of topologies. An algorithm, called ANNA ELEONORA (standing for Artificial Neural Networks Adaptation: Evolutionary LEarning Of Neural Optimal Running Abilities), is presented that includes both these possibilities.

In particular, it is shown how genetic evolution can reduce both weights coding granularities and network topologies. Short coding allows rapid identification of promising distributions of the network's weights, eventually to be refined by subsequently increasing the code length. Good topologies entail a small number of nodes of the net and a loose connectivity among them; both these aspects essentially contribute to reducing the number of links of the net, thus reducing the length of the coding strings that have to be optimized by the GA [36].

The proposed procedure is not restricted to ANN evolution, but it can be used whenever a real-valued fitness function has to be discretized to allow binary-coded solutions [35].

The serial and parallel implementation of the proposed system has been tested first over Rumelhart's standard problem test suite. A completely different application area, robot control, was then chosen and the controller of an autonomous computational agent (termed an *animat*, after Wilson [57]) that had to learn sequences of behaviors was evolved.

This paper is organized as follows: Section II introduces GA's and Parallel GA's (PGA's) and concludes with a brief overview on how they have been applied to ANN learning. Section III presents the ANNA ELEONORA system, both in its sequential and parallel versions. Section IV contains the computational results obtained by applying ANNA ELEONORA to Boolean function learning. Section V illus-

Manuscript received March 31, 1993; revised August 9, 1993.

The author is with the Artificial Intelligence and Robotics Project, Dip. Elettronica e informazzione, Politecnico di Milano, 20133 Milano, Italy.

IEEE Log Number 9214268.

trates a second application of ANNA ELEONORA, relative to the animat control problem; Section VI summarizes the results achieved, and outlines efforts for future research.

II. Background

Genetic algorithms [22], [28], are a computational model inspired by population genetics. They have been used mainly as function optimizers and they have been demonstrated to be effective global optimization tools, especially for multimodal and non-continuous functions [16]; even NP-hard combinatorial optimization problems have been effectively tackled [13], [17], [42]. Their strength is essentially due to their updating of a whole population of possible solutions at each iteration; this allows for a parallel exploration of the search space [5], [28]. The following provides an introduction to the GA computational paradigm and to its most successful parallel versions (for a thorough review see [18]).

A. The Sequential Genetic Algorithm (SGA)

The genetic algorithm evolves a multiset[1] of elements, called a *population* of *individuals*. Each individual $x_i (i = 1, \cdots, n)$ of the population X represents a trial solution of the problem to solve. Individuals are usually represented by strings of variables, each element of which is called a *gene*. The value of a gene is called its *allelic value*, and it ranges on a set that is usually restricted to $\{0, 1\}$, but that can potentially be continuous and even structured (Koza [32] reports applications where the allelic values are LISP S-expressions).

A GA is capable of maximizing a given *fitness function* (FF) computed on each individual of the population. If the problem is to minimize a given *objective function*, then it is required to map increasing objective function values into decreasing FF values; this can be achieved by a monotonically decreasing function.

The standard SGA is the following sequence:

Step 1: Randomly generate an initial population $X(0) := (x_1, \cdots, x_n)$;

Step 2: Compute the fitness $FF(x_i)$ of each individual x_i of the current population $X(t)$;

Step 3: Generate an intermediate population $X_r(t)$ applying the *reproduction* operator;

Step 4: Generate $X(t+1)$ applying other operators to $X_r(t)$;

Step 5: $t := t + 1$; if not (*end_test*) goto Step 2.

The most commonly used operators are the following:

1) Reproduction (selection): This operator produces a new population, $X_r(t)$, extracting with repetition individuals from the old population, $X(t)$. The extraction can be carried out in several ways. One of the most commonly used is the *roulette wheel selection* [22], where individuals are extracted in probability following a Monte Carlo procedure. The extraction probability $p_r(x_i)$ of each individual x_i is proportional to its fitness $FF(x_i)$ as a ratio to the average fitness of all the individuals in $\mathbf{X}(t)$.

[1] Recall that a multiset is a collection of elements, each of which can be present more that once in the extensional definition of the multiset.

2) Crossover: This operator is applied in probability, where the crossover probability is a system parameter, p_c. To apply the standard crossover operator (several variations have been proposed) the individuals of the population are randomly paired. Each pair is then recombined, choosing one point in accordance with a uniformly distributed probability over the length of the individual strings *(parents)* and cutting them in two parts, accordingly. The new individuals *(offspring)* are formed by the juxtaposition of the first part of one parent and the last part of the other parent.

3) Mutation: The standard mutation operator modifies each allele of each individual of the population in probability, where the mutation probability is a system parameter, p_m. Usually, the new allelic value is randomly chosen with uniform probability distribution.

4) Local search: The necessity of this operator for optimization problems is still under debate. Local search is usually a simple gradient-descent heuristic that carries each solution to a local optimum. The rationale behind this has been first advocated by Mühlenbein [42], suggesting that search in the space of local optima is much more effective than search in the whole solution space; recent results on the Quadratic Assignment and on the Timetable Problems [14], [37] support this hypothesis.

B. Parallel Genetic Algorithms (PGA's)

GA's are good candidates for effective parallelization, given their proportional nature. However, the classical genetic algorithm is not straightforwardly parallelizable because of the need (in the selection and in the crossover steps) for global statistics that result in high communication costs.

Several approaches have been proposed to overcome this problem. The different models used for distributing the computation and to ease parallelization follow two main approaches: the *island model* [8], [9], [30], [33], [43], [45], [51], and the *fine-grained model* [23], [34], [41], [42], [47], [53]. These two models naturally lend themselves to different parallel hardware: MIMD machines, typically transputer-based, in the case of the island model, and SIMD machines, array processors or Connection Machines, in the case of the fine-grained model (which has also proved to be very effective with transputer-based implementations).

In the *island model*, several isolated subpopulations evolve in parallel, periodically exchanging, by migration, their best individuals with the neighboring subpopulations.

In the *fine-grained model (or neighborhood model)*, a single population evolves, each individual of which is placed in a cell of a planar (or toroidal) grid; selection and crossover are applied only between neighboring individuals on the grid. This is the approach implemented in ANNA ELEONORA.

This distributed model contains a notion of *spatiality*, in that the individuals of the population are assigned to a specific position in a given spatial environment, usually a two-dimensional grid implemented as a toroidal array in order to avoid border effects. In this way a single population of individuals is maintained, over which a neighborhood relation is defined that specifies spatial proximity among individuals. The operators of

the GA are seen as stochastic local updating rules: the resulting model is thus fully distributed with no need for any global control structure.

Selection is performed, cell by cell, only over the individual assigned to that cell and its neighbors; crossover mates neighboring individuals, while mutation is standard. In addition, other operators have been proposed, which will be outlined below. The basic algorithm shared by all systems following the fine-grained model is as follows:

Step 1: Randomly generate an initial population of candidate solutions (individuals) and assign each of them to a cell of a grid; compute the fitness of each individual.

Step 2: For each cell c_{ij} of the grid execute steps 3, 4, 5, and 6.

Step 3: Select the new individual to occupy c_{ij} among those assigned to c_{ij} and to its neighbors.

Step 4: Randomly select an individual in the neighborhood of c_{ij} and with probability p_c, recombine it with that assigned to c_{ij}; assign one of the offspring to c_{ij}.

Step 5: Mutate the individual assigned to c_{ij} with probability p_m.

Step 6: Compute the fitness of the new individual assigned to c_{ij}.

Step 7: If not(*end_test*) goto Step 2.

The *end_test* is usually a test on the number of generations or on the time-length of the run, but it could also consider more subtle phenomena, such as search stagnation.

A comparison of the performance of this algorithm with that of the standard one shows that the fine-grained approach provides a more careful exploration of the search space, due to its local selection mechanism that alleviates the selection pressure [34]. Easier problems are therefore solved less efficiently, asthe extra exploration represents only extra computational load with no reward, while more difficult problems benefit from this thorough search [11].

The optimal neighborhood size depends on the problem faced, but even small neighborhoods (9 or 25 cells, centered on the current one) provide a robust choice ensuring good performance.

C. Introduction to ANN Evolution

The use of evolutionary learning for ANN is recent but considerable (see [50] or [58] for reviews). Several different and possibly synergistic approaches have already been presented. Most of the applications regard the use of evolutionary algorithms, usually genetic algorithms, as a means to learn connection weights; some applications also regard learning of network topology; and only a few use evolutionary algorithms to define the parameters of a learning algorithm (backpropagation, Hebbian or others) that will be used in the training phase. This section briefly outlines the main approaches so-far-proposed for evolving ANNs.

Evolutionary learning for ANNs has been introduced to perform a global exploration of the search space, thus avoiding the problem of stagnation that is characteristic of local search procedures.

When evolving the connection weights of a network with a given topology, the usual approach is to code the net as a string obtained by concatenating the weight values one after the other. Weight values can be either binary coded [7], [55], or real—valued [19], [40], [51].

Standard genetic algorithms, with no tricks to sped up convergence, are very robust and effective for global search, but very slow in fine-tunning a good solution once a promising region of the search space has been identified [37]. Not surprisingly therefore, Kitano [31] found that for three-layer networks and not too rugged fitness functions, GA's were outperformed by fast variants of backpropagation (BP).

The idea to hybridize the two approaches, GA and BP, follows naturally. As presented in Section II, there is a trend in the GA community to routinely use local search as a module for GA applied to optimization problems [37], [42] and BP appears to be a natural local search integration for GA's, in the case of ANN optimization.

Kitano [31] proposes some empirical evidence that GA/BP mating does not provide any advantage over a randomly initialized, multiple application of quickprop (afast variant of BP) alone, at least for shallow networks and easy fitness functions. But Belew, McInerney, and Schraudolph [3] and Hancock and Smith [26] report successful results with a hybrid approach.

A second research area is concerned with the evolution of network topologies, together or independently from the evolution of relative weights. Miller, Todd, and Hegde [39] suggested that GA's are very good candidates to search in the space of topologies, because the fitness function associated with such a space has several properties that make it unappealing for classic methods: it is complex, noisy, nondifferentiable, multimodal, and deceptive. Experimental results support this hypothesis.

This observation led several researchers to concentrate on the topic. Miller, Todd, and Hegde [39] identified two approaches to code the topology in a string: the *strong specification scheme* (or direct encoding scheme), where each connection of the network is specified by its binary representation, and a *weak specification scheme* (or indirect encoding scheme), where the exact connectivity pattern is not explicitly represented but it is computed on the basis of the information encoded in the string by a suitable developmental rule.

The strong specification scheme has been adopted by Whitley, Starkweather and Bogart [56] and by Schaffer, Caruana and Eshelman [49]. One acknowledged advantage of this approach is that it is easy to evolve networks with special connectivity properties, either by constraining the representations allowed or by including some specific penalty term in the fitness function. One disadvantage is that these representations do not scale well to large networks, with hundreds of nodes and thousands of connections. But when dealing with networks of smaller size, the strong specification scheme induces a less-fuzzy fitness function, i.e., the fitness value associated with each chromosome is more coherent when it is computed several times using different training sets. It therefore permits

gaining better insight into the network structural properties that generate good or bad results.

The weak specification scheme has been adopted for example by Kitano [31], Harp, Samad, Guha [27], Merrill and Port [38], and others. It is more biologically plausible and it uses much shorter network representations. It is possible to evolve representations that are translated into connectivity patterns by given developmental rules [27] or to evolve the developmental rules themselves [31]. Learning developmental rules has the advantage of a good scalability of the resulting systems, as rules do not change with the size of the network, but interpretation of fitness values is more difficult because there is a complex procedure that translates genotypes into phenotypes.

III. Anna Eleonora

This section presents the evolutionary algorithm on which the ANNA ELEONORA system is based. It consists essentially of a genetic algorithm, used for learning both topology and for connection weights; however, two new and general mechanisms have been incorporated. The first consists of a new genetic operator, so far untested in complex discrete optimization problems, called GA-simplex recently proposed by Bersini and Seront [4]. The second consists of a new encoding procedure, called *granularity encoding* [35], [36], that allows the algorithm to autonomously identify an opportune length of the coding string (therefore the discretization of the potentially real-valued weights). The possibility to evolve the granularity is allowed by the interpretation of a segment of the string as the value of the length control parameter to be applied to the string itself, a process much like the evolution of mutation distributions carried on by evolution strategies [1] and by evolutionary programming [20].

In Section III-A we detail the structure and the functioning of the GA used, by making reference to its sequential version; in Section III-B we present the parallel version that has been run on a Connection Machine.

A. Sequential Implementation

The utilization of evolutionary optimization in a specific application entails two related activities: the definition of the encoding structure and the definition of the operators to use. In the case of evolution of ANN's substantial work has already been presented, allowing some insight in the strengths and weaknesses of these alternatives.

Our choice has been to evolve both the net architecture and its weight distribution. However, the representation scheme followed differs from the ones already presented in the literature because greater emphasis is placed on the minimization of the length of the coding string (and consequently of the search space). We propose a variable-length binary encoding, where also a control parameter—*granularity*–is represented. Standard genetic operators act upon these strings: only the probability distribution of mutation is modified to gain efficiency.

1) Encoding scheme: An extended direct encoding scheme is used , where each connection is represented directly by its binary definition. This allows for achieving better genotype evaluation than could be achieved by using an indirect encoding scheme and a developmental rule. This reliability in fitness assessment is needed especially in the animat control problem, where fitness information is intrinsically fuzzy (different fitness assessments for the same network—same phenotype—are very likely to result in different fitness values; the fitnesses are therefore fuzzy numbers [59]) and time-consuming to compute.

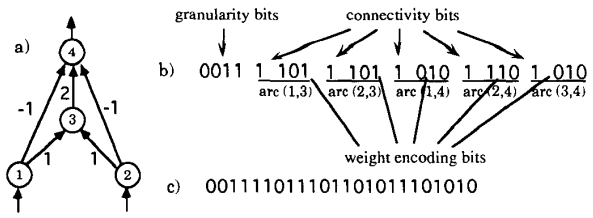


Fig. 1. Network encoding of a complete topology graph: (a) Annotated code, in which connectivity is coded by presence/absence bits (the *connectivity bits*) relative to each possible arc of the complete graph and the first byte of the string specifies the number of bits (the *granularity*) according to whichthe weights of the present connections have been codified. (b) Actual string, as seen by the GA.

The internal organization of the encoding string is decisive for the effectiveness of crossover. Different organizations were tried and generated, obtaining substantially different results, both in terms of quality of the solution found and in speed of convergence. The representation chosen uses the networks nodes as basic functional units and encodes all the information relevant for a node in nearby positions, including its input connectivity pattern and the relative weight distribution. Connectivity is coded by presence/absence bits *(connectivity bits).* When connection is present, immediately after each connectivity bit there is the binary encoding[2] of the relative weight. The first byte of the string specifies the number of bits (the *granularity*) according to which the weights of the present connections have been codified.

For example, Fig. 1 presents a simple network and its coding; for simplicity, biases have been omitted.

The network presented in Fig. 1 is layered, but not strictly layered,[3] given the existence of connections (1, 4) and (2, 4). All possible connections are present, and in fact all connectivity bits are set to 1 [Fig. 1(b)]. The encoding string is composed by a fixed-length initial substring, which encodes the granularity: in this example the length of the substring is 4 bits and it encodes a granularity value of 3. This means that all the weights of the present connections are encoded using 3 bits. Every connection (arc of the topology graph) is then repre-

[2] Gray coding does not provide any significant improvement of performance. In fact, Gray codes with a Hamming distance of 1 do not always represent contiguous integers and the average distance of integers Gray-coded by strings differing in only one bit is not less than the average distance of integers binary-coded by string differing in only one bit.

[3] In this document we use the term *layered feedford neural networks* to refer to those networks whose topology digraph is partioned in such a way that nodes within each partition—called *layers*—are not connected, and that a total ordering is imposed upon layers, so that nodes of a layer can be connected in input only to nodes of lower-order layers. Only the lowest-order layer has the nodes connected to the network inputs, only the highest order layer has the nodes connected to the network outputs. A *strictly layered feedforward neural network* is a layered network whose nodes belonging to level i receive their inputs only from the nodes of level $i-1$ and provide their outputs only to the nodes of level $i+1$.

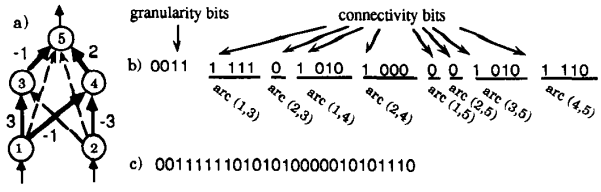


Fig. 2. Network encoding of a sparse topology graph: (a) Annotated code, where some connectivity bits are set to 0. (b) Actual string seen by the GA.

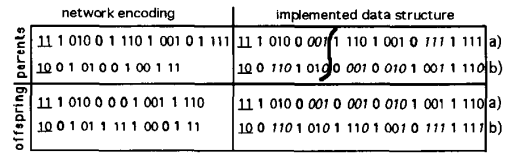


Fig. 3. Two networks of different granularity and with different topology undergo crossover. The networks are encoded in the left-hand side column's "parent" strings (a) and (b), but the data structures used by ANNA ELEONORA represent them as in the right-hand side column.

sented by a substring, whose first bit is the relative *connectivity bit,* possibly followed by the weight encoding. These substrings are ordered in the chromosome in such a way that connections coming in to the same neuron are kept nearby. In Fig. 1 in fact, the first two connection-encoding substrings refer to the inputs of node 3, the remaining ones to the inputs of node 4. Each of these substrings has its connectivity bit, followed in this case by three *weight encoding bits,* which can represent 8 different weight values, ranging from -3 (000) to 4 (111).

Fig. 2 shows another net coded with granularity 3, where some connections are declared nonexistent. These nonexistent connections are drawn with dashed arcs on the graph on the left and the corresponding connectivity bit is 0 in representation a). Note how no weight follows 0 connectivity bits, leading to a variable-length encoding even for networks with the same granularity.

The possibility of having strings of different length is further emphasized by the granularity mechanism. As already mentioned, coding granularity is a control parameter whose value is coded within the string it is used to interpret. The possibility of evolving a control parameter together with the individual it contributes to control is not new: it is a distinctive feature of other evolutionary paradigms [1], [20] and it has already been used in the context of genetic evolution of neural networks, for example by Harp, Samad and Guha [26], who proposed a combined genetic/backpropagation learning algorithm and encoded BP parameters in the individuals, together with the network structure.

Granularity evolution is a mechanism that could be adopted in any problem of optimization of binary-coded functions, and is not restricted to the optimization of ANN's.

2) Operators: ANNA ELEONORA employs four operators: reproduction, crossover, mutation, and GA-simplex.

a) Reproduction: This is the standard roulette wheel reproduction operator [22], with MonteCarlo selection with probabilities based on fitness levels. As no explicit local optimizing operator (but see the discussion on GA simplex later on), is used, we introduced fitness scaling—a procedure that enhances differences among similar fitness values [22]—to improve fine-tuning of the solutions. An elitist strategy [16] has also been implemented (in the sequential version), preserving the best individual of the last iteration in the new population.

b) Crossover: There has been some debate in the literature about the opportunity of applying crossover to the evolution of ANN's. Kitano [31] points out that ANN's training seems to lack the compositionality property that makes crossover worthwhile. (Compositionality assures that a partial solution in a lower-dimensional subspace is likely to be part of an overall good solution). Moreover, Hancock [26] identified a permutation problem, stating that there can be functionally identical networks that differ only for permutations of the hidden nodes: if two such networks, well fit, are mated, the offspring will probably be far worse than any of the parents, since duplicates of some hidden neuron will be present in both of them while other useful neurons will be missing.

These and other considerations have induced Fogel, Fogel, and Porto [19], Parisi, Cecconi, and Nolfi [42] and others to use mutation as the only operator.

We do not want to delve into theoretical considerations here, but surely compositionality is highly codification-dependent. Section IV presents some computational results showing how the use of crossover can benefit the evolution, using our representation scheme and allowing the GA to evolve both architecture and weights.

The crossover used is standard: a single cutting point chosen with uniform probability over the string length and a swap of the genetic material following it (see Section II). A simple implementational trick allows mating of network with different connectivity and/or different granularity, with no modification of the operator. Coded networks are actually implemented using fixed-length arrays (bit strings), defined with the maximum possible length (all connection present and maximal granularity). Crossover is applied to those strings and therefore it has no problems in application. Granularity and connectivity bits are actually used to determine how many of the weight bits following each connectivity bit are to be considered part of the network encoding. Fig. 3 exemplifies a crossover of two different-length strings: granularity bits are underlined and connectivity bits are boldface; non-chromosomic bits are italic; spaces have been inserted in the strings to ease readability.

In the example of Fig. 3, two networks of different granularity and with different topology undergo crossover. The networks are encoded in the "parent" strings a) and b), as they are proposed in the left column of Fig. 3, but the data structures used by ANNA ELEONORA represent them as in the right column. Note how parent a) has granularity 3 while parent b) has granularity 2. The length of the encoding of parent a) is 20 bits, of parent b) is 14 bits; however, the arrays used to store them in memory are both 26 bits long. A cutting point is randomly chosen: in Fig. 3 it is located between two different connection substrings, but it could have been placed also among weight encoding bits, or among granularity bits or anywhere else in the string. The result of crossover are the two offspring a) and b). Offspring a) has granularity 3 and a 17-bit-long encoding, offspring b) has granularity 2 and a 16-bit-long encoding.

This way of processing allows the use of a standard operator: the only non-standard effect from the viewpoint of the GA is a bias in the choice of the cutting points toward the end of weight coding substrings, as it is possible to cut among bits that will not be used either for search by the GA, or for fitness evaluation. Obviously the search space is still constrained by the granularity and the connectivity and does not extend to all possible internal representations [36]. Note in fact that the invisible bits do not affect the search process: the only moments when they are imortant for the seach is when they become visible, following an increment of granularity. For example, when granularity is mutated from 1 to 3 (supposing maximum granularity greater than three), there are two new low-significance bits encoding the weights of each connection present in the network. In this case, the new bits can be randomly initialized, or can be given the value of zero or of one, or they can be initialized in any way that seems computationally efficient.

c) Mutation: This is the standard mutation operator, which negates each bit with probability p_m, except that we split it into three: a granularity bit mutation operator (applied on each granularity bit with probability p_{mg}), a connectivity bit mutation operator (applied on each connectivity bit with probability p_{mc}) and a weight bit mutation operator (applied on each weight bit with probability p_{mw}). This has been done because of the different interpretation of the bits, which suggests that a unique mutation probability value will be suboptimal with respect to three different values.

d) GA simplex: This operator has recently been presented by Bersini and Seront [4] as a way of hybridizing the exploration effectiveness of GA's with classical optimization methods for local optimization of the GA individuals. Several authors have advocated the need of a local optimizing operator [37], [42], which departs from the biological inspiration of the GA paradigm, but which has proved to dramatically increase the effectiveness of the search process.

Local search in the case of of feedforward ANN's evolution is usually performed by applying backpropagation (which is a gradient-descent algorithm with respect to the error function) to the individuals of the GA population [26], [31]. GA allows search over the whole search space, thus providing the possibility of escaping from local minima (but [21] indicates that GA can stall anywhere in the search space). BP locally optimizes the weight distribution. There are two main drawbacks in this approach: it is extremely computationally expensive and it requires too many computations of the objective function. GA simplex, on the other hand, does not require any extra computation of the objective function, and it optimizes on the basis of a *set* of solutions, thus promoting cooperation among individuals. The use of this operator ensures more exploitational power to the GA, even though it does not guarantee the local optimality of the solutions achieved.

GA-simplex is a ternary operator, inspired by the linear programming simplex procedure [15], that tries to exploit the fitness landscape identified by the extant solutions. It works on three individuals of the population x_1, x_2, x_3 and generates a new one, x_4, on their basis. The simple constructive algorithm is:

individuals	ff(x)=x (max)	
1011101	93	
0110110	54	→ 1110101 (117)
0011010	26	

Fig. 4. Example of a GA-simplex application, where a fitness function given by the interpretation of each individual as a binary coding of an integer value is maximized.

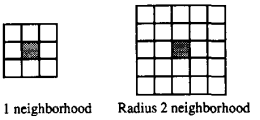


Fig. 5. Neighborhood structures used in the parallel algorithm on the CM2 Connection Machine.

Step 1: Rank the three parents by fitness value; suppose $FF(x_1) \geq FF(x_2) \geq FF(x_3)$.
Step 2: For each ith bit of the strings
Step 3: if $x_{1i} = x_2 i$, then $x_{4i} = x_{1i}$
Step 4: else $x_{4i} :=$ negate(x_{3i})

The original proposal suggested to apply step 4 in probability. We experimentally found better results applying it deterministically and subjecting the use of the operator to a specific probability parameter, p_s.

An example of the application of GA simplex is presented in Fig. 4, where we maximize a fitness function given by the interpretation of each individual as a binary coding of an integer value.

B. Parallel Implementation

After having defined and tested the algorithm presented in the previous section, we parallelized it to exploit the potentialities it offers on parallel hardware, both for GA evolution and for ANN fitness assessment. Of the two possibilities presented in Section II, we chose to implement a fine-grained model and let it run on a CM2 Connection Machine. The parallelization involved the following adjustments of the basic sequential algorithm.

Each individual x_{ij} of the population is assigned to a cell c_{ij} of a toroidal grid. A *neighborhood* of each cell $N(c_{ij})$ is defined, consisting of the cell itself and the 8 immediately connected ones (radius 1 neighborhood) or of the cell itself and the 24 ones surrounding it (radius 2 neighborhood), as shown in Fig. 5.

Every operator is applied in parallel to each individual, on the basis of its neighborhood and has effect on the cell containing the individual; in the following, t is supposed to be the iteration counter of the current generation and $t+1$ that of the next.

1) Reproduction: Reproduction is still implemented as a Monte Carlo procedure. It assigns to the current cell in the next population, $c_{ij}(t+1)$, one individual of the neighborhood $N(c_{ij}(t))$ of the cell in the current population chosen on the basis of a reproduction probability proportional to the fitnesses of the individuals of the neighborhood.

2) Crossover: Crossover assigns to $c_{ij}(t+1)$ one individual obtained mating $x_{ij}(t)$ with one randomly chosen in $N(c_{ij}(t)) - x_{ij}(t)$. Having so identified the pair of parents, crossover goes on as described in Section III.

3) Mutation: It is standard mutation.

4) GA simplex: It assigns to $c_{ij}(t+1)$ one individual obtained applying the procedure described in Section III to the triplet formed by $x_{ij}(t)$ and two individuals randomly chosen in $N(c_{ij}(t))$.

IV. Boolean Functions Learning

In this and in the following section the results obtained applying ANNA ELEONORA to two different problems, Boolean functions learning and robot control, are presented. The first set of tests was run in order to identify an optimal parameter settings and to evaluate the effectiveness of the approach on a standard set of problems; the second experiment was designed to evaluate the robustness of the approach and to provide indications on the opportunity of learning neural controllers.

A. Rumelhart's Test Suite

The set of problems used was originally proposed by Rumelhart, Hinton, and Williams [48] to assess the effectiveness of BP. We used six of the proposed problems:

1) Xor: A 2-input, 1-output problem, where the output is the exclusive or of the input values. The initial configuration of the network was 2-2-2-1 (2 input neurons, 2 neurons in the first hidden layer, 2 neurons in the second hidden layer and one output neuron).

2) Parity: A 4-input, 1-output problem, where the output should be 1 if there is an odd number of 1s in the input pattern, 0 otherwise. The difficulty of the problem is due to the fact that input patterns differing in only one bit require opposite outputs. The initial configuration of the network was 4-4-4-1.

3) Encoding: A 8-input, 8-output problem, requiring a layered net with only 3 neurons in the hidden layer. The output should be equal to the input for any of the 8 combination of seven 0s and one 1. The net is required to build an internal representation of the input provided. The configuration of the network was 8-3-8, strictly layered. Evolution in this case was not allowed to modify the network topology.

4) Symmetry: A 4-input, 1-output problem where the output is required to be 1 if the input configuration is symmetrical, 0 otherwise. The initial configuration of the network was 4-4-4-1.

5) Addition: A 4-input, 3-output problem, where the output should be the result of the sum of two 2-bits input numbers. The initial configuration of the network we used to solve it was 4-4-4-3.

6) Negation: A 4-input, 3-output problem, where the three output bits should be equal to the three rightmost input bits if the leftmost one (acting as negation bit) is 0, to their negations if it is 1. This problem can be seen as a set of three exclusive or between the negation bit and each of the three other (rightmost) input bits. The initial configuration of the network was 4-4-4-3.

As the neurons have a real-valued output ranging on $[-1, 1]$, we scaled -1 to 0 and computed the fitness as the LMS error; a standard genetic scaling mechanism [22] has also been used to enhance difference among similar networks.

TABLE I
BEST PARAMETER SETTINGS

p_{mw}	0.0005
p_{mc}	0.01
p_{mg}	0.1
p_c	0.4
p_s	0.9
γ	1
n	100

B. Parameter Settings

The algorithm has 7 parameters and there is no way to *a priori* identify useful combinations of values. The parameters are: n, cardinality of the population; p_c, crossover probability; p_s, GA-simplex probability; p_{mg}, granularity mutation probability; p_{mc}, connectivity mutation probability; p_{mw}, weight mutation probability; and γ, slope of the neuron sigmoid transfer function in the origin. In order to define good settings extensive computational tests with different combinations of values have been performed. All tests were made using the Addition problem and letting the algorithm run ten times at each setting for 3000 generations. The values used for each parameter were: $n \in \{30, 50, 100, 200\}$; $p_c \in \{0, 0.2, 0.4, 0.7, 0.9\}$; $p_s \in \{0.5, 0.7, 0.9, 1\}$; $p_{mg} \in \{0.01, 0.1, 0.3, 0.5\}$; $p_{mc} \in \{0.001, 0.01, 0.1, 0.3\}$; $p_{mw} \in \{0.0001, 0.0005, 0.001, 0.01, 0.1, 0.3\}$; $\gamma \in \{0.1, 0.5, 1, 2, 10\}$; we tested almost 500 different combinations of these values. The best settings obtained are presented in Table I.

Some considerations are in order.

- As expected, the best values for the three proposed mutation probabilities are different; in particular, p_{mw} is the order of $1/l$ where l is the string length (in accordance with the prescriptions of Mühlenbein [41]) while the other two mutation probabilities are larger, suggesting a less disruptive impact of the mutations they control.
- Crossover probability is smaller than the commonly adopted values suggested by De Jong (0.6, in [16]) or by Grefenstette (0.95, in [24]). This suggests a less efficient building block decomposition than in the case of function optimization; however, very small crossover probabilities or no crossover at all resulted in lower performance. There is, therefore, empirical evidence that, within the representation used, functional (building) blocks can indeed be identified and the search process benefits from their recombination.
- There is evidence of the effectiveness of GA simplex. Its use greatly improves the results obtained, even though applying it too often (i.e., with a probability > 0.9) can lead to early convergence and local minima problems.
- Larger populations provide better results. But the computational cost of evolving large populations does not seem to be rewarded by a comparable improvement of the solutions obtained, for $n \geq 30$. In fact, for $n = 30$ we obtained an average of E(FF) = 14.46 over the ten runs, each one lasting $t \approx 15'$; for $n = 50$ we obtained E(FF) = 14.52 and $t \approx 30'$, while for $n = 100$ we obtained E(FF) = 14.58 and $t \approx 60'$. Therefore, we considered $n = 30$ to be a good choice for sequential evolution. Far larger

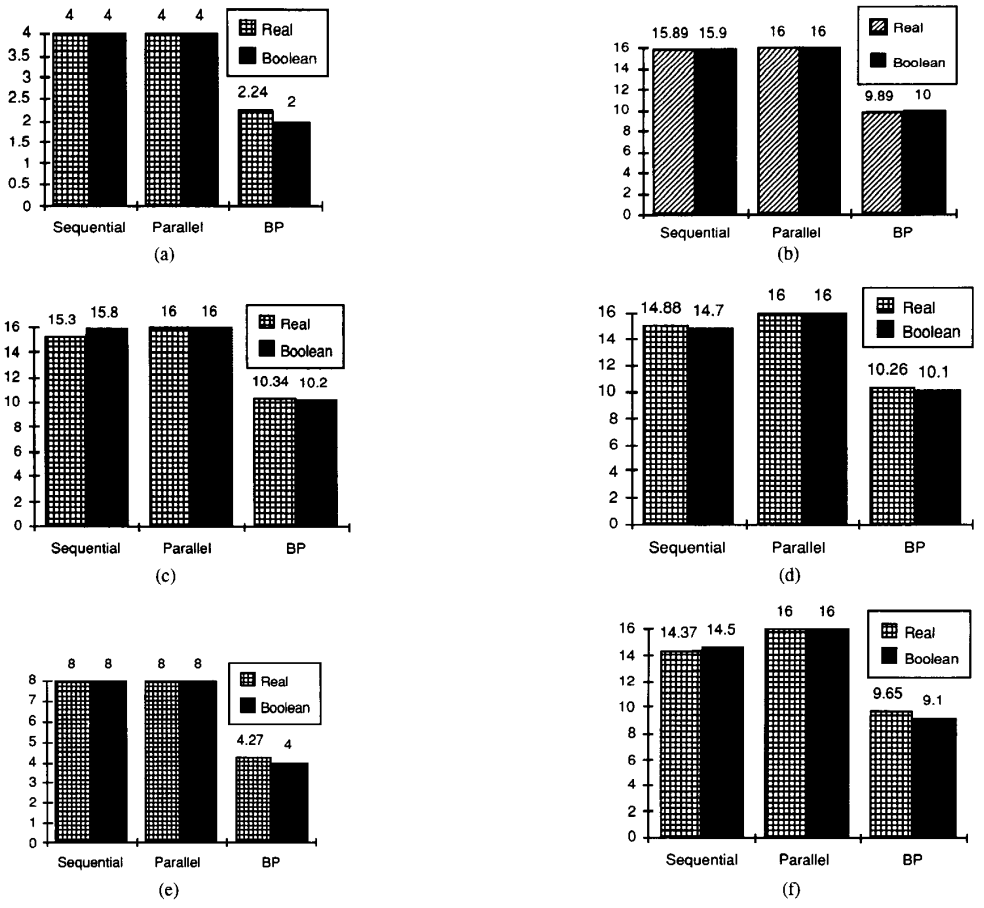


Fig. 6. Performance evaluation. Average results of 10 trials of the sequential version, of the parallel one, and of BP, in terms of both real-valued and Boolean fitness (which is the fitness obtained by interpreting the output values as 0 or 1, according to a threshold paramenter), applied to: (a) the Xor problem, (b) the symmetry problem, (c) the parity problem, (d) the addition problem, (e) the encoding problem, (f) the negation problem.

populations have been tested on the Connection Machine ($n = 4096$ and $n = 16384$), with the results presented in Section IV–C.

- The sigmoid slope is a parameter that dramatically affects the results: a slope of 5 almost makes it a step function, while a slope of 0.2 results in a very mild inclination. Given the Boolean discretization of the weights, mild inclination ensures more input combinations to non-input neurons and more graded output, but it also entails a correspondingly harder search space. The best slope value corresponds to a rather steep sigmoid, and so it is for neurons that have a quasi-Boolean output behavior. This is correct in this application, but as we will see in Section V, it cannot be generalized to every class of problem.

C. Search Effectiveness

Having identified a good parameter setting, we started a set of runs to evaluate the effectiveness of ANNA ELEONORA on Rumelhart's test suite. Each problem was tested ten times, both on a vax (the sequential model) and on a CM2 (the parallel model). Each run was stopped after 3000 iterations.

For comparison, we also implemented a basic backpropagation learning procedure and tested it on the vax. BP was executed over the same number of function evaluations used by the GA's ($30 \times 3000 = 90\,000$ evaluations). BP was run with the parameter settings proposed by Rumelhart *et al.* [48] but it was applied to networks with the maximally redundant topology that ANNA ELEONORA encoded in its genomes; i.e., with the topologies specified in Section IV-A.[4]

As the neurons we use are real-valued, we implemented a thresholding procedure to binarize the network output values; however, the fitness was computed by a scaled LMS of the error. It is therefore possible to have suboptimal fitnesses for networks that have correctly learned the function proposed. In Fig. 6 we present, for each test problem, the average results of the 10 trials of the sequential version, of the parallel one,

[4]This could seem unfair to BP, but we already know that given a simple topology and good intial weights BP can solve the presented problems to the optimum; we also know that modified BP algorithms can solve them efficiently. The purpose of our data is simply to present how ANNA ELEONORA compares to basic BP; any improvement that new BP versions can assure with respect to the results produced by the basic one we implemented could also apply to the outcome we present here.

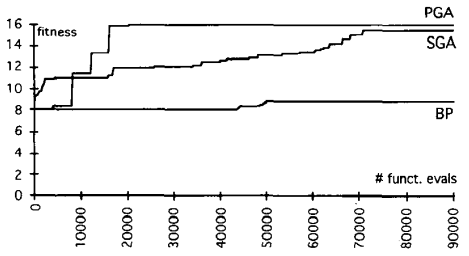


Fig. 7. Evolutions of the FF of the networks that achieved the median result over the 10 trials of the PGA, the SGA, and BP applied to Neg. plotted against the number of function evaluations.

and of BP, in terms of both real-valued and Boolean fitness (which is the fitness obtained interpreting the output values as 0 or 1, according to a threshold parameter).

Again, some considerations are worth making:

- The Xor problem was consistently solved optimally by ANNA ELEONORA in a few dozen generations. This was the only problem for which the parallel model often already had an optimal solution in the first randomly constructed population; i.e., an optimal solution could be identified for the xor with 4096 random trials.
- The parallel version was able to solve to the optimum every instance of every problem, usually within 200 iterations. Tests of our PGA involved populations of 4096 individuals and a radius 2 neighborhood, and lasted about 20' on a CM2 in time-sharing mode.
- BP has a high variance in results over the 10 runs. Sometimes it moves steadily (though slowly) toward the optimum, sometimes it never greatly modifies the initial fitness, which is close to the half of the optimum. In Fig. 7 the evolutions of the FF of the networks that achieved the median result over the trials for the negation problem are presented, plotted against the number of function evaluations. In the case of the PGA, the number of function evaluations is computed as population size (4096) times the number of generations; this causes the stepwise ascent of the fitness function, where each step has in fact a width of 4096.

D. Irrelevancies Detection

ANNA ELEONORA does not contain any explicit mechanism for removing nodes from the network. However, it is possible that all the connections of a node are deleted, so that the node is actually removed from the network. The possibility of removing arcs is one of the main mechanisms for shortening the string length, and thus for speeding up search: it is therefore common to have sparsely connected neurons, or neurons removed entirely.

Fig. 8. shows the topology of one of the evolved nets for the xor problem. Two neurons have been identified as useless and completely removed from the network. This is very interesting, given the very small number of iterations needed by the algorithms to find an optimally performing network, and thus for having the possibility of getting a better fitness when removing connections.

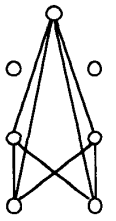


Fig. 8. Topology of one of the evolved nets for the Xor problem. Two neurons have been identified as useless and completely removed from the network.

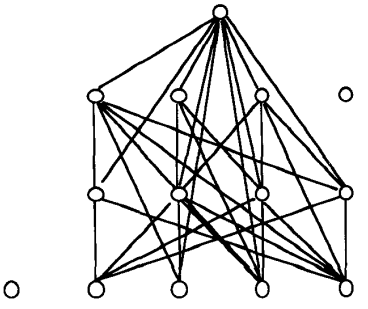


Fig. 9. Evolved topology for a modified parity problem with five inputs, one of which just added noise, being non-influential for parity detection. The added neuron was correctly identified and completely removed from the net (together with a neuron of the second hidden layer).

To explicitly test this intrinsic capability of the algorithm, we have applied ANNA ELEONORA to a modified parity problem with five inputs, one of which just added noise, being non-influential for parity detection. ANNA ELEONORA could also solve this modified instance. The resulting topology is shown in Fig. 9. The added neuron has been correctly identified and completely removed from the net (together with a neuron of the second hidden layer). This effectiveness is somewhat surprising: we expected to have some low-weight connection from the noisy neuron, but in this case the algorithm outperformed our highest expectations.

As a final remark, note that the networks proposed by our algorithm do not have the minimal possible topology for solving the problems they face (the more so because they are given no reward for minimizing their topology). However, there is empirical evidence that the algorithm implicitly works in that direction, stopping as soon as it finds a topology related to a good solution.

E. Structure Optimization

The process underlying the results presented in Section IV-D, connection removal, has been directly tested by a modified encoding problem. We proposed the problem to a 8-4-4-8 network, where the nodes of a layer were connected in input to all those of the layers below (i.e., the constraint of using a *strictly* layered topology was removed). The result is shown in Fig. 10, where connections with low weight have not been reported for ease of interpretation.

The system has correctly discovered that the most efficient way to solve the problem is to directly connect input to corresponding output neurons, thereby directly copying the input values into the output ones..

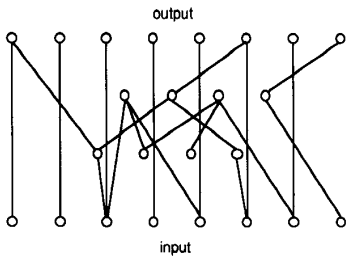


Fig. 10. Optimized structure for the modified encoding problem, in which the nodes of a layer were connected in input to all those of the layers below. The system has correctly discovered that the most efficient way to solve the problem is to connect inputting directly to corresponding output neurons, thereby directly copying the input values into the output values.

TABLE II
GRANULARITIES OF BEST NETWORKS

	Granularity	Best value	Iteration
Xor	3	4	877
Enc	3	8	2838
Par	3	16	545
Neg	6	15.42	2795
Sym	3	16	1580
Add	7	15.44	2694

F. Search for Smaller Spaces

One of the main assumptions of our work is that, given the possibility, the search process tries to identify a promising rregion of search space by reducing the length of the coding strings. This has been confirmed in the case of the connection removal mechanism; here we present some results for the granularity updating case.

Table II presents the granularities of the best-performing networks for each problem. Note that easy problems (Xor, Par) were solved in few iterations with low granularity, while problems that were harder to solve (Neg, Add) were best tackled with high granularities. The high value of p_{mg} ensures that the search always considers all granularity spaces, so it is possible, as in the case of Enc, to find an optimal solution with low granularity even in a very late phase of the search.

The effort to reduce granularity is also evident during the search. Fig. 11. indicates the evolution of the best performing net and its granularity over a 5000 iterations on a negation problem. Note how in the first 300 iterations the algorithm tunes in to a promising search space, with granularity of 5 in this case. It explores other possibilities but then it sticks to one granularity value and gets increasingly better results with that value.

In Fig. 12 the same process is presented for a 50-iteration run on the Xor problem (optimal fitness value of 4). Again, the first iterations reduce the granularity; note how at generation 16 a solution with granularity 3 could beat, for some very small increase of fitness value, the previous better solution that had granularity 4, thus lowering the final optimal granularity.

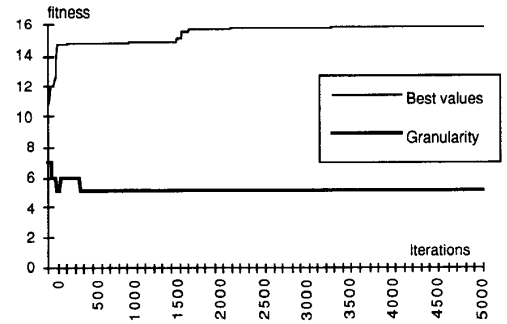


Fig. 11. Granularity and fitness evolutions for the negation problem; the effort to reduce granularity is evident.

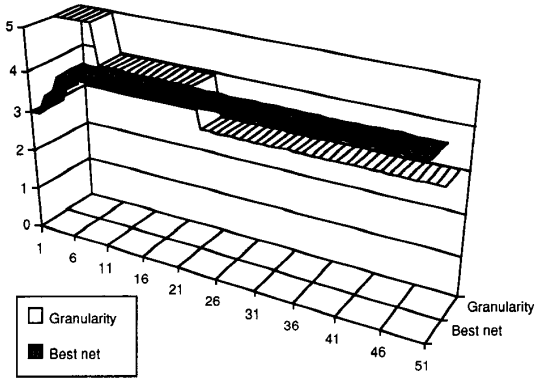


Fig. 12. Granularity evolution for the Xor problem.

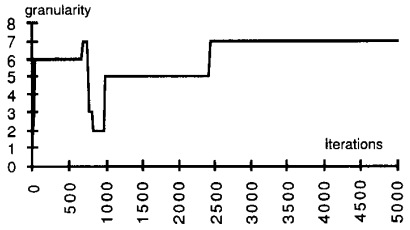


Fig. 13. Granularity evolution for the negation problem, resulting in augmented granularity.

Fig. 13 shows the granularity evolution of a run on the neg problem where the exploration of the search spaces is particularly evident (the best value found for this particular run was 14.96). The granularity of the best individual at the first iteration was 4; then different search spaces were explored until the best solution found had a granularity of 7, which is bigger than the intial value.

As a final remark, note that the granularity value obtained at the end of a run is not guaranteed to be the optimal one; i.e., the minimal value that allows the possibility to represent an optimal solution. If an optimum has not been found, it is possible that with more iterations, or with a re-initialization of the population, better solutions are found with different granularity. Conversely, if an optimal solution was found there is no way to change the granularity of the best-so-far solution; even when it would be possible to express the optimum with less bits: there is in fact no explicit fitness premium for lowering granularity.

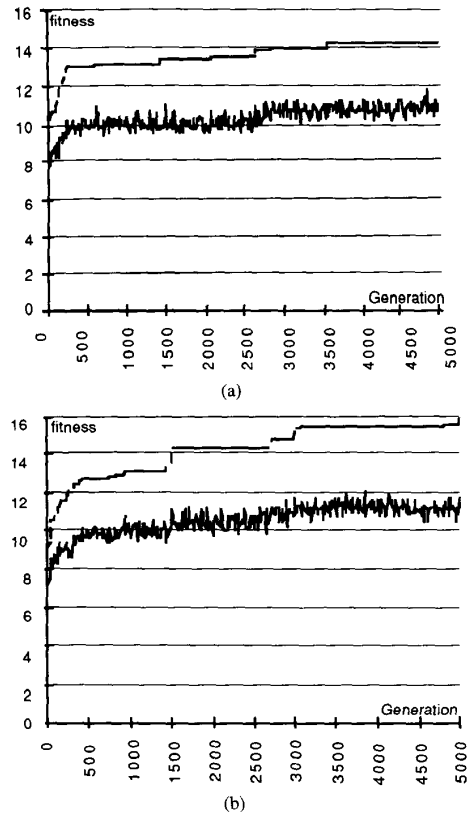


Fig. 14. Comparison of two runs on the negation problem: fitness evolution (a) with and (b) without GA-simplex. The GA-simplex–based algorithm could better exploit new genotypic information contained in the genome variations.

G. Effectiveness of GA-Simplex

GA simplex ensures a much more effective search, both in terms of the quality of the solution and of speed of convergence.

To get some insight, compare the two runs on the negation problem presented in Fig. 14 (a) and (b), respectively without and with GA simplex. The two figures present the evolution of the FF values plotted against the generation at which that fitness was obtained. In both figures, the upper curve depicts the best value in the population, the lower curve indicates the average fitness value of the population.

While the beginning of the search is similar for the two, even somewhat slower for the GA simplex version, the difference is manifested around generation 1500. Both algorithms improved their solution, but the GA simplex-based algorithm could better exploit the new genotypic information contained in the genome variation that allowed for the improvement, quickly achieving a sizable increase in fitness. The same increase in FF happened around generation 2800.

V. Learning to Control Animats

The second application tackled with ANNA ELEONORA involves an animat [57] control problem. Specifically, we are interested in providing an animat with a small set of sensorial and motorial primitives and the goal to learn how to correlate stimuli to actions in order to maximize a fitness function defined over the space of its possible behaviors.

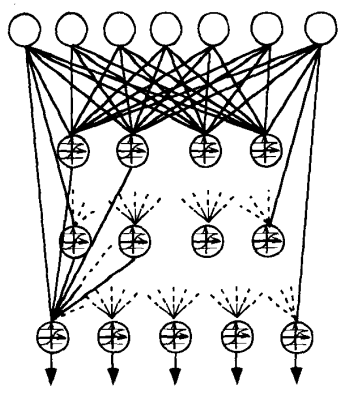


Fig. 15. Structure of a player's neural controller. The seven inputs of each agent network are the basic sensory inputs the player has available (i.e., the coordinates) relative to the player, of both the players' goals and of the ball, and the status of the ball (controlled or not by the player). The five outputs of the network are the elementary actions the single player can perform: it can move itself and kick the ball—in this case, the relative coordinates of the point—about the playing field; where the ball is kicked to is also recorded.

Several evolutionary approaches are possible, including LCS [29], [57], genetic programming [32]; and genetically evolved neural controllers ([10], [44]). While in general the problem of relating stimuli to actions for stimuli-response (SR) systems, i.e., for systems for which a reward always immediately follows the choice of an action, can be solved with current techniques, the problem is more difficult if, in order to solve it, the animat has to learn to perform a sequence of behaviors [12], [52].

In particular, a neural controller characterized by a static distribution of weights, in order to sequentialize two actions, has to define weights so that the fulfillment of the first triggers a different neuronal activity pattern that leads to the execution of the second action. This always requires a very delicate balance of the weight distribution to learn.

The testbed for the experiment uses an environment originally designed for the study of the emergence of cooperative behavior among members of a team. It consists of an environment where the agents have to learn to play football (soccer, to American readers) [6].

A. The Task to Learn

One agent is set on a field, where one ball and two goals are also present. The inputs of each agent network (Fig. 15) are the basic sensory inputs the player has available; i.e., the polar coordinates, relative to the player, of both the opponant's and the player's goals and of the ball, and the status of the ball (controlled or not by the player). This implies that the sensorial input of a player is coded into seven input neurons.

The outputs of the network are the elementary actions the single player can perform; it can move itself and kick the ball, in this case, the relative coordinates of the point, in the playing field; where the ball is kicked to is also coded. This makes a five-neuron output.

The inputs and the outputs are connected to two hidden layers with a parametric number of neurons. The outputs of the neurons of each layer are calculated as a function of the weighted sum of all the neurons of the levels below; the transfer function is a sigmoid whose slope is a parameter.

A population of neural networks is associated to each player and is evolved by ANNA ELEONORA. Each network is coded as described in Section III. The fisrt experiments run evolved one single player, and subsequent runs considered multiple players (Section IV-D).

The goal of the agent is to learn to score into the opposite goal. To do that, it has to learn to go to the ball, to get it, to drag it toward the opponant's goal, and eventually to kick it into the goal.

B. The Learning System

The evolution of neural controllers required the definition of a suitable fitness function. The number of scored goals is ineffective, because the huge amount of connections and of neuron activation combinations for the controller make extremely improbable the random construction of a network capable of controlling an agent to score a goal. We had therefore to reward intermediate achievements (a procedure analogous to that used to teach animals to perform behavioral sequences [12]).

The fitness function has been split into several components, and is implicitly hierarchical, given the different orders of magnitude of the rewards it provides to the player.

- A very small reward is given whenever the distance between the player and the ball is below a given distance threshold. This allows the early selection of individuals that are attracted by the ball and do not wander aimlessly or stay still within the field.
- A higher reward is given to individuals that drag the ball toward the opposite score. This intermediate level has experimentally proven necessary, as otherwise ball-controlling players never autonomously (i.e., by chance) evolved a goal-seeking behavior.
- Finally, a very high reward is given to players when they score a goal. Note that no specific reward is given for kicking the ball into the goal instead of dragging it into it.

To assess the fitness of a player, it is randomly put in the field, with the ball set in a random position. It receives its inputs and deterministically computes the relative motorial output, possibly modifying its environment; this process is iterated for a number of times, which is regulated by a system parameter. To minimize the possibility of having players that receive very high fitness only because of fortunate environmental conditions (such as finding the ball and the opposite goal in a straight line), each player is given three chances to demonstrate its effectiveness under different environmental conditions, and the average fitness value is retained for reproduction purposes.

The system wasexecuted using this fitness function and the best parameter settings determined in the Boolean functions study. The only parameter that we changed, with respect to the Boolean optimal values, is the sigmoid slope. As was mentioned in Section III, optimal sigmoid slopes are problem-sensitive, and a network acting as a controller needs much more sensibility in the activation pattern than a network that has to compute a Boolean function. A set of possible gamma values was tested, and the better-performing one in this case was $\gamma = 0.1$.

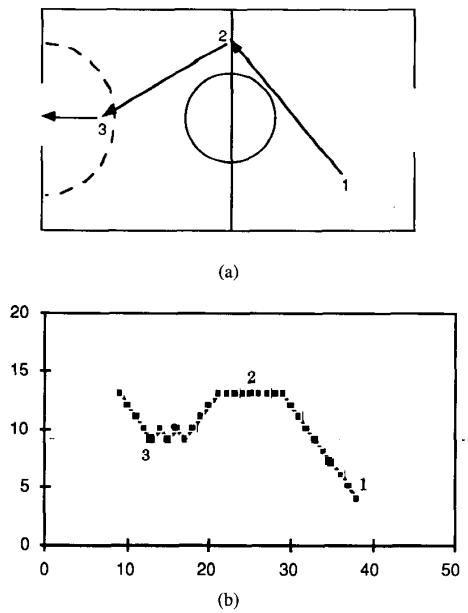


Fig. 16. A typical learned action. Fig. 16(a) sketches the main phases of the performed action. Fig. 16(b) shows the actual positions occupied by the player during the considered action. In Fig 16(b) the figures on the axes refer to the pixel coordinate on the screen.

C. Evolution Results

Within a few hundreds iterations the system was able to evolve networks capable of displaying the desired behavioral sequence. Fig. 16 indicates a typical action of a player. Fig. 16(a) sketches the main phases of the performed action. Fig. 16(b) shows the actual positions occupied by the player during the considered action. In Fig. 16(b) the figures on the axes refer to to the pixel coordinate on the screen.

The evolved behaviors are the following:

1) The player perceives the ball as the most interesting input, disregarding those relative to the goals positions, and it moves toward the ball.
2) The player gets the ball and suddenly the opposite player's goal becomes the most relevant input; the player drags the ball toward the goal.
3) The player has realized a position in the field, internal to the dashed area, that will allow it to score if it kicks.

Note how, in the actual action, after having kicked the ball the player does not move any more toward the goal but follows a meaningless trajectory. This is due to the sudden change of the input denoting the possession of the ball. Note also the zig-zagging final approach to the score: since we compute independently the x- and y-velocity, there is no penalty for going that way rather than following a straight line.

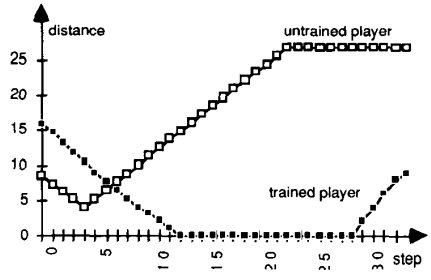


Fig. 17. Distance from the ball of a trained and of an untrained player during actions starting with the same environmental setting. The trained player heads toward the ball, gets it, drags it for 16 steps, and then kicks it.

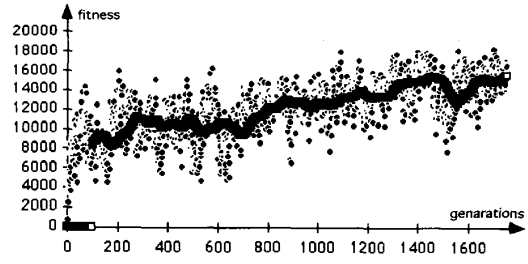


Fig. 18. Fitness values of the best player at each generation for a 1800–generation evolution (dotted line) and moving average of 100 data (heavy line).

Fig. 17 shows the distance from the ball of the player during the action presented in Fig. 16 compared to that of a player at the beginning of the evolution. Note how the untrained player is initially set in a more favorable position, but it just sticks to a straight random trajectory, which initially goes close to the ball, but then departs from it until the border of the playing field is met. The trained player instead heads toward the ball, gets it, drags it for 16 steps and then kicks it.

The evolution of the net allowed the construction of a static internal representation of the task to face, that allows the player to change behavior both in front of definite variations of the input pattern and of very subtle ones. An example of a definite variation is the ball_possess bit switching from off to on at the end of action 1; a subtle one takes place when the activation of the input relative to the distance to the opposite goal goes below the threshold; this indicates the possibility of a successful kick.

The kicking behavior emerges depending on the number of actions a player is granted, for the computation of its fitness function. If they are few (6065 in our system), the player has difficulties in fetching the ball and dragging it inside the goal, so the player quickly learns to kick it. When more actions are allowed (80–90), the players find it more advantagious to drag the ball into the goal, because they receive an extra reward due to the longer time of possession of the ball. If still more actions are granted, it becomes possible to score multiple goals and the kicking behavior emerges again, even though it is not dominant.

In Fig. 18 we present the fitness evolution of a simulation, where the reward for the possession of the ball was set to 1, that for dragging it toward the opposite goal to 5 and the reward for scoring to 1000.

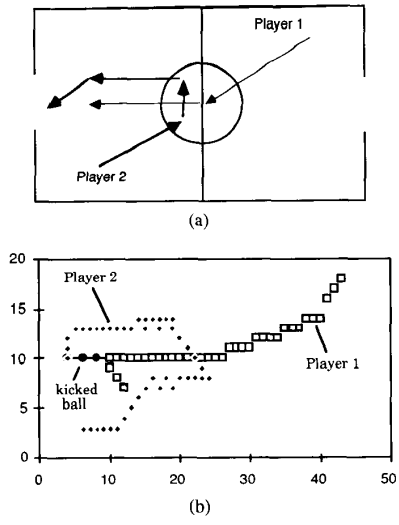


Fig. 19. An observed action in which a player was able to intercept a kick. Fig. 19(a) shows the abstract schema. Fig. 19(b) shows the actual successive players and ball positions on the field.

As a final remark, we want to point out the generalization capability of the evolved network, that is able to correctly instantiate the learned behavioral sequence for any environmental configuration, even for those that it has never experienced in the training phase.

D. Experiments with Multiple Players

Experiments were conducted with more than one player on the field. Each player is unable to perceive the others, it does not have the necessary input units, but the evolution becomes competitive, since only one player at a time can control the ball and score. Players must therefore put a premium on reaching the ball. Moreover, when one player catches the ball, it is slowed down, so that the others can reach it and tackle for the ball. With this modality therefore, players learn to reach for a moving ball. Another possibility that has been given to the player is that of intercepting a kick. If a kicked ball passes sufficiently near to a player, it is intercepted by the player. Fig. 19 depicts an observed action in which an interception took place. Fig. 19(a) shows the abstract action schema. Fig. 19(b) shows the actual successive players and ball positions on the field.

The results of this kind of training are however still preliminary and will be considered in a second part of this research, where players will be allowed to perceive other players, opponents and team-mates, in the hope to witness cooperative behaviors emerge.

VI. Conclusion

This paper presents an evolutionary computational approach suitable to maximize very complex functions, such as those that assign a fitness to a neural network on the basis of its topology and its weights distribution.

Search effectiveness is achieved through an encoding procedure for candidate solutions, i.e., through a genotype coding,

that allows the algorithm to autonomously identify promising search spaces at evolution time. Specifically, different-length genotypes are used in connection with codifying a control parameter, the coding granularity. A cooperative locally optimizing operator adapted from standard linear programming was also used. A derived benefit following the search for short encodings is that with ANNA ELEONORA it is possible to apply the strong specification scheme (cfr. Section II) to bigger networks. Search effectiveness depends in fact on the number of bits of the chromosomes, and ANNA ELEONORA permits squeezing more connections and possibly more neurons in a chromosome of fixed length than those that can be represented by conventional encoding techniques. This therefore allows for the use of a more reliable strong specification scheme for problems for which only the weak one was previously conceivable: we will sketch one such application at the end of this Section.

A parallel version of the algorithm has been presented, suitable for both SIMD and MIMD machines. The two applications described in this paper demonstrate Boolean functions learning and neural controller evolution.

For the former we used Rumelhart's problem test suite, comparing our serial and parallel algorithm with basic backpropagation. The comparison was carried out under comparable limited computational constraints and shows the superior efficiency of the evolutionary approach.

Moreover, it is shown that our system is capable of evolving low-connectivity topologies where useless nodes are also discarded. The implicit search for short effective codes—i.e., for small search spaces—is testified to by the fitness evolutions and results achieved.

For the second application we wanted a neurally controlled simple agent to learn a sequence of behaviors. This entails a very fuzzy fitness function to guide the search, and a high computational cost to assess it. ANNA ELEONORA has been nevertheless able to evolve controllers in a few hundred iterations that could successfully cope with the task proposed. The efficiency shown in this learning is surprising, considering that the experimental setting does not make use of a very detailed fitness function; therefore several important actions, such as turning toward the ball or moving toward the opposite goal when controlling the ball, must be learned without any explicit requirement for them.

Future developments of this research will move along two main axes. The first one regards the development of the animat application: we are currently studying the possibility of evolving teams of players that show co-operative behaviors. The currently available players can in fact play a game trying to prevent of their opponent's goals and to score themselves, but so far they show no co-operation between team-mates. This study will not affect the ANNA ELEONORA algorithm, but its use in the reinforcement learning context of this application. We look forward to obtaining behaviors, such as passing the ball between team-mates or evolution of roles (defense, attack) of different players, without mentioning them explicitly in the fitness function. The evolutionary advantage ensured by co-operating agents should prove sufficient to let a team composed of such players consistently win against teams such as the ones we have now.

The second application we are tackling regards the interpretation of electroencephalographic (EEG) data. We want to identify distinguished time-points, scanning the sequences of thousands of time-tagged numbers provided by the EEG apparatus. This may require networks with hundreds of nodes: we consider this application a benchmark of the scalability of our approach.

ACKNOWLEDGMENT

This research has been partially supported by grants of Progetto Finalizzato Informatica e Calcolo Parallelo of the Italian National Research Council. The author would like to thank Marco Dorigo, Marco Colombetti, Alberto Colorni and Giancarlo Storti-Gajani for useful discussions and comments on a draft version of the paper. Special thanks go to D. B. Fogel for the many useful comments provided.

REFERENCES

[1] T. Bäck, F. Hoffmeister, and H. P. Schwefel, "A survey of evolution strategies," *Proc. of the Fourth Int. Conf. on Genetic Algorithms,* pp. 2–9, 1991.

[2] R. Battiti, "The reactive tabu search for machine learning," *Proc. Of Italian Nat. Wks on Machine Learning,* Milano, 1993, in press.

[3] R. K. Belew, J. McInerney, and N. N. Schraudolph, "Evolving networks: Using the genetic algorithm with connectionist learning," Tech. Rep, CSE90–174, UCSD, 1990.

[4] H. Bersini and G. Seront, "In search of a good crossover between evolution and optimization," in *Parallel Problem Solving from Nature 2,* B. Manderick and R. Männer, Eds. Amsterdam: Elsevier, 1992, pp. 479–488.

[5] A. Bertoni and M. Dorigo, "Implicit parallelism in genetic algorithms," Tech. Rep. 92–012, Dipartimento di Elettronica, Politecnico di Milano, 1992.

[6] V. Caglioti, V. Maniezzo, and D. Sorrenti, "The emergent football team: An experiment on the self-organization of cooperative processes," Tech. Report No. 92–027, Politecnico di Milano, Italy, 1992.

[7] T. P. Caudell and C. P. Dolan, "Parametric connectivity: Training a constrained network using genetic algorithms," in *Proc. of the Third Int. Conf. on Genetic Algorithms and Their Applications,* 1989, pp. 370–374.

[8] J. P. Cohoon, W. N. Martin, and D. Richards, "Genetic algorithms and punctuated equilibria," *Parallel Problem Solving from Nature.* Heidelberg: Springer-Verlag, 1991, pp. 134–144.

[9] J. P. Cohoon, W. N. Martin, and D. Richards, "A multi-population genetic algorithm for solving the K-partition problem on hyper-cubes," in *Proc. of the Fourth Int. Conf. on Genetic Algorithms, 1991,* pp. 244–248.

[10] R. J. Collins and D. R. Jefferson, "An artificial neural network representation for artificial organisms," in *Parallel Problem Solving from Nature,* 1990, pp. 259–263.

[11] R. J. Collins and D. R. Jefferson, "Selection in massively parallel genetic algorithms," in *Proc. of the Fourth Int. Conf. on Genetic Algorithms,* 1991, pp. 249–256.

[12] M. Colombetti and M. Dorigo, "Learning sequential behavior patterns by a hierarchy of classifier systems," ICSI Technical Report, in preparation, 1993.

[13] A. Colorni, M. Dorigo, and V. Maniezzo, "Genetic algorithms and highly constrained problems: The time-table case," in *Parallel Problem Solving from Nature,* H. P. Schwefel and R. Männer, Eds. Heidelberg: Springer-Verlag, 1991, pp. 55–59.

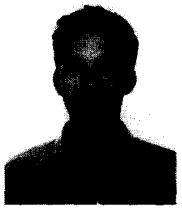
[14] A. Colorni, M. Dorigo, and V. Maniezzo, "Scheduling school teachers by genetic algorithms," in *Combinatorial Optimization, New Frontiers in Theory and Practice,* NATO ASI series, Series F, M. Akgül, H. W. Hamacher, and S. Tfekci, Eds. vol. 82, 1992.

[15] G. B. Dantzig, *Linear Programming and Extensions.* Princeton, NJ: Princeton University Press, 1963.

[16] K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, Univ. of Michigan, 1975.

[17] K. A. De Jong and W. M. Spears, "Using genetic algorithms to solve NP-complete problems," *Proc. 3rd Int. Conf. on Genetic Algorithms and Their Application,* 1989, pp. 124–132.

[18] M. Dorio and V. Maniezzo, "Parallel genetic algorithms: Introduction and overview of current research," in *Parallel Genetic Algorithms: Theory and Applications*, J. Stenders, Ed. Amsterdam: IOS Press, 1992, pp. 5–42.
[19] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biological Cybernetics*, vol. 63, pp. 487–493, 1990.
[20] Fogel *et al.*, in *1992 Aggiornamento Prob. Mut. in EP*
[21] Fogel, *1992 (I Genetici Possono Stagnare Ovunque Nello Spazio di Ricerca).*
[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.
[23] M. Gorges-Schleuter, "ASPARAGOS: An asynchronous parallel genetic optimization strategy," in *Proc. of the Third Int. Conf. on Genetic Algorithms*, 1989, pp. 422–427.
[24] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transaction on Systems, Man and Cybernetics*, SMC–16, no. 1, pp. 122–128, 1986.
[25] P. J. B. Hancock and L. S. Smith, "Gannet: Genetic design of a neural network for face recognition," in *Proc. Parallel Problem Solving from Nature*, H. P. Schwefel and R. Männer, Eds. PPSN–1, Heidelberg: Springer Verlag, 1991, pp. 292–296.
[26] P. J. B. Hancock, "Recombination operators for the design of neural nets by genetic algorithm," in *Parallel Problem Solving from Nature, 2*, R. Männer and B. Manderick, Eds. Amsterdam: Elsevier, 1992, pp. 441–450.
[27] S. A. Harp, T. Samad, and A. Guha, "Toward the genetic synthesis of neural networks," in *Proc. of the Third Int. Conf. on Genetic Algorithms*, 1989, pp. 360–369.
[28] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.
[29] J. H. Holland, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Pattern-Directed Inference Systems*, D. Waterman and F. Hayes-Roth, Eds. Academic Press, 1986.
[30] P. Husbands and F. Mill, "Simulated co-evolution as the mechanism for emergent planning and scheduling," in *Proc. of the fourth Int. Conf. on Genetic Algorithms*, 1991, pp. 264–270.
[31] H. Kitano, "Empirical studies on the speed of convergence of neural network training using genetic algorithms," *Proc. of the Eigth Nat. Conf. on AI (AAAI–90)*, 1990, pp. 789–795.
[32] J. Koza, "Evolution and co-evolution of computer programs to control independently-acting agents," in *From Animals to Animats*, J.-A. Meyer and S. W. Wilson, Eds. Cambridge, MA: MIT Press/Bradford Books, 1991.
[33] B. Kröger, P. Schwenderling, and O. Vornberger, "Parallel genetic packing of rectangles," in *Proc. Parallel Problem Solving from Nature*, H. P. Schwefel and R. Männer, Eds. Heidelberg: Springer Verlag, 1991, pp. 8–11.
[34] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithms," in *Proc. of the Third Int. Conf. on Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 428–433.
[35] V. Maniezzo, "Granularity evolution" in *Proc. of the Fifth Int. Conf. on Genetic Algorithms and Their Applications*, S. Forrest, Ed. San Mateo, CA: Morgan Kaufmann, 1993, p. 644.
[36] V. Maniezzo, "Searching among search spaces: Hastening the genetic evolution of feedforward neural networks," in *Int. Conf. on Neural Networks and Genetic Algorithms, GA-ANN'93*, Heidelberg: Springer-Verlag, 1993, pp. 635–642.
[37] V. Maniezzo, A. Colorni, and M. Dorigo, "ALGODESK: An experimental comparison of eight evolutionary heuristics applied to the AQP problem," *European Journal of Operation Research*, forthcoming.
[38] J. W. L. Merrill and R. F. Port, "Fractally configured neural networks," *Neural Networks*,4, pp. 53–60, 1991.
[39] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks," *Neural Networks*, 4, pp. 53–60, 1991.
[40] D. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. 11th Joint Conf. on Artificial Intelligence, IJCAI-11*, 1989, pp. 762–767.
[41] H. Mühlenbein, M. Georges-Schleuter, and O. Krämer, "Evolution algorithms in ccombinatorial optimization," *Parallel Computing 7*, 1, pp. 65–88, 1988.
[42] H. Mühlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proc. of the Third Int. Conf. on Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 416–421.
[43] H. Mühlenbein, M. Schomisch, and J. Born, "The parallel genetic algorithm as function optimizer," in *Proc. of The Fourth Int. Conf. on Genetic Algorithms*, R. K. Belew, L. B. Booker, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 271–278.
[44] D. Parisi, F. Cecconi, and S. Nolfi, "Econets: neural networks that learn in an environment," *Network*, 1, 1990, pp. 149–168.
[45] C. B. Pettey, M. R. Lutze, and J. J. Grefenstette, "A parallel genetic algorithm," in *Proc. of the Second Int. Conf. on Genetic Algorithms*, J. J. Grefenstette, Ed. Hillsdale, CA: Lawrence Erlbaum, 1987, pp. 155–161.
[46] C. B. Pettey and M. R. Lutze, "A theoretical investigation of a parallel genetic algorithm," in *Proc. of the Third Int. Conf. on Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 398–405.
[47] F. C. Richards, T. P. Meyer, and N. H. Packard, "Extracting cellular automaton rules directly from experimental data." *Physica D*, 45, pp. 189–202, 1990.
[48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, D. E. Rumelhart and J. L. McLelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.
[49] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks," *Physica D*, 42, pp. 244–248, 1990.
[50] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *Proc. Int. Wks. on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, 1992, pp. 1–37.
[51] M. Scholz, "A learning strategy for neural networks based on a modified evolutionary strategy," in *Proc. Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Eds. Heidelberg: Springer Verlag, 1991, pp. 314–318.
[52] S. P. Singh, "Transfer of learning by composing solutions of elemental sequential tasks," *Machine Learning*, 8, pp. 323–339, 1992.
[53] T. Starkweather, D. Whitley, and K. Mathias, "Optimization using distributed genetic algorithms," in *Proc. Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Eds. Heidelberg: Springer Verlag, 1991, pp. 176–186.
[54] G. von Laszewski and H. Mühlenbein, "Partitioning a graph with a parallel genetic algorithm," in *Proc. Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Eds. Heidelberg: Springer Verlag, 1991, pp. 165–169.
[55] D. Whitley and T. Hanson, "Optimizing neural networks using faster, more accurate genetic search," in *Proc. of the Third Int. Conf. on Genetic Algorithms and Their Applications*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 391–396.
[56] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Computing*, 14, 1990, pp. 347–361.
[57] S. W. Wilson, "Classifier systems and the animat problem," *Machine Learning*, 2, pp. 199–228, 1987.
[58] X. Yao, "A review of evolutionary artificial neural networks," Tech. Rep. Commonweath Scientific and Industrial Research Organization, Victoria, Australia, 1992.
[59] L. A. Zadeh, "Fuzzy sets," in *Information and Control*, 8, pp. 33–353, 1965.



Vittorio Maniezzo was born in Ferrara, Italy, in 1962. He received the Laurea (Master of Technology) in electronic engineering in 1986 from the Politecnico di Milano (Milan, Italy), where he also studied for his Ph.D. in Automatic Control and Computer Science Engineering. He currently works at the University of Bologna and is a member of the Politecnico di Milano Artificial Intelligence and Robotics Project. He took part in several CEC ESPRIT and National research projects. His current research interests are in the fields of machine learning (evolutionary techniques for sensorimotor coordination, cognitive modeling) and of combinatorial optimization (evolutionary heuristic algorithms). Dr. Maniezzo is a member of the Italian Association for Artificial Intelligence (AI*IA) and of the IEEE SMC Society.