

Chapter 12

MEMETIC ALGORITHMS

Natalio Krasnogor¹, Alberto Aragón² and Joaquín Pacheco²

(1) *School of Computer Science and I.T. University of Nottingham. England*

(2) *Departamento Economía Aplicada. University of Burgos, Spain*

Abstract: This chapter introduces and analyzes a Memetic algorithm approach for the training of artificial neural networks, more specifically Multilayer Perceptrons. Our Memetic algorithm is proposed as an alternative to gradient search methods, such as *backpropagation*, which have shown limitations when dealing with rugged landscapes with many poor local optima. The aim of our work is to design a training strategy that is able to cope with difficult error manifolds, and to quickly deliver trained neural networks that produce small errors. A method such as the one we proposed might also be used as an “on-line” training strategy.

Key words: Memetic Algorithms, Neural Network, Metaheuristic Algorithms, Evolutionary Algorithms

1. INTRODUCTION TO MEMETIC ALGORITHMS

Memetic Algorithms (MAs) are a class of stochastic global search heuristics in which Evolutionary Algorithms-based approaches are combined with problem-specific solvers. The latter might be implemented as local search heuristics techniques, approximation algorithms or, sometimes, even (partial) exact methods. The hybridisation is meant to either accelerate the discovery of good solutions, for which evolution alone would take too long to discover, or to reach solutions that would otherwise be unreachable by evolution or a local method alone. As the large majority of Memetic Algorithms use heuristic local searches rather than, e.g., approximation or exact methods, in what follows we will focus only on the local search adds for the Evolutionary Algorithm (EA). It is assumed that the evolutionary search provides for a wide exploration of the search space while the local

search can somehow zoom-in on the basin of attraction of promising solutions. MAs have been proven very successful across a wide range of problem domains such as combinatorial optimization (Merz, 2000), optimization of non-stationary functions (Vavak et.al, 1996), multi-objective optimization (Knowles and Corne, 2001), bioinformatics (Krasnogor, 2004), etc.

Memetic algorithms have received various names throughout the literature and scientist not always agree what is and what is not an MA due to the large variety of implementations available. Some of the alternative names used for this search framework are hybrid GAs, Baldwinian EAs, Lamarckian EAs, genetic local search algorithms, and other names are not unheard of. (Moscato 1989) coined the name Memetic Algorithm to cover a wide range of techniques where evolutionary-based search is augmented by the addition of one or more phases of local search.

The natural analogies between human evolution and learning, and EAs and artificial neural networks (ANNs) prompted a great deal of research into the use of MAs to evolve the design of ANNs. Some research concentrated mainly in the automated design of the architecture of interconnection among neurons, which is a combinatorial optimisation problem, and others on the adjustment of the weights associated to the links between neurons, which is a continuous optimisation problem. During the 1980s and early 1990s, ANNs were trained using, for example, back-propagation, conjugate gradients or related methods. At the same time, seminal work by Hinton and Nowlan in the late 80s (Hinton and Nowlan 1987) provided much insights into the interplay between evolution and learning. Other researchers (Whitley et.al. 1994; Mayaley 1996; Turney 1996 and Houck et.al. 1997) followed similar trends, which reinforced the perception that, in order to distil an evolutionary algorithm that could achieve maximum performance on a real world application, much domain knowledge needs to be incorporated. Domain knowledge was oftentimes encoded by means of problem specific local searchers.

Research in Memetic Algorithms has progressed substantially, and several Ph.D. dissertations have been written analysing this search framework and proposing various extensions to it (Hart 1994; Land 1988; Merz 2000; Moscato 2001 and Krasnogor 2002). There are several reasons why it is worthwhile hybridising Evolutionary algorithms with Local Searchers, among them we could mention:

1. Complex problems might be decomposable to some extent and its not far fetched to think that different subproblems could be better solved by different methods.

2. The EA (or the local searcher) can be used as pre/post processors of solutions. One often uses a local searcher in tandem with an evolutionary search as a means of fine-tuning or repairing the best solution(s) produced by the evolutionary algorithm. In turn, one might have very powerful local search methods but would like to introduce diversity (or robustness) into the solutions afforded by the local searcher, in that case one could use an Evolutionary algorithm after several runs of local search have delivered the initial population.
3. (Sub)Problem specific information can be distilled into variation operators, e.g. crossover and mutation, or into local searchers as to effectively bias the search process towards promising regions of the search space.
4. In some cases there are exact/approximate methods for (some of) the subproblems and, when these are available, they can be incorporated into the evolutionary algorithm.
5. (Sub)Problems have constraints associated to solutions and heuristics/local search are used to repair solutions found by the EA. If heuristic/local search strategies in MAs are considered “first class citizens”, rather than simple plug-ins, then a much richer definition of adaptive hybrid metaheuristics is possible: the local search strategies might be generated at the *same time* with the solutions they intend to improve (e.g. see Krasnogor 2004)

However, the reader must note that it is well established that generally good black-box optimisers that excel at solving any problem do not exist. This is why successful EAs are usually found in “hybridized form”. Thus, a “conservation of competence” principle applies: the better one algorithm is solving one specific instance (class) the worst it is solving a different instance (class) (Wolpert and Macready 1997). That is, it cannot be expected that a black-box metaheuristic will suit all problem classes and instances all the time, because, it is theoretically impossible to have both **ready made of-the-shelf general and good** solvers. Memetic algorithms are popular because they are good a *algorithmic template* that aid in the balancing act needed to solve challenging problems by successfully re-using a general, of-the-shelf, solver (e.g. EAs), which can readily incorporate domain specific features in the form of local searchers.

Virtually everybody agrees that for an evolutionary algorithm to succeed in a real-world scenario it must contain sufficient domain knowledge, which is built-in not only in the representation, fitness function and genetic operators used but also in the local search(ers) employed. However, researchers and practitioners sometimes fail to consider in detail some of the main design issues that are pertinent to Memetic algorithms. In the next

section we will assume that the reader is familiar with both Evolutionary Algorithms and Local Search and we will address some important issues that must be kept in mind when designing MAs.

1.1 Evolutionary Algorithms + Local Search = Memetic Algorithms

As suggested above, there are a number of benefits that can be gained by combining the global search of EAs with local search or other methods for improving and refining an individual's solution. However, as there are no free lunches these benefits must be balanced against an increase in the complexity in the design of the algorithm. That is, a careful consideration must be placed on exactly *how* the hybridisation will be done.

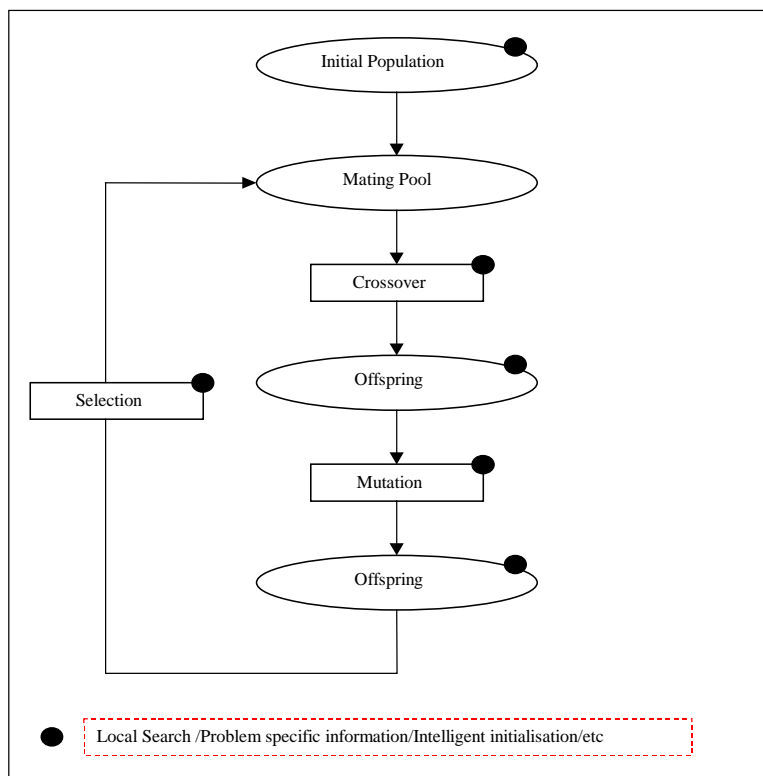


Figure 12-1. A Memetic Algorithm Template

Consider for example the Memetic Algorithm template in *Figure 12-1*. A quick glance at the figure allows us to identify the basic structure of an Evolutionary algorithm for which *hot-spots*, i.e the places where hybridisation could take place, have been identified and marked with red circles. Each of these hot-spots provides opportunity for hybridisation. For example, the initial population could be seeded with solutions arising from sophisticated problem specific heuristics, the crossover (mutation) operator could be enhanced with domain specific and representation specific constrains as to provide better searchability to the EA. Moreover, local search could be applied to any or all of the intermediate sets of solutions (e.g. the offspring set). Hard won empirical evidence points to the fact that the more problem-specific knowledge is incorporated within a Memetic algorithm the better it will perform. However, the most popular form of hybridisation is to apply one or more phases of local search, based on some probability parameter, to individual members of the population in each generation. Some, but not all, of the most important design decisions one needs to make when designing an MA, are concerned with one or more of the following: Lamarckianism vs. Baldwinian effect, preservation of diversity vs. the need to quickly zoom-in on good solutions, the choose of neighbourhoods for the local searcher and use of performance profiles. In what follows we briefly describe each of them.

1.2 Lamarckianism vs. Baldwinian effect

When integrating local search with evolutionary search we are faced with the dilemma of what to do with the improved solution that is produced by the local search. That is, suppose that individual i belongs to the population P in generation t and that the fitness of i is $f(i)$. Furthermore, suppose that the local search produces a new individual i' with $f(i') < f(i)$ for a minimisation problem. The designer of the algorithm must now choose between two alternative options. Either (option 1) he/she replaces i with i' , in which case $P = P - \{i\} + \{i'\}$ and the genetic information in i is lost and replaced with that of i' , or (option 2) the genetic information of i is kept but its fitness altered: $f(i) = f(i')$. The first option is commonly known as Lamarckian learning while the second option is referred to as Baldwinian Learning (Baldwin, 1896). The issue of whether natural evolution was Lamarckian or Baldwinian was hotly debated in the nineteenth century until Baldwin suggested a very plausible mechanism whereby evolutionary progress can be guided towards favorable adaptation without the inheritance of life-time acquired features. Unlike in natural systems, the designer of a Memetic algorithm may want to use either of these adaptation mechanisms. Hinton and Nowlan (1987) showed that the Baldwin effect could be used to improve

the evolution of artificial neural networks, and a number of researchers have studied the relative benefits of Baldwinian versus Lamarckian algorithms, e.g., Whitley et.al. (1994), Mayaley (1996), Turney (1996), Houck et.al. (1997), etc. Most recent work, however, favored either a fully Lamarckian approach, or a stochastic combination of the two methods. It is a priori difficult to decide what method is best, and probably no one is better in all cases. Lamarckianism tends to substantially accelerated the evolutionary process with the caveat that it often results in premature convergence. On the other hand, Baldwinian learning is more unlikely to bring a diversity crisis within the population but it tends to be much slower than Lamarckianism.

1.3 Preservation of Diversity

We have mentioned above the difficulties one may face when using a very aggressive local search method, chief among those difficulties is the need to preserve diversity. Diversity could be lost if, for example, every individual within the population would be taken to a local optimum by means of a vigorous local search (i.e. complete local searches), or, in the case of partial local searches, if the search space under consideration had very wide basins of attraction from which crossover and mutation could not easily escape. Various mechanism have been studied as a way to avoid premature convergence in Memetic Algorithms. For example, if a smart population initialisation is performed, only a small proportion of individuals should be seeded with problem specific information. Alternatively, other authors have resorted to do local search selectively, that is, only a small number of individuals are improved by local search. A very common strategy to deal with diversity preservation has been the introduction of very specific crossover operators (e.g. Freisleben and Merz 1996) that help keeping the genetic variety always above a minimum threshold. The modification of the selection operator as to prevent multiple copies of individuals has also been a preferred strategy (Eshelman 1990). More recently we have introduced three distinct and powerful method which not only help preserve diversity but also enhance the overall algorithmic performance. Multiple local searchers, where each one induces a different search space with distinct local optima thus avoiding local traps have been used in (Krasnogor and Smith 2001; Krasnogor et.al. 2002 and Krasnogor 2004). Also, fuzzy sets and systems have been implemented to explicitly control diversity within the decision rule in the local search stage (eg. Krasnogor and Pelta 2002). Finally, it is also possible to modify the selection operator, or local search acceptance criteria, to use an Adaptive Boltzmann method so as to preserve diversity (eg. Krasnogor and Smith 2000). This is similar to simulated annealing (Kirckpatrick et.al 1983), where worsening

moves can be accepted with nonzero probability to aid escaping from local optima. A promising method that tackles the diversity issue explicitly was proposed in Krasnogor and Smith (2000), where during the local search phase a less-fit neighbor may be accepted with a probability that increases exponentially as the range of fitness values in the population decreases:

$$\text{prob}(\text{accept}) = \begin{cases} 1 & \Leftrightarrow \Delta f > 0 \\ e^{-k \frac{\Delta f}{f_{\max} - f_{\text{avg}}}} & \text{otherwise} \end{cases}$$

where k is a normalisation factor, $\Delta f = f(i') - f(i)$, and we are assuming a maximisation problem. This mechanism seamlessly induces the Memetic Algorithm to oscillate between periods of intense exploitation whenever the spread of fitness in the population, i.e. $f_{\max} - f_{\text{avg}}$, is large, and periods of vigorous exploration when that spread is confined to a narrow interval.

1.4 Which Neighborhood Must the Local Searcher Use?

When the algorithmicist is confronted with the design of a Memetic Algorithm he must carefully consider the topology he/she will give to the space of all feasible solutions. In the case of local search, this is done by defining the structure of the neighborhood of a solution, that is, by specifying which are the feasible solutions that are reachable from a given point in the solution space. If the neighborhood is too small or restrictive it is unlikely that good local optima will be outputted by the local search. On the other hand, if the neighborhood is complete, then from every point in the search space it is possible to build a path to any other point, but in this case, completeness is paid with very long (maybe even exponential) searches. Furthermore, the size of the neighborhoods –although important– is not the only feature to consider. Merz and Freisleben (1999), Kallel et al (2001) and many others have shown through some statistical analysis of fitness landscapes that the choice of move operator can have a significant impact on the efficiency and effectiveness of the local search, and hence of the overall performance of the Memetic Algorithm. Moreover, we have formally shown recently that, in order to reduce the worst-case run times of Evolutionary Algorithms hybridised with local search, it is necessary to choose a local search method whose move operator is different from the ones that the

recombination and mutation operators (Krasnogor, 2002) induce. This formalizes the intuitive point that within an algorithmic template such as a Memetic Algorithm, the recombination, and particularly the mutation phase, have valuable roles in generating points that lie in different basins of attraction with respect to the local search operator. This last result should be of particular interest to the practitioner as it is usually easy to provide the Memetic Algorithm with several distinct local searchers and endow it with the capacity to learn, on-the-fly, which one to use (Krasnogor and Smith 2001; Krasnogor et.al 2002; Krasnogor 2004 and Krasnogor and Gustafson 2004). The use of a set of possible local search strategies is analogous to Dawkin's memes (Dawkins 1976). The extension of this approach to allow the adaptation of the local search "memes" in the form of a coevolving population, and the implications for search is currently under way in different research groups most notably by J.E. Smith at the University of the West of England, Bristol and N. Krasnogor at the University of Nottingham, both in the U.K.

1.5 Use of Performance Profiles

Performance profiles could be use to track the progress of the search vis-à-vis the various algorithmic components of the Memetic Algorithm, that is, one might want to take into consideration the reuse of knowledge gained about the search landscape(s) the algorithm is exploring *throughout* the optimisation process. One possible hybridization that explicitly uses knowledge about previously seen solutions as a mean to guide optimisation is with Tabu search (Glover 1989). For example, the "tabu" list would maintain a set of already visited points, which the algorithm is forbidden to re-visit thus also helping to maintain diversity. In a related way, the Adaptive Boltzmann scheme mentioned above could be extended as to keep track not only of the spread of fitness within the current generation but more generally of the genotypic variety on historical populations.

1.6 Specific Considerations for Continuous Domains

The design issues we mentioned above are mainly concerned with the optimisation of combinatorial, i.e. discrete, problems, such as the optimisation of the architecture of a neural network. As previously mentioned, the automated design of neural networks also involves the optimisation of continuous problems, e.g., weight adjustments. In this case, a person engineering a Memetic Algorithm, must mull over different design factors. As it is well known, optimisation in continuous domains requires that appropriate search scales be identified for both the local and global

search. Moreover, it is often difficult to recognise whether a feasible solution represents or not a local optimum. If gradient information is not available then very long local searches might be needed to ensure convergence with large accuracy. The uncertainty about which local search method is better on which problems is even more acute in continuous optimization. Many different local search methods have been proposed, but as they are general methods it is not clear whether any given local search method is effective for a particular continuous optimisation task. Thus the design of competent MAs for continuous domains can be quite different than for combinatorial problems. A simple example will clarify what we mean: it is common in Memetic Algorithms for combinatorial search (although not always desirable) to apply local search until local optima are identified, however, in general one cannot assume that a local search method will quickly converge towards a good local optimum within a continuous domain. Researchers and practitioners working on continuous domains have found that this is a usual problem when applying derivative-free methods (e.g. the Nelder-Mead simplex), but it may also occur in light of derivative information. Two common strategies are normally used to deal with this difficulty, which essentially account for a careful balancing between global and local search. This balance has been often implemented as either truncated local searches or selective local searchers. The reader is invited to consult with (Rudolph, 1996; Hart et.al., 2003) for further insights on these issues.

Memetic Algorithms are a very powerful algorithmic templates that can be applied to a wide variety of problems and they have been gaining the favor of both researchers and practitioners as they provide a simple formula that allows to combine a robust global search technique, i.e. Evolutionary Algorithms, with powerful, domain-specific local searchers. The reader interested in a more detailed discussion (with a large variety of examples) is referred to Hart, Krasnogor and Smith (2004) and Krasnogor and Smith (2005).

2. NEURAL NETWORK ARCHITECTURE

In what follows we describe a training problem for multi-layer perceptrons and the Memetic algorithm we have used to tackle it. The architecture of the neural network is assumed to be fix once the training starts.

Consider a multiple-layer perceptron neural network with just one hidden layer. Let n denote the number of neurons in the input layer (which is

logically the same as the number of input variables), and m the number of neurons in the hidden layer. Let us assume that the network has only one neuron in the output layer. Each neuron in the hidden layer, as well as in the output layer, has a bias term which functions as a constant. A schematic representation is shown in *Figure 12-2* (from Laguna and Martí 2002).

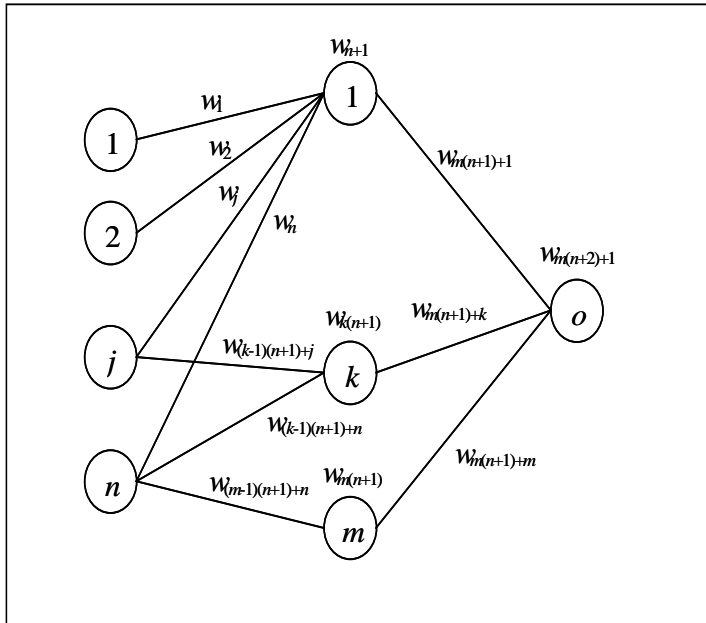


Figure 12-2. Neural network architecture

Weights are sequentially numbered such that the weights that link the input layer with the first neuron of the hidden layer range from w_1 to w_n where their bias term is w_{n+1} .

The network reads an input vector $e = (e_j)$ and the information propagates from one layer to another until output s is produced. Thus, the input information of each neuron, $k = 1, \dots, m$, of the hidden layer is given by

$$z_k = w_{k(n+1)} + \sum_{j=1}^n w_{(k-1)(n+1)+j} e_j$$

and the output neuron receives the following data

$$w_{m(n+2)+1} + \sum_{k=1}^m w_{m(n+1)+k} a_k$$

where a_k are outputs from the hidden neurons.

In our example we use a sigmoid function as a transfer function for the hidden layer and an identity function for the output neuron. Thus,

$$a_k = \frac{1}{1 + e^{-z_k}} y$$

$$s = w_{m(n+2)+1} + \sum_{k=1}^m w_{m(n+1)+k} a_k$$

3. THE TRAINING PROBLEM

3.1 Background

Training consists in adjusting the weight vector $\mathbf{w} = (w_p)_{p=1..m(n+2)+1}$, in the following way: a set of training vectors S is presented, each consisting of an input and output vector; $\forall (e, y) \in S$, e is the input vector and y the expected output. Training consists of determining the weights that minimize the difference between the expected output and those produced by the network o when given the input vectors e . Using the least squares criterion this can be expressed as

$$\min_w E = \frac{1}{|S|} \sum_{(e,y) \in S} (o - y)^2$$

where E is the Mean Squared Error.

The values of each of the variables of the input vectors are normalized between -1 (the lowest value in the training set) and +1 (the highest) for the training stage; the output values are also normalized between -0.8 and +0.8. This normalization is recommended in different works, (Sexton et al., 1998 and 1999).

The algorithms based on gradient descent, such as the *back-propagation method* (developed by Werbos 1974 and later by Parker 1985; LeCun 1986

and Rumelhart et al. 1986), basically use the following formula to update the weights

$$w_{ji} = w_{ji} - \eta \frac{\partial E}{\partial w_{ji}}$$

where η is known as the *learning parameter*. This is an iterative process that terminates when there is no further improvement in E . In fact, one of the limitations of this method is convergence to poor local optima and their dependence on the initial solution. In addition, they are not applicable to network models that use transfer functions for which derivatives cannot be calculated.

The most recent global optimization methods, especially metaheuristic strategies, have generally obtained better results than previous methods based on gradient descent. Thus, traditionally, *Genetic Algorithms* have been highlighted in works such as Montana and Davis (1989), Schaffer et al. (1992), Schaffer (1994) and Dorsey (1994). *Memetic Algorithms* have also yielded interesting results as in the works of Moscato (1993), Topchy et al. (1996), Yao (1993) and Ichimura and Kuriyama (1998). Excellent results have also been obtained in Sexton et al. (1998) and (1999) with *Genetic Algorithms*, *Taboo Search* and *Simulated Annealing*, and in Laguna and Martí (2002), where *Scatter Search* was used.

The aims of our paper is to propose learning strategies that:

1. Can Minimise the mean squared error E for the training set (i.e. learning)
2. Can Minimise the mean square error E for unseen input sets (i.e. generalization)
3. Are fast and robust across a variety of domains and that can be used as on-line learning methodologies.

Points 1 and 2 above refer to the well known trade-off between strong memorisation and good generalization. We are aiming at producing a learning protocol that delivers the correct balance of both. On the other hand, we want our protocols to be fast and robust but at the same time scalable. By scalable we mean that if the training method is given more CPU time to learn then it can minimise even further the error term E . In addition we would like that these new strategies are consistent, i.e., that they are not too sensitive to initial conditions such as the particular training set used or the initial population of the Memetic algorithm.

3.2 The Memetic algorithm

We describe next a memetic algorithm for the training of neural networks together with its main characteristics. One aspect to take into account is that due to the type of transfer functions used, these methods only seek weight values in the hidden layer: those in the output layer are obtained by the least squares method, ensuring good results, (this was suggested by Sexton et al., 1998). In Section 4 we describe an improvement method used by the Memetic algorithm. The Memetic algorithm is described in Section 5.

4. IMPROVEMENT BY LOCAL SEARCH

In this section we analyze the probabilities of improving a given solution by using a simple hillclimbing algorithm. The hillclimber algorithm starts from an initial solution (i.e. a weights vector w), samples a small set of weights vectors around w , and selects as its next vector the one that minimises the error E . Specifically, let w be any solution with w_i components and $radius$ a sufficiently small real number; the following procedures are defined:

Change_Weight Procedure (w , $radius$, w')

Generate a solution w' whose components w'_i take random values in the interval $(w_i - radius, w_i + radius)$

Likewise, given that w_0 is any solution of the parameters $n_neighbours \in N$ and $radius \in R$; the following is defined:

Best_Neighbour Procedure (w_0 , $radius$, $n_neighbours$, w')

$min = \infty$

For $i = 1$ up to $n_neighbours$ do:

Change_Weight (w , $radius$, w'')

If $E(w'') < min$ do: $w' = w''$ and $min = E(w'')$

Finally, given that w is an initial solution we define

Best_Environment Procedure (w , $radius$, $n_neighbours$, w')

Repeat

Previous_value = $E(w)$

Execute Best_Neighbour (w , $radius$, $n_neighbours$, w')

If $E(w') < E(w)$ do $w = w'$

Until $E(w') \geq Previous_value$

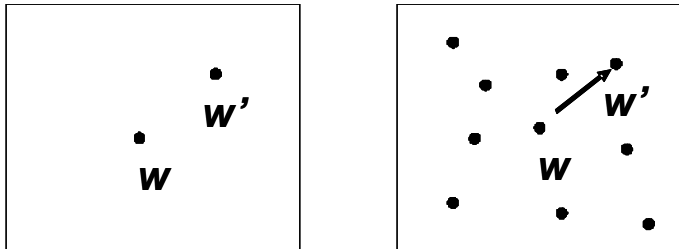


Figure 12-3. The Change_Weight and Improve_Neighbour procedures

The latter procedure is thus an improvement method: in each iteration a set of points is generated by changes around the current solution. Then, the best solution is determined and if this one improves the current solution the change is implemented. The process stops when, from among the solutions generated in one iteration, none is found which improves the current one.

4.1 Analysis of the $n_neighbours$ and the radius parameters

To study the influence of the number of neighbour solutions generated in each pass, a set of 50 training vectors with 2 input variables x_1 and x_2 is generated. The input values are integers generated within $[-100, +100]$ for x_1 and within $[-10, +10]$ for x_2 . The output values y are given by $y = x_1 + x_2$. In addition, a second set of training vectors is generated with the same input vectors and $y = x_1 \cdot x_2$ as the output vectors. Similarly, 10 initial solutions (hidden layer weights) are generated for each problem. Their components are random values between -5 and $+5$. With each solution as the starting point the *Best_Environment* procedure described earlier was executed. Different values of $n_neighbours$ were used: 100, 200 and 500. In all cases the value of *radius* takes 0.1. Table 12-1 shows the following results for each problem: mean computational time in seconds, number of solutions generated during each process, and statistical values obtained for E (minimum, mean and maximum). All tests were done on a 600 Mhz Pentium III (DUAL processor). The BORLAND PASCAL (7.0) and BORLAND DELPHI (3.0) compilers were used.

Table 12-1. hillclimber analysis

Function		Initial solution	<i>n_neighbours</i>		
			100	200	500
$y = x_1 + x_2$	Min	1.25E-01	2.64E-06	3.74E-06	1.55E-06
	<i>E</i> Mean	8.42E-01	4.46E-04	2.74E-04	9.81E-05
	Max	2.55E+00	1.33E-03	1.33E-03	8.81E-03
	Comp. Time		7.1283	13.2654	34.3502
	Evaluations		3870	7200	18650
$Y = x_1 \cdot x_2$	Min	8.34E+00	3.10E-02	1.37E-02	6.12E-03
	<i>E</i> Mean	2.74E+01	1.97E-01	1.19E-01	5.27E-02
	Max	4.72E+01	5.91E-01	3.82E-01	2.34E-01
	Comp. Time		9.5344	19.8031	53.9314
	Evaluations		5140	10680	29100

As expected, a larger number of generated neighbours improves the final result, although at the cost of greater computational time.

To analyse the influence of the radius used to generate values around the current solution, the same tests were carried out i.e., the same sets of training vectors and 10 initial solutions generated for each of them. The following values were used for the parameter *radius*: 1, 0.1 and 0.01. To avoid excessive computational time *n_neighbours* was set to 100. The results are shown below

Table 12-2. Radius analysis

Function		Initial solution	<i>Radius</i>		
			1	0.1	0.01
$y = x_1 + x_2$	Min	2.15E-01	5.33E-05	9.62E-05	7.13E-08
	<i>E</i> Mean	8.14E-01	6.66E-03	1.01E-03	2.10E-04
	Max	1.29E+00	3.19E-02	2.20E-03	9.99E-04
	Comp. Time		0.8517	5.3984	55.9828
	Evaluations		440	2790	28940
$y = x_1 \cdot x_2$	Min	7.23E+00	1.82E-01	1.15E-02	1.12E-03
	<i>E</i> Mean	1.68E+01	8.62E-01	1.34E+00	2.93E-01
	Max	2.56E+01	2.30E+00	1.07E+01	1.55E+00
	Comp. Time		1.3703	7.464	90.8813
	Evaluations		690	3760	45780

Decreasing the *radius* greatly decreases the error, but at the cost of increasing the computational time. Generally, high values of *radius* entail a rapid decrease in computational time whereas low values lead to slower behaviour but yield better solutions in the long run. In addition, it is clear that the effect of the parameter *radius* is more significant than that of *n_neighbours*.

4.2 Improvement with a variable radius

As mentioned in previous sections, our method attempts to combine speed and 'depth' (i.e. quality of the final solution). This method, called *Variable_Radius Improvement*, consists in modifying the algorithm such that the value of *radius* changes depending on whether improvements are found or not: if the hillclimber finds a better solution, the new solution is accepted and the value of the *radius* with which the hillclimber searched is increased. On the other hand, if no improvement is found the value of the radius is decreased and takes effect in the following iterations. The algorithm ends when a given number of consecutive iterations have taken place without any improvement. That idea have been taken from Vavak, Fogarty and Jukes (1996), Krasnogor and Smith (2000) and Krasnogor and Pelta (2002).

With w as an initial solution, the real parameters *radius0* (initial value of *radius*) and *alpha*, and the integer parameters *n_neighbours* and *max_int*, the proposed algorithms can be written in pseudocode as follows:

```

Improve_variable_radius Procedure (w, radius0, alpha, n_neighbours,
                                   max_int)
  Do radius = radius0, n_int = 0;
  Repeat
    Execute Best_Neighbour (w, radius, n_neighbours, w')
    If  $E(w') < E(w)$  do:  $w = w'$ ,  $radius = radius * alpha$  and  $n\_int = 0$ 
    Otherwise:  $radius = radius / alpha$  and  $n\_int = n\_int + 1$ 
  Until  $n\_int = max\_int$ 

```

To compare how well this procedure works, the same set of training vectors have been used as in subsection 3.1. Similarly, 10 initial solutions were generated for each of these sets. The parameters take the following values: *n_neighbours* = 10 and 25, *alpha* = 10, *radius0* = 1 and *max_int* = 3. The results were as follows:

Table 12-3. Comparing *n_neighbours* in *Improve_variable_radius*

Function		Initial solution	<i>n_neighbours</i>	
			10	25
$y = x_1 + x_2$	Min	4.22E-02	5.83E-06	3.44E-07
	<i>E</i> Mean	5.31E-01	2.41E-04	1.05E-04
	Max	1.73E+00	1.15E-03	4.84E-04
$y = x_1 \cdot x_2$	Comp. Time		3.4296	3.3905
	N. Evaluations		1850	1832
	Min	6.02E+00	3.23E-04	9.60E-05
<i>E</i>	Mean	1.52E+01	1.19E-01	2.71E-03
	Max	2.28E+01	9.89E-01	1.55E-02

Function	Initial solution	<i>n_neighbours</i>	
		10	25
Comp. Time		27.0063	253.6608
N. Evaluations		14651	137615

A comparison of *Tables 12-1, 12-2 and 12-3* reveals very interesting results: good solutions are obtained within reasonable computational times. Therefore, in improvement by local search methods, an appropriate choice of radius, and the possibility of locally changing the radius is more decisive than increasing the number of neighbours.

In any case, the improvement methods proposed in this section are at least 'universal', since they can easily be applied to any problem and are not specially made for the network model. For example, no derivability or continuity are demanded from the transfer functions, but this is not the case with gradient-based methods.

5. MEMETIC ALGORITHM

In the previous section we described and tested the hill-climber method with which we hybridised a Genetic algorithm, thus obtaining a Memetic approach. The pseudocode in Figure 12.4 shows how the genetic operators, local search method and selection procedure are “tied-up” within the Memetic algorithm.

The improvement procedure is *Variable_Radius_Improvement* with the same values as the parameters $radius0 = 0.1$, $alpha = 1$, $n_neighbours = 2$, $max_int=1$. The *fitness* of each individual is given by the value of the corresponding objective function E . The probability of selecting a given solution w is proportional to $E_{max} - E(w)$, where E_{max} is the maximum value of $E(w)$ within the population.

The crossover operator is a ‘one-point crossover’, that is, if w and w' are two parent solutions,

$$w = (w_1, w_2, \dots, w_{m(n+1)}) \quad y \quad w' = (w'_1, w'_2, \dots, w'_{m(n+1)})$$

a ‘crossover point’ is randomly generated between 1 and $m(n+1)-1$, (crossover_point), in such a way that the offspring solutions are defined as

$$\begin{aligned} w^* &= (w_1, w_2, \dots, w_{point_crossover}, w'_{point_crossover+1}, \dots, w'_{m(n+1)}) \\ w^{**} &= (w'_1, w'_2, \dots, w'_{point_crossover}, w_{point_crossover+1}, \dots, w_{m(n+1)}) \end{aligned}$$

The weights of each new child solution can change or *mutate* with a low probability, p_mut . To decide whether a given weight changes or not, a

random value is uniformly generated in (0, 1); if this value is less than p_mut the change is implemented. This consists in assigning to it a new random point in the interval (-0.5, +0.5), (*Mutation*). This mutation process aims at adding diversity to the process and prevents it from being locked in a poor local minimum.

Memetic Procedure:

Generate an initial population of solutions

Improve them with a pre-established method

Repeat

- *Create a mating pool by sampling the population with fitness proportional probability.*
- *Crossover: Randomly pair members (i.e. parents) from the mating pool and apply the crossover operator producing 2 offspring from each pair. Newly created offspring are added to the offspring set.*
- *Mutation: apply with low probability the mutation operator to the offspring set.*
- *Perform local search with the variable radius hill-climber starting from each member of the offspring set*
- *Replace the worst solutions in the population with the new offspring*
- *Implement the mutation parameter*

Until reaching some stopping criterion

Figure 12-4. Memetic Algorithm's pseudocode

A relatively original feature of our procedure is that the value of parameter p_mut is not fixed but varies between two pre-set values p_mut_min and p_mut_max . The aim is as follows: when a new and better solution is found the mutation probability decreases, and thus the search intensifies in such a region. As generations pass (iterations) without improvement, the mutation probability increases to intensify the search in other regions. Three rules govern the adaptation of the mutation rate:

- If in the current iteration a new and better solution is found do
 $p_mut = p_mut_min$
- In the opposite case do $p_mut = p_mut + p_mut_inc$
- If $p_mut > p_mut_max$ do $p_mut = p_mut_max$.

The values p_mut_min and p_mut_max represent the maximum and minimum mutation probabilities.

The following values are used: $p_mut_min = 0,05$, $p_mut_max = 0,1$ and $p_mut_inc = 0,0005$. In addition, the number of population elements is 100, and the number of selected parents (size of *mating pool*) is 20. In each iteration the worst 20 members in population are replaced for the new 20 offsprings.

6. COMPUTATIONAL RESULTS

In order to test the quality of the Memetic algorithm, and compare it with other relevant methods found in the literature, we carried out the various tests. We have considered 6 test-beds:

$$1) y = x_1 + x_2$$

$$2) y = x_1 x_2$$

$$3) y = \frac{x_1}{1 + |x_2|}$$

$$4) y = x_1^2 + x_2^3$$

$$5) y = x_1^3 + x_2^2$$

$$6) y_t = y_{t-1} + 10.5 \left(\frac{0.2 y_{t-5}}{1 + (y_{t-5})^{10}} - 0.1 y_{t-1} \right)$$

Note that these are continuous and derivable functions except for the third one. The sixth function is a discrete version of the well-known MacKey-Glass equation that has been frequently used in neural network research (Gallant and White 1992; Goffe et al. 1994).

For each of the first five functions a training set with 50 vectors is considered: the input values are integers generated within $[-100, +100]$ for x_1 and within $[-10, +10]$ for x_2 ; the output value y is given by the corresponding function. The same input vectors were used in the five cases. For the sixth function the five nodes or input variables were used although, obviously, only three were needed. However, their inclusion for checking the network's capacity to ignore unnecessary information is of interest. In this case, the training vectors were recursively generated from the values (1, 6, 0, 0, 0, 0).

The following methods were compared:

- BP: Backpropagation algorithm used in *Neural Works Profession II/Plus* from *NeuralWare*. This freeware package was rated the best among several commercial programs in Sexton et al. (1998)
- TS: *Tabu Search* algorithm proposed in Sexton et al. (1999).
- SA: *Simulated Annealing* algorithm proposed in Sexton et al. (1999).
- GA: The Genetic Algorithm proposed in Sexton et al. (1999).
- SS: *Scatter Search Algorithm* proposed by Laguna and Martí (2002).
- ME: *Memetic Algorithm* proposed in Section 5.

The tests were repeated ten times for each method and function. *Table 12-4* shows the best results obtained by the different algorithms for each of these problems where we identified in bold the best result. The results of each method are taken from the references mentioned where the same functions and neural network architectures were used.

Table 12-4. Best results obtained for BP, SA, GA, TS and SS

Function	BP	SA	GA	TS	SS	ME
1	5.23E-01	1.05E-05	4.16E-07	1.42E-06	8.93E-08	6.32E-14
2	1.10E+01	3.17E-02	1.27E-02	8.38E-02	5.97E-03	6.56E-05
3	8.43E+00	1.76E+00	1.82E-01	2.45E-01	2.96E-03	4.62E-04
4	1.88E+02	4.84E-01	4.09E-02	4.32E-01	2.31E+00	1.93E-03
5	8.57E+03	4.39E-01	3.06E-03	2.56E-02	2.76E+02	4.58E-01
6	1.55E-01	1.02E-02	2.53E-02	5.35E-02	3.34E-03	1.51E-04

According to Laguna and Martí (2002) and Sexton (1998) and (1999), SS limits the number of evaluations of the objective function to 50,000; BP carries out 4.18 million, SA between 112,501 and 12.6 million, GA 100,000, TS between 190,021 and 928,061, and ME 100,000. Our ME achieves the best results in all functions except in the 5th function, where it is outperformed by SA, GA and TS.

Table 12-5 shows the evolution (value of the best solution) of our *Memetic* algorithm according to the number of evaluations (25,000, 50,000, 75,000 and 100,000): Note that it demonstrates its capacity to evolve and improve regarding all functions except for the function 6. Also, note from table 12-4 and 12-5 that with 50,000 evaluations (and even with 25,000) our *Memetic* performs better than other strategies (except in function 5).

Table 12-5. Evolution of the best results of ME

Function	25,000	50,000	75,000	100,000
1	1.86E-13	7.13E-14	6.45E-14	6.32E-14
2	1.67E-03	2.10E-04	1.20E-04	6.56E-05
3	2.30E-03	7.05E-04	6.20E-04	4.62E-04
4	6.02E-02	5.37E-03	4.24E-03	1.93E-03
5	3.43E+00	1.12E+00	4.92E-01	4.58E-01
6	1.51E-04	1.51E-04	1.51E-04	1.51E-04

Table 12-6 presents the mean and maximum values of the final results obtained. Even the worst results of our ME are competitive when compared to the best results of other strategies. This is an indicator of the consistency of our method.

Table 12-6. Mean and maximum values of E
($n_{evaluations} = 100,000$)

Function	ME	
1	Mean	9.70E-13
	Maximum	2.28E-12
2	Mean	2.15E-03
	Maximum	1.09E-02
3	Mean	1.35E-02
	Maximum	5.85E-02
4	Mean	2.17E-02
	Maximum	9.12E-02
5	Mean	3.96E+00
	Maximum	1.35E+01
6	Mean	7.84E-04
	Maximum	1.10E-03

Subsequently, a test set of 150 vectors was generated where the input values were generated with the same distribution as the training set. This test set was used to examine the capacity of the network to predict values of y when the values of x are not in the training set. Table 12-7 shows the results achieved in the test set for the weights obtained by the algorithms. The results obtained with our *Memetic* algorithm for the test set are not significantly worse than for the training set.

Table 12-7. Best value of E obtained for BP, SA, GA, TS and SS for the test set

Function	BP	SA	GA	TS	SS	ME
1	1.76E+00	1.56E-05	3.68E-07	2.79E-06	9.37E-07	1.13E-12
2	9.11E+01	1.85E-01	1.72E-02	1.96E-01	2.13E-01	1.45E-04
3	2.34E+01	5.53+00	1.38E+00	1.69E+00	1.55E+00	9.89E-02
4	4.31E+02	2.73E+00	4.53E-02	1.15E+00	9.26E+00	6.52E-03
5	5.28E+04	1.36E+00	2.02E-02	5.68E-02	5.98E+03	2.43E+00
6	1.95E-01	2.69E-01	3.40E-02	6.75E-02	2.90E-01	8.64E-03

7. FINAL CONCLUSIONS AND CONSIDERATIONS

In this chapter we proposed a new Memetic algorithm for the training of artificial neural networks. This algorithm combines traditional genetic operators with an improvement method – a hillclimber- specially designed for this problem. In our approach we rely heavily on self-adapting strategies of the search parameters. For example, the mutation rate of the underlying genetic algorithm is adapted on-line accordingly to the state of the search. In addition, the hill-climber has a variable radius search strategy that either diversifies or intensifies the search accordingly to which region of the search space the algorithm is currently sampling. Both these self-adapting mechanism are derivative independent and as such could be use in many other neural network topologies.

The following features of our approach are worth highlighting:

- It generates good solutions in very short time (in this case, the number of evaluations), which makes it suitable for online learning procedures.
- It outperforms or equals the results obtained with other strategies proposed in recent works.
- It shows the capacity to evolve and to not 'stall' when more computational time is available.
- The results obtained in the test set do not undergo significant deterioration in relation to the training set, and it continues to be better than the other strategies in almost all cases, thus showing it generalization capabilities.

Thus, the new algorithm fulfils the aims set out in Section 2. From a more general point of view we can say that the method proposed -as well as other metaheuristic methods- are often more convenient than traditional gradient-based methods for a number of reasons: first, they are not restricted transfer functions for which gradient information is available, second, metaheuristic methods such as our Memetic algorithm can also handle

continuous cases, third, population based training methods are more unlikely to get stuck in poor local optima like some gradient based methods.

ACKNOWLEDGEMENTS

The authors are grateful for financial support from the Spanish Ministry of Education and Science (National Plan of R&D - Project SEJ2005-08923/ECON)

REFERENCES

- Aragón, A., 2002, *Métodos Evolutivos para el Aprendizaje de Redes Neuronales*, PhD thesis, University of Burgos, Spain, available on:
www2.ubu.es/econapli/profesores/jpacheco/Investigacion/Tesis/Alberto/Tesis_Alberto
- Baldwin, J., 1896, A new factor in evolution, *American Naturalist*, 30.
- Dawkins, R., 1976, *The Selfish Gene*, Oxford University Press, Oxford, UK.
- Dorsey, R. E., Johnson, J. D., and Mayer, W. J., 1994, A genetic algorithm for the training of feedforward neural networks, in: *Advances in Artificial Intelligence in Economics, Finance and Management*, vol. 1, J.D. Johnson, A.B. Whinston, eds., JAI Press, Greenwich, CT, pp. 93–111.
- Eshelman, L., 1990, The {CHC} adaptive search algorithm: how to have safe search when engaging in non-traditional genetic recombination, in: *Foundations of Genetic Algorithms*, G. Rawlins, ed., Morgan Kaufmann, San Francisco, pp. 263–283.
- Friesleben, B., and Merz, P., 1996, A genetic local search algorithm for solving the symmetric and asymmetric travelling salesman problem, in: *Proceedings of the {IEEE} Conference on Evolutionary Computation*, IEEE Press, Piscataway, NJ.
- Gallant, R. A., and White, H., 1992, On learning the derivatives of an unknown mapping with multilayer feedforward networks, in: *Artificial Neural Networks*, Blackwell, Cambridge, MA, pp. 206–223.
- Glover, F., 1989, Tabu search, *ORSA Journal on Computing*, 1: 190–206.
- Goffe, W. L., Ferrier, G. D., and Rogers, J., 1994, Global optimization of statistical functions with simulated annealing, *Journal of Econometrics*, 60: 65–99.
- Hart, W., 1994, *Adaptive Global Optimization with Local Search*, PhD thesis, University of California, San Diego, USA.

- Hart, W., De Laurentis, J., and Ferguson, L., 2003, On the convergence of an implicitly self-adaptive evolutionary algorithm on one-dimensional unimodal problems, *IEEE Trans Evolutionary Computation* (to appear).
- Hart, W. E., Krasnogor, N., and Smith, J. E., 2004, *Recent Advances in Memetic Algorithms*, Series Studies in Fuzziness and Soft Computing, Springer-Verlag.
- Houck, C., Joines, J., Kay, M., and Wilson, J., 1997, Empirical investigation of the benefits of partial Lamarckianism, *Evolutionary Computation*, 5: 31–60.
- Hinton, G., and Nowland, S., 1987, How learning can guide evolution, *Complex Systems*, 1: 495–502.
- Ichimura, T., and Kuriyama, Y., 1998, Learning of neural networks with parallel hybrid GA using a royal road function, in: *IEEE International Joint Conference on Neural Networks*, IEEE, New York, NY, vol. 2, pp. 1131–1136.
- Kallel, L., Naudts, B., and Reeves, C., 2001, Properties of fitness functions and search landscapes, in *Theoretical Aspects of Evolutionary Computing*, L. Kallel, B. Naudts and A. Rogers, eds., Springer, Berlin, Heidelberg, New York..
- Kirkpatrick, S., Gelatt, C., and Vecchi, M., 1983, Optimization by simulated annealing, *Science*, 220: 671–680.
- Knowles, J., and Corne, D., 2001, A comparative assessment of memetic, evolutionary and constructive algorithms for the multi-objective d-msat problem, in: *2001 Genetic and Evolutionary Computation Workshop Proceeding*.
- Krasnogor, N., and Smith, J., 2000, A memetic algorithm with self-adaptive local search: {TSP} as a case study, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee and H.G. Beyer, eds., Morgan Kaufmann, San Francisco, pp. 987–994.
- Krasnogor, N., and Smith, J., 2001, Emergence of profitable search strategies based on a simple inheritance mechanism, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, L. Spector, E. Goodman, A. Wu, W. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, eds., Morgan Kaufmann, San Francisco, pp. 432–439.
- Krasnogor, N., Blackbourne, B., Burke, E., and Hirst, J., 2002, Multimeme algorithms for protein structure prediction, in: *Proceedings of the Parallel Problem Solving From Nature*, Lecture Notes in Computer Science, Springer, pp. 769–778.
- Krasnogor, N., 2002, *Studies in the Theory and Design Space of Memetic Algorithms*, PhD thesis, University of the West of England, Bristol, U.K.
- Krasnogor, N. and Pelta, D., 2002, Fuzzy memes in multimeme algorithms: a fuzzy-evolutionary hybrid, in: *Fuzzy Sets based Heuristics for Optimization*, J. Verdegay, ed., Springer.

- Krasnogor, N., 2004, Self-generating metaheuristics in bioinformatics: The protein structure comparison case, in: *Genetic Programming and Evolvable Machines*, Kluwer academic Publishers, vol. 5, pp. 181–201.
- Krasnogor, N., and Gustafson, S., 2004, A study on the use of "self-generation" in memetic algorithms, *Natural Computing*, 3: 53–76.
- Krasnogor, N., and Smith, J., 2005, A tutorial for competent memetic algorithms: Model, taxonomy and design issues, *IEEE Transactions on Evolutionary Computation*, (in press)
- Laguna, M., and Martí, R., 2002, Neural network prediction in a system for optimizing simulations, *IEE Transactions* 34 (3): 273–282.
- Land, M., 1998, *Evolutionary Algorithms with Local Search for Combinatorial Optimization*, PhD thesis, University of California, San Diego, USA.
- Lecun, Y., 1986, Learning process in an asymmetric threshold network, in: *Disordered Systems and Biological Organization*. Springer, Berlin, pp. 233–240.
- Mayaley, G., 1996, Landscapes, learning costs and genetic assimilation, *Evolutionary Computation*, (4): 213–234.
- Merz, P., and Freisleben, B., 1999, Fitness landscapes and memetic algorithm design, in: *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover, eds., McGraw Hill, London.
- Merz, P., 2000, *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*, PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany.
- Montana, D. J., and Davis, L., 1989, Training feedforward neural networks using genetic algorithms, in: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 379–384.
- Moscato, P., 1989, On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, *Caltech Concurrent Computation Program*, C3P Report 826.
- Moscato, P., 2001, *Problemas de Otimização NP, Aproximabilidade e Computação Evolutiva: Da Prática a Teoria*, PhD thesis, Universidade Estadual de Campinas, Brasil.
- Parker, D., 1985, Learning logic, Technical Report TR-87. *Center for Computational Research in Economics and Management Science*, MIT, Cambridge. MA.
- Rudolph, G., 1996, Convergence of evolutionary algorithms in general search spaces, in *Proceedings of the International Congress of Evolutionary Computation*, pp 50–54.
- Rumelhart, D., Hinton, G., and Williams, R., 1986, Learning internal representations by error propagation, in: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol.1. D. Rumelhart and J. McClelland, eds., MIT Press. Cambridge. MA.

- Schaffer, J. D., 1994, Combinations of genetic algorithms with neural networks or fuzzy systems, in: *Computational Intelligence: Imitating Life*, J. M. Zurada, R. J. Marks and C. J. Robinson, eds., IEEE Press, pp. 371–382.
- Schaffer, J. D., Whitley, D., and Eshelman, L. J., 1992, Combinations of genetic algorithms and neural networks: A survey of the state of the art, in: *COGANN-92 Combinations of Genetic Algorithms and Neural Networks*, IEEE Computer Society Press, Los Alamitos, CA, pp. 1-37.
- Sexton, R. S., Alidaee, B., Dorsey, R. E., and Johnson, J. D., 1998, Global optimization for artificial neural networks: A tabu search application, *European Journal of Operational Research*, 106: 570–584.
- Sexton, R. S., Dorsey, R. E., and Johnson, J. D., 1999, Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing, *European Journal of Operational Research*, 114: 589–601.
- Topchy, A. P., Lebedko, O. A., and Miagkikh, V. V., 1996, Fast learning in multilayered networks by means of hybrid evolutionary and gradient algorithms, in *Proceedings of International Conference on Evolutionary Computation and its Applications*, pp. 390-398.
- Turney, P., 1996, How to shift bias: lessons from the Baldwin effect, *Evolutionary Computation*, 4: 271–295.
- Vavak, F., Fogarty, T., and Jukes, K., 1996, A genetic algorithm with variable range of local search for tracking changing environments, in: *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science 1141, H. M. Voigt, W. Ebeling, I. Rechenberg and H. P. Schwefel, eds., Springer.
- Werbos, P., 1974, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis., Harvard, Cambridge.
- Whitley, L., Gordon, S., and Mathias, K., 1994, Lamarckian evolution, the Baldwin effect, and function optimization, in *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science 866, Y. Davidor, H. P. Schwefel, and R. Manner, eds., Springer.
- Wolpert, D., and Macready, W., 1997, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation*, 1: 67–82.
- Yao, X., 1993, Evolutionary artificial neural networks, *Int. Journal of Neural Systems*, 4 (3), 203-222.