Training feed-forward neural networks with ant colony optimization: An application to pattern classification*

Christian Blum
ALBCOM, LSI
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: cblum@lsi.upc.edu

Krzysztof Socha
IRIDIA
Université Libre de Bruxelles
Brussels, Belgium
Email: ksocha@ulb.ac.be

Abstract

Ant colony optimization (ACO) is an optimization technique that was inspired by the foraging behaviour of real ant colonies. Originally, the method was introduced for the application to discrete optimization problems. Recent research efforts led to the development of algorithms for the application to continuous optimization problems. In this work we extend and apply one of the most succesful variants for the training of feed-forward neural networks. For evaluating our algorithm we apply it to patter classification problems from the medical field. The results show that our algorithm is comparable to specialized algorithms for neural network training, and that it has advantages over other general purpose optimizers.

1 Introduction

Pattern classification is an important real-world problem. In the medical field, for example, pattern classification problems arise when physicians are interested in reliable classifiers for diseases based on a number of measurements. Feed-forward neural networks (NNs) are commonly used systems for the task of pattern classification [4], but require prior configuration. Generally the configuration problem consists hereby of two parts: First, the structure of the feed-forward NN has to be determined. Second, the numerical weights of the neuron connections have to be determined such that the resulting classifier is as correct as possible. In this work we focus only on the second part, namely the optimization of the connection weights. We adopt the NN structures from earlier works on the same subject.

Ant colony optimization (ACO) [6] is an optimization technique that was introduced for the application to discrete optimization problem in the early 90's by M. Dorigo and colleagues. The origins of ACO are in a field called swarm intelligence (SI), which studies the use of certain properties of social insects, flocks of birds, or fish schools, for tasks such as optimization. The inspiring source of ACO is the foraging behaviour of real ant colonies, which enables them to find shortest paths between their nest and food sources. The shortest path finding capabilities of real ant colonies are exploited in artificial ant colonies for solving optimization problems.

While ACO algorithms were originally introduced to solve discrete optimization (i.e., combinatorial) problems, their adaptation to solve continuous optimization problems enjoys an increasing attention. Early applications of the ants metaphor to continuous optimization include algorithms such as Continuous ACO (CACO) [2], the API algorithm [10], and Continuous Interacting Ant Colony (CIAC) [7]. However, all these approaches follow rather loosely the original ACO framework. The latest approach, which is at the same time the approach that is closest to the spirit of ACO for combinatorial problems, was proposed in [13]. In this work we extend this approach, and apply it to the problem of optimizing the weights of feed-forward NNs for the task of pattern classification.

The outline of our work is as follows. In Section 2 we shortly present the structure of feed-forward NNs for the purpose of pattern classification. Then, in Section 3 we present the ACO algorithm, while in Section 4 we compare our algorithm to methods specialized for feed-forward NN training, as well as to a genetic algorithm. Finally, in Section 5 we offer a conclusion and a glimpse of future work.

*This work was supported by the Spanish CICYT project TRACER (grant TIC-2002-04498-C05-03), and by the "Juan de la Cierva" program of the Spanish Ministry of Science and Technology of which Christian Blum is a post-doctoral research fellow. This work was also partially supported by the ANTS project, an Action de Recherche Concertée funded by the Scientific Research Directorate of the French Community of Belgium.

Proceedings of the Fifth International Conference on Hybrid Intelligent Systems (HIS'05)
0-7695-2457-5/05 $20.00 © 2005 IEEE

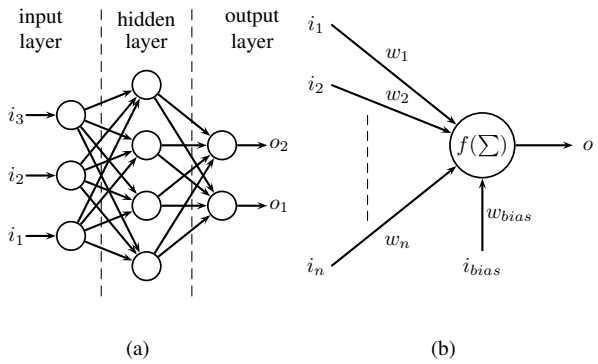


Figure 1. (a) shows a feed-forward NN with one hidden layer of neurons. (b) shows one single neuron (from either the hidden layer, or the output layer).

2 Feed-forward neural networks for pattern classification

A dataset for pattern classification consists of a number of patterns together with their correct classification. Each pattern consists of a number of measurements (i.e., numerical values). The goal consists in generating a classifier that takes the measurements of a pattern as input, and provides its correct classification as output. A popular type of classifier are feed-forward neural networks (NNs).

A feed-forward NN consists of an input layer of neurons, an arbitrary number of hidden layers, and an output layer (for an example, see Figure 1). Feed-forward NNs for pattern classification purposes consist of as many input neurons as the patterns of the data set have measurements, i.e., for each measurement there exists exactly one input neuron. The output layer consists of as many neurons as the data set has classes, i.e., if the patterns of a medical data set belong to either the class normal or to the class pathological, the output layer consists of two neurons. Note that each neuron of a certain layer is connected to each neuron of the next layer. A neuron receives inputs (i.e., signals i_l, weighted by weights w_l) from each neuron of the previous layer. Additionally, it receives a so-called bias input i_{bias} with weight w_{bias}. The transfer function $f(\sum)$ of a neuron transforms the sum of all the weighted inputs into an output signal, which servers as input for all the neurons of the following layer. Input signals, output signals, biases and weights are real values. Given the weights of all the neuron connections, in order to classify a pattern, one provides its measurements as input to the input neurons, propagates the output signals from layer to layer until the output signals of the output neurons are obtained. Each output neuron is identified with one of the possible classes. The output neuron that produces the highest output signal classifies the respective pattern.

The process of generating a NN classifier consists of determining the weights of the connections between the neurons such that the NN classifier shows a high performance. Since the weights are real-valued, this is a continuous optimization problem of the following form: Given are n decision variables $\{X_1, \ldots, X_n\}$ with continous domains. These domains are not restricted, i.e., each real number is feasible. Furthermore, the problem is unconstrained, which means that the variable settings do not depend on each other. Sought is a solution that minimizes the objective function called *square error percentage (SEP)*:

$$SEP = 100 \frac{o_{\max} - o_{\min}}{n_0 n_p} \sum_{p=1}^{n_p} \sum_{i=1}^{n_0} \left(t_i^p - o_i^p\right)^2 \quad , \quad (1)$$

where $o_{\max}$ and $o_{\min}$ are respectively the maximum and minimum values of the output signals of the output neurons, n_p represents the number of patterns, n_0 is the number of output neurons, and t_i^p and o_i^p represent respectively the expected and actual values of output neuron i for pattern p.

3 ACO for continuous optimization

ACO algorithms are iterative methods that try so solve optimization problems as follows. At each iteration candidate solutions are probabilistically constructed by sampling a probability distribution over the search space. Then, this probability distribution is modified using the better ones among the constructed solutions. The goal is to bias over time the sampling of solutions to areas of the search space that contain high quality solutions.

In ACO algorithms for discrete optimization problems, the probabilitiy distribution is discrete and is derived from artificial pheromone information. In a way, the pheromone information represents the stored search experience of the algorithm. In contrast, our ACO algorithm for continuous optimization, henceforth denoted by $\text{ACO}_{\mathbb{R}}$, utilizes a continuous probability density function (PDF). This density function is – for each solution construction – produced from a population P of solutions that the algorithm keeps at all times. The management of this population works as follows. Before the start of the algorithm, the population—whose size k is a parameter of the algorithm—is filled with random solutions. Even though the domains of the decision variables are not restricted, we used the initial interval $[-1, 1]$ for the sake of simplicity. Then, at each iteration a set of m solutions is generated and added to P. The same number of the worst solutions are removed from P. This biases the search process towards the best solutions found during the search.

For constructing a solution an ant acts as follows. First, it transforms the original set of decision variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ into a set of temporary variables $\mathbf{Z} = \{Z_1, \ldots, Z_n\}$. The purpose of introducing temporary variables is to improve the algorithms performance by limiting the correlation between decision variables. Note that this transformation also affects the population of solutions: All the solutions are transformed to the new coordinate system as well. The method of transforming the set of decision variables is presented towards the end of this section.

Then, at each construction step $i = 1, \ldots, n$, the ant chooses a value for decision variable Z_i. For performing this choice it uses a Gaussian kernel PDF, which is a weighted superposition of several Gaussian functions. For a decision variable Z_i the Gaussian kernel G_i is given as follows:

$$G_i(z) = \sum_{j=1}^{k} \omega_j \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu_j)^2}{2{\sigma_j}^2}} \quad , \forall\, z \in \mathbf{R} \ , \qquad (2)$$

where the j-th Gaussian function is derived from the j-th member of the population P (remember that k is the size of P). Note that $\vec{\omega}$, $\vec{\mu}$, and $\vec{\sigma}$ are vectors of size k. Hereby, $\vec{\omega}$ is the vector of weights, whereas $\vec{\mu}$ and $\vec{\sigma}$ are the vectors of means and standard deviations respectively.

Sampling directly the Gassian kernel PDF as defined in Equation 2 is problematic. Therefore we accomplish it with the following procedure. It may be proven that this procedure is exactly equivalent to sampling the PDF G_i directly. Before starting a solution construction, we choose exactly one of the Gaussian functions j, which is then used for all n construction steps.[1] A Gaussian function j^* is chosen with the following probability distribution:

$$\mathbf{p}_j = \frac{\omega_j}{\sum_{l=1}^{k} \omega_l} \quad , \forall\, j = 1, \ldots, k \ , \qquad (3)$$

where ω_j is the weight of Gaussian function j, which is obtained as follows. All solutions in P are ranked according to their quality (i.e., their SEP value) with the best solution having rank 1. Assuming the rank of the j-th solution in P to be r, the weight ω_j of the j-th Gaussian function is calculated according to the following formula:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} \cdot e^{-\frac{(r-1)^2}{2q^2k^2}} \quad , \qquad (4)$$

which essentially defines the weight to be a value of the Gaussian function with the argument of rank r, with mean in 1.0 and standard deviation of qk, where q is a parameter of the algorithm. When parameter q is small, the best-ranked solutions are strongly preferred, and when it is larger, the probability becomes more uniform. Thanks to using the ranks instead of the actual fitness function values, the algorithm is not sensitive to the scaling of the fitness function.

The sampling of the chosen Gaussian function j^* may be done using a uniform random generator in conjunction with (for instance) the Box-Muller method [5]. However, before doing that we have to specify the mean and the standard deviation of the j^*-th Gaussian function. As mean μ_{j^*} we choose the value of the i-th decision variable in solution j^*. It remains to specify the standard deviation σ_{j^*}. For doing that we calculate the average distance of the other population members from the j^*-th solution and multiply it by a parameter ρ, which regulates the speed of convergence:

$$\sigma_{j^*} = \rho \sum_{l=1}^{k} \sqrt{\left(z_i^l - z_i^{j^*}\right)^2} \qquad (5)$$

Parameter ρ has a role similar to the pheromone evaporation rate ρ in the combinatorial ACO. The higher the value of $\rho \in (0, 1)$, the lower the convergence speed of the algorithm, and hence the lower the learning rate. Since this whole process is done for each dimension (i.e., decision variable) in turn, each time the distance is calculated only with the use of one single dimension (the rest of them are discarded). This ensures that the algorithm is able to adapt to convergence, but also allows the handling of problems that are scaled differently in different directions.

Finally it remains to explain how the set of temporary decision variables $\mathbf{Z}$ is created from the original set $\mathbf{X}$.[2] An obvious choice for adapting the coordinate system to the distribution of population P would be the Principal Component Analysis (PCA). Although PCA works very well for reasonably regular distributions, its performance is no longer that interesting in case of more complex functions. The mechanism that we designed instead, is relatively simple. Each ant at each step of the construction process chooses a direction. The direction is chosen by randomly selecting a solution u from the population that is reasonably far away from the solution j^* chosen as mean of the PDF. Then, the vector $\vec{j^*u}$ becomes the chosen direction. The probability of choosing solution u (having solution j^* chosen as the mean of the PDF) is the following:

$$\mathbf{p}(u|j^*) = \frac{d(u, j^*)^4}{\sum_{l=1}^{k} d(l, j^*)^4} \quad , \qquad (6)$$

where function $d(\cdot)$ is the function that returns the distance between two members of the population P. Once this vector is chosen, the new orthogonal basis for the ant's coordinate system is created using the Gram-Schmidt process. It

[1] Note that this has also the advantage that it allows to exploit the (possibly existing) correlation between the variables.

[2] Note that ACO algorithms in general do not exploit correlation information between different decision variables (or components). In ACO$_{\mathbb{R}}$, due to the specific way the search experience is stored (i.e., as a population of solutions), it is in fact possible to take into account the correlation between the decision variables.

Table 1. Summary of the NN structures that we use for the three data sets.

Data set	Inp. layer	Hid. layer	Outp. layer	# of weights
Cancer1	9	6	2	74
Diabetes1	8	6	2	68
Heart1	35	6	2	230

takes as input all the (already orthogonal) directions chosen in earlier ant's steps and the newly chosen vector. The remaining missing vectors (for the remaining dimensions) are chosen randomly. Then, all the current coordinates of all the solutions in the population are rotated and recalculated according to this new orthogonal base resulting in the set of new temporary variables $\mathbf{Z}$. Only then is the ant able to measure the average distance, and subsequently to sample from the PDF (as it can now calculate the mean and standard deviation). At the end of the construction process, the chosen values of the temporary variables $\mathbf{Z}$ are converted back into the original coordinate system $\mathbf{X}$.

4 Experimental evaluation

An important collection of medical data sets for pattern classification is the well-known PROBEN1 data repository [11], which has been used various times in the past to evaluate and compare the performance of different methods for training NN classifiers. From the available data sets we chose Cancer1, Diabetes1, and Heart1 for our experimentation.

Cancer1 concerns the diagnosis of breast cancer (possible outcomes: yes or no). The data set contains altogether 699 patterns. Each pattern has 9 input parameters (i.e., measurements). We used the first 525 of the patterns (i.e., about 75%) as training set (i.e, for optimizing the NN weights), and the remaining 174 as test set. Diabetes1 concerns the diagnosis of diabetes (possible outcomes: yes or no). Each pattern has 8 input parameters. The data set contains altogether 768 patterns. We used the first 576 of them as training set and the remaining 192 as test set. Finally, Heart1 concerns the diagnosis of a heart condition (possible outcomes: yes or no). Each pattern has 35 input parameters. The data set contains altogether 920 cases. We used the first 690 of them as training set and remaining 230 as test set.

Concerning the structure of the feed-forward NNs that we used, we took inspiration from the literature. More specifically we used the same network structures that were used in [1]. For an overview of these NN structures see Table 1.

Algorithmis for comparison. For comparison purposes we have re-implemented some algorithms traditionally used for training NNs, namely the back-propagation (BP) algorithm [12], and the Levenberg-Marquardt (LM) algorithm [8, 9]. Both algorithms (i.e., BP and LM) require gradient information, hence they require the neuron transfer function $f(\cdot)$ to be differentiable. Consequently, these algorithms may not — in contrast to $\text{ACO}_{\mathbb{R}}$ — be used in case the neuron transfer function is not differentiable or is unknown. In case of training NNs whose transfer function is differentiable, the drawback of general optimization algorithms such as $\text{ACO}_{\mathbb{R}}$ is however that they do not exploit available additional information e.g., the gradient. In order to see how the additional gradient information influences the performance of $\text{ACO}_{\mathbb{R}}$, we have also implemented hybridized versions of $\text{ACO}_{\mathbb{R}}$, namely $\text{ACO}_{\mathbb{R}}$-BP and $\text{ACO}_{\mathbb{R}}$-LM. In these hybrids, each solution generated by the $\text{ACO}_{\mathbb{R}}$ algorithm is improved by running a single improving iteration of either BP or LM before being evaluated. Finally, we wanted to see how all the algorithms tested compare to a simple random search (RS) method. This is an algorithm that randomly generates a set of values for the weights and then evaluates these solutions. As we used a sigmoid function as neuron transfer function, it was sufficient to limit the range of weight values to values close to 0. Hence, we arbitrarily chose a range of [-5,5]. We have performed a limited scope parameter tuning for all algorithms using Birattari's F-RACE method (see [3]). The outcome is shown in Table 2. Not included in the table are the parameters common to all $\text{ACO}_{\mathbb{R}}$ versions, namely q and m. For these parameters we used the settings $q = 0.01$, and $m = 2$ (the number of ants used in each iteration).

Results. Figure 2 presents the results obtained for the cancer, diabetes, and heart test problems in the form of box-plots. Each figure presents the distributions of the actual classification error percentage (CEP) values obtained by the algorithms (over 50 independent runs). As stopping condition we used 1000 fitness function evaluations (adopted from [1]).

Cancer1 (see Figure 2(a)) appears to be the easiest data set among the three that we tackled. All algorithms obtained reasonably good results, including the RS method. However, the best performing algorithm is BP. From the fact that the results obtained by RS do not differ significantly from the results obtained by other–more complex algorithms, it may be concluded that the problem is relatively easy, and that there are a lot of reasonably good solutions scattered over the search space. None of the algorithms was able to classify all the test patterns correctly. This may be due to the limited size of the training set, i.e. there might have been not enough information in the training set to generalize per-

Table 2. The final parameter values for our algorithms. Note that η is the step-size parameter of BP, and β is the adaptation-step parameter of LM.

	Cancer1				Diabetes1				Heart1			
Algorithm	k	ρ	η	β	k	ρ	η	β	k	ρ	η	β
$ACO_{\mathbb{R}}$	148	0.95	-	-	136	0.8	-	-	230	0.6	-	-
$ACO_{\mathbb{R}}$-BP	148	0.98	0.3	-	136	0.7	0.1	-	230	0.98	0.4	-
$ACO_{\mathbb{R}}$-LM	148	0.9	-	10	136	0.1	-	10	230	0.1	-	10
BP	-	-	0.002	-	-	-	0.01	-	-	-	0.001	-
LM	-	-	-	50	-	-	-	5	-	-	-	1.5

fectly.

Diabetes1 (see Figure 2(b)) is a problem that is more difficult than Cancer1. All our algorithms clearly outperform RS. However, the overall performance of the algorithms in terms of the CEP value is not very good. The best performing is again BP. The less good overall performance of the algorithms may again indicate that the training set does not represent fully all the possible patterns.

The Heart1 problem (see Figure 2(c)) is–with 230 weights–the largest problem that we tackled. It is also the one on which the performance of the algorithms differed mostly. All tested algorithms clearly outperform RS, but there are also significant differences among the more complex algorithms. BP, which was performing quite well on the other two test problems, did not do so well on Heart1. $ACO_{\mathbb{R}}$ achieves results similar to BP. In turn, LM which was not performing so well on the first two problems, obtains quite good results. Very interesting is the performance of the hybridized versions of $ACO_{\mathbb{R}}$, namely $ACO_{\mathbb{R}}$-BP and $ACO_{\mathbb{R}}$-LM. The $ACO_{\mathbb{R}}$-BP hybrid clearly outperforms both $ACO_{\mathbb{R}}$ and BP. $ACO_{\mathbb{R}}$-LM outperforms respectively $ACO_{\mathbb{R}}$ and LM. Additionally, $ACO_{\mathbb{R}}$-LM performs best overall.

Finally, it is interesting to compare the performance of the $ACO_{\mathbb{R}}$ based algorithms to some other general optimization algorithms. Alba and Chicano [1] have published the results of a genetic algorithm (GA) used for tackling exactly the same three problems as we did. They have tested not only a stand-alone GA, but also its hybridized versions: GA-BP and GA-LM. Their results can be compared to ours due to the fact that 1000 evaluations as stopping criterion were used for all the algorithms. The results presented in Table 3 clearly show that the stand-alone $ACO_{\mathbb{R}}$ performs better than the stand-alone GA for all the test problems. $ACO_{\mathbb{R}}$-BP and $ACO_{\mathbb{R}}$-LM perform respectively better than GA-BP and GA-LM on both of the more difficult problems Diabetes1 and Heart1 and worse on Cancer1. For the Heart1 problem the mean performance of any $ACO_{\mathbb{R}}$ based algorithm is significantly better than the best GA based algorithm (which was reported as the state-of-the-art for this problem in 2004).

5 Conclusion

We have presented an ACO algorithm (i.e., $ACO_{\mathbb{R}}$) for the training of feed-forward neural networks for pattern classification. The performance of the algorithm was evaluated on real-world test problems and compared to specialized algorithms for feed-forward neural network training, namely BP and LM, and also to algorithms based on a genetic algorithm. The performance of the stand-alone $ACO_{\mathbb{R}}$ was comparable (or at least not much worse) than the performance of specialized algorithms for neural network training. This result is particularly interesting as $ACO_{\mathbb{R}}$—being a much more generic approach—allows also the training of networks in which the neuron transfer function is either not differentiable or unknown. The hybrid of $ACO_{\mathbb{R}}$ and the LM algorithm (i.e., $ACO_{\mathbb{R}}$-LM) was in some cases able to outperform BP and LM. Finally, the results indicate that $ACO_{\mathbb{R}}$ compare favorably against other general-purpose optimizers such as GAs.

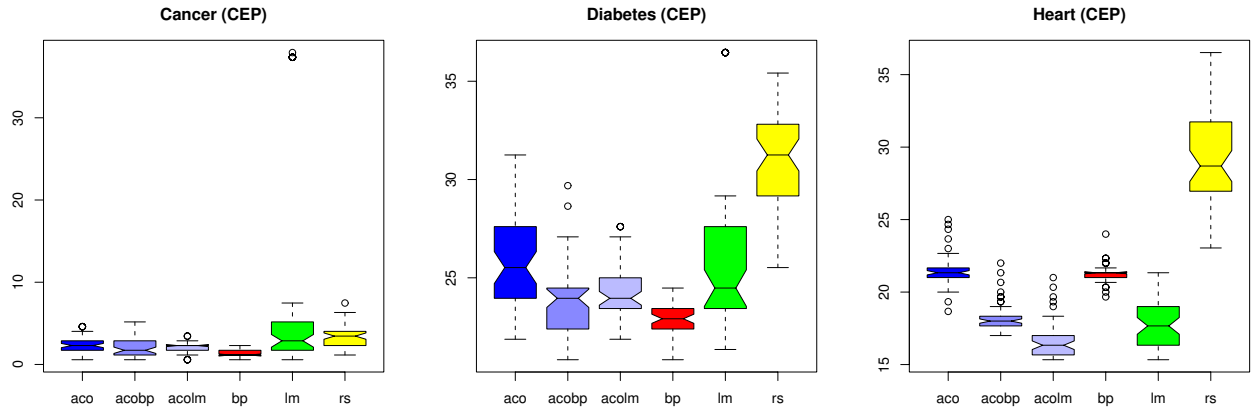
References

[1] E. Alba and J. F. Chicano, "Training neural networks with GA hybrid algorithms," in *Proc. of GECCO 2004*, ser. LNCS, K. Deb et al., Ed., vol. 3102. Springer Verlag, Berlin, Germany, 2004, pp. 852–863.

[2] B. Bilchev and I. C. Parmee, "The ant colony metaphor for searching continuous design spaces," in *Proc. of the AISB Workshop on Evolutionary Computation*, ser. LNCS, vol. 993, 1995, pp. 25–39.

[3] M. Birattari, "The problem of tuning metaheuristics as seen from a machine learning perspective," Ph.D. thesis, ULB, Brussels, Belgium, 2004.

[4] C. M. Bishop, *Neural networks for pattern recognition*. MIT Press, 2005.



(a) Cancer1 (b) Diabetes1 (c) Heart1

Figure 2. Box-plots for (a) Cancer1**, (b)** Diabetes1**, and (c)** Heart1**. The boxes are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95 % confidence interval.**

Table 3. Comparison of $\mathrm{ACO}_{\mathbb{R}}$ with GA based algorithms (see [1]). For each problem-algorithm pair we give the mean (over 50 independent runs), and the standard deviation (in brackets). The best result of each comparison is indicated in bold.

	GA	$\mathrm{ACO}_{\mathbb{R}}$	GA-BP	$\mathrm{ACO}_{\mathbb{R}}$-BP	GA-LM	$\mathrm{ACO}_{\mathbb{R}}$-LM
Cancer1	16.76 (6.15)	**2.39 (1.15)**	**1.43 (4.87)**	2.14 (1.09)	**0.02 (0.11)**	2.08 (0.68)
Diabetes1	36.46 (0.00)	**25.82 (2.59)**	36.36 (0.00)	**23.80 (1.73)**	28.29 (1.15)	**24.26 (1.40)**
Heart1	41.50 (14.68)	**21.59 (1.14)**	54.30 (20.03)	**18.29 (1.00)**	22.66 (0.82)	**16.53 (1.37)**

[5] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *Annals of Mathematical Statistics*, vol. 29, no. 2, pp. 610–611, 1958.

[6] M. Dorigo and T. Stützle, *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[7] J. Dréo and P. Siarry, "A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions," in *Proc. of ANTS 2002*, ser. LNCS, M. Dorigo et al., Eds., vol. 2463. Springer Verlag, Berlin, Germany, 2002, pp. 216–221.

[8] K. Levenberg, "A method for the solution of certain problems in least squares," *Quarterly Applied Mathematics*, vol. 2, pp. 164–168, 1944.

[9] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM Journal on Applied Mathematics*, vol. 11, pp. 431–441, 1963.

[10] N. Monmarché, G. Venturini, and M. Slimane, "On how pachycondyla apicalis ants suggest a new search algorithm," *Future Generation Computer Systems*, vol. 16, pp. 937–946, 2000.

[11] L. Prechelt, "Proben1—a set of neural network benchmark problems and benchmarking rules," Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, Tech. Rep. 21, 1994.

[12] D. Rummelhart, G. Hinton, and R. Williams, "Learning representations by backpropagation errors," *Nature*, vol. 323, pp. 533–536, 1986.

[13] K. Socha, "Extended ACO for continuous and mixed-variable optimization," in *Proc. of ANTS 2004*, ser. LNCS, M. Dorigo et al., Eds. Springer Verlag, Berlin, Germany, 2004, 25–36.