Neural Comput & Applic (2007) 16:235–247
DOI 10.1007/s00521-007-0084-z

ORIGINAL ARTICLE

An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training

Krzysztof Socha · Christian Blum

Received: 1 December 2006 / Accepted: 21 December 2006 / Published online: 2 March 2007

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. Recently we proposed a first ACO variant for continuous optimization. In this work we choose the training of feed-forward neural networks for pattern classification as a test case for this algorithm. In addition, we propose hybrid algorithm variants that incorporate short runs of classical gradient techniques such as backpropagation. For evaluating our algorithms we apply them to classification problems from the medical field, and compare the results to some basic algorithms from the literature. The results show, first, that the best of our algorithms are comparable to gradient-based algorithms for neural network training, and second, that our algorithms compare favorably with a basic genetic algorithm.

Keywords Ant colony optimization · Continuous optimization · Feed-forward neural network training

1 Introduction

Ant colony optimization (ACO) is an optimization technique that was introduced for the application to discrete optimization problems in the early 1990s by Dorigo et al. [13–15]. The origins of ant colony optimization are in a field called swarm intelligence [8] 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. When searching for food, ants initially explore the area surrounding their nest in a random manner. While moving, ants leave a chemical pheromone trail on the ground. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails guide other ants to the food source. It has been shown in [12] that the indirect communication between the ants via pheromone trails 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) [3], the API algorithm [27], and Continuous Interacting Ant Colony (CIAC) [16]. However, all these approaches do not follow the original ACO framework. The latest approach—called $ACO_{\mathbb{R}}$ —was proposed in [31, 33]. Up to now, $ACO_{\mathbb{R}}$ is the ACO variant that is closest to the spirit of ACO for combinatorial problems. In [31, 33] it was shown that $ACO_{\mathbb{R}}$ has clear advantages over the existing ACO variants when applied to continuous benchmark functions.

K. Socha (✉)
IRIDIA, CoDE, Université Libre de Bruxelles,
Brussels, Belgium
e-mail: ksocha@ulb.ac.be

C. Blum
ALBCOM, LSI, Universitat Politècnica de Catalunya,
Barcelona, Spain
e-mail: cblum@lsi.upc.edu

1.1 The goal of this work

In this work we choose as a test case for $ACO_{\mathbb{R}}$ the problem of feed-forward neural network (NN) training for pattern classification, which is an important real-world problem. Feed-forward NNs are commonly used for the task of pattern classification [6], but they require prior configuration. Generally, the configuration problem consists 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. Readers interested in the evolution of NN structures might refer, for example, to [17, 35, 38].

We want to state clearly at this point, that $ACO_{\mathbb{R}}$ is still a quite basic algorithm. For example, it does not include possible techniques for a more efficient exploration of the search space such as the use of multiple colonies. This is the reason why we compare $ACO_{\mathbb{R}}$ only to *basic* versions of other algorithms including gradient-based techniques as well as general purpose optimizers. We intentially do not compare to highly specialized algorithms, neither do we intent to improve the state-of-the-art results for the three benchmark instances that we tackle. This is left for future work. The important aspect of this work is the comparison of $ACO_{\mathbb{R}}$ with other basic algorithms under the *same conditions*.[1]

1.2 Prior work on bio-inspired techniques for NN training

During the last 20 years, numerous bio-inspired algorithms have been developed for the problem of NN training. In particular the evolutionary computation community has produced a vast number of works on this topic. Representative examples are genetic algorithms [1, 28], evolution strategies [24], or estimation of distribution algorithms [11]. Recent developments from the swarm intelligence field include the particle swarm optimization algorithm proposed in [26]. For an overview, the interested reader might refer to [2]. $ACO_{\mathbb{R}}$ has inevitably similarities with already existing techniques; in particular with some estimation of distribution algorithms [23], or with evolution strategies that employ the covariance matrix adaptation (CMA) method [21]. However, an advantage of our algorithm is that it originates from a different field with a different point of view. This means that possibly our algorithm can benefit from techniques for guiding the search process or for making the search process more efficient other than the ones available in evolutionary computation. This can prove beneficial for our algorithm in the future.

The outline of our work is as follows. In Sect. 2 we present the $ACO_{\mathbb{R}}$ algorithm for continuous optimization. In Sect. 3 we describe the test case of neural network training, including the hybridizations of $ACO_{\mathbb{R}}$ with backpropagation (BP) and the Levenberg–Marquardt (LM) algorithm. Furthermore, we present the experimental evaluation of our algorithms. Finally, in Sect. 4 we offer conclusions and a glimpse of future work.

2 ACO for continuous optimization ($ACO_{\mathbb{R}}$)

In general, the ACO approach attempts to solve an optimization problem by iterating the following two steps:

1. Candidate solutions are constructed in a probabilistic way by using a probability distribution over the search space.
2. The candidate solutions are used to modify the probability distribution in a way that is deemed to bias future sampling toward high quality solutions.

ACO algorithms for combinatorial optimization problems make use of a pheromone model in order to probabilistically construct solutions. A pheromone model is a set of so-called pheromone trail parameters. The numerical values of these pheromone trail parameters (that is, the pheromone values) reflect the search experience of the algorithm. They are used to bias the solution construction over time to regions of the search space containing high quality solutions.

However, instead of solving a combinatorial problem, let us assume that we want to minimize a continuous function

$$f : S \subseteq \mathbb{R}^n \mapsto \mathbb{R}.$$

The search space S can hereby be modelled by a set of n decision variables X_i ($i = 1,\ldots, n$) with continuous domains. In order to tackle this problem, $ACO_{\mathbb{R}}$ uses a solution archive for the derivation of a probability distribution over the search space. While a pheromone model can be seen as an implicit memory of the search history, a solution archive is an explicit memory. A similar idea was proposed by Guntsch and Middendorf in [19] for combinatorial optimization problems. This related approach is called *Population-Based ACO* (PB-ACO). The components of the

[1] Note that this paper is an extension of the work published in [7, 32]. The extension consists in a more detailed explanation of the algorithm itself, the conduction of a fourfold cross-validation for all applications to test instances, and the conduction of tests for determining the statistical significance of the obtained results.

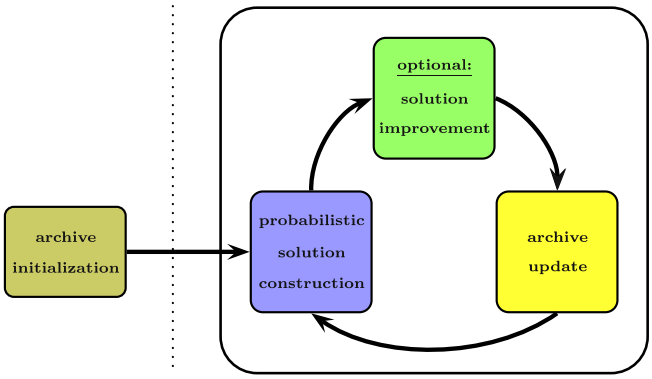

Fig. 1 The working of $ACO_{\mathbb{R}}$. First, the solution archive must be initialized. Then, at each iteration a number of solutions are probabilistically constructed. These solutions may be improved by any improvement mechanism (for example, local search or a gradient technique). Finally, the solution archive is updated with the generated solutions

$ACO_{\mathbb{R}}$ algorithm are schematically shown in Fig. 1. In the following we outline the components of $ACO_{\mathbb{R}}$ in more detail.

2.1 Archive structure, initialization, and update

$ACO_{\mathbb{R}}$ keeps a history of its search process by means of storing solutions in a *solution archive* T. Given an n-dimensional optimization problem and a solution s_l, $ACO_{\mathbb{R}}$ stores in T the values of the solutions' n variables and the value of the objective function denoted by $f(s_l)$. The value of the ith variable of lth solution hereby denoted by s_l^i. The structure of the solution archive T is presented in Fig. 2.

We denote the number of solutions memorized in the archive by k. This parameter influences the complexity of the algorithm.[2] Before the start of the algorithm, the archive is initialized with k random solutions. Even though the domains of the decision variables are—in the case of feed-forward NN training—not restricted, we used the initial interval [–1,1] for the sake of simplicity. At each algorithm iteration, a set of m solutions is probabilistically generated and added to T. The same number of the worst solutions are removed from T. This biases the search process towards the best solutions found during the search.

2.2 Probabilistic solution construction

For the probabilistic construction of solutions, $ACO_{\mathbb{R}}$ uses so-called *probability density functions* (PDFs). Before describing the solution construction mechanism, we first discuss certain characteristics of PDFs. In principle, a PDF may be any function $P(x) : \mathbb{R} \ni x \rightarrow P(x) \in \mathbb{R}$ such that:

$$\int_{-\infty}^{\infty} P(x)\mathrm{d}x = 1 \tag{1}$$

For a given probability density function $P(x)$, an associated *cumulative distribution function* (CDF) $D(x)$ may be defined, which is often useful when sampling the corresponding PDF. The CDF $D(x)$ associated with PDF $P(x)$ is defined as follows:

$$D(x) = \int_{-\infty}^{x} P(t)\mathrm{d}t \tag{2}$$

The general approach to sampling PDF $P(x)$ is to use the *inverse* of its CDF, $D^{-1}(x)$. When using the inverse of the CDF, it is sufficient to have a pseudo-random number generator that produces uniformly distributed real numbers.[3] However, it is important to note that for an arbitrarily chosen PDF $P(x)$, it is not always straightforward to find $D^{-1}(x)$.

One of the most popular functions that is used as a PDF is the Gaussian function. It has some clear advantages (outlined below) but it also has some disadvantages. For example, a single Gaussian function is not able to describe a situation in which two disjoint areas of the search space are promising. This is because it only has one maximum. Due to this fact, $ACO_{\mathbb{R}}$ uses a PDF based on Gaussian functions, but slightly enhanced—*a Gaussian kernel PDF*. Similar constructs have been used before [9], but not exactly in the same way. A Gaussian kernel is defined as a weighted sum of several one-dimensional Gaussian functions $g_l^i(x)$:

$$G^i(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} \mathrm{e}^{-\frac{(x-\mu_l^i)^2}{2\sigma_l^{i\,2}}}, \tag{3}$$

[2] Note that k can not be smaller than the number of dimensions of the problem being solved. This is due to the explicit handling of correlation among variables as explained in Sect. 3: In order to be able to rotate the coordinate system properly, the number of solutions available has to be at least equal to the number of dimensions.

[3] Such pseudo-random number generators are routinely available for most programming languages.

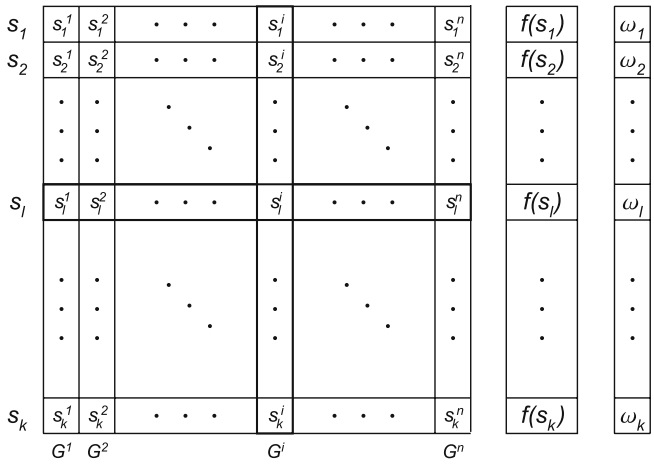

Fig. 2 The archive of k solutions kept by $ACO_{\mathbb{R}}$. The solutions are ordered according to their quality, i.e., for a minimization problem: $f(s_1) \leq f(s_2) \leq \ldots \leq f(s_l) \leq \ldots \leq f(s_k)$. Each solution s_l has an associated weight ω_l that depends on the solution quality: $\omega_1 \geq \omega_2 \geq \ldots \geq \omega_l \geq \ldots \geq \omega_k$

where $i = 1,\ldots,n$, that is, for each dimension of the given continuous optimization problem we will define a different Gaussian kernel PDF. Each Gaussian kernel PDF $G^i(x)$ is parametrized by means of three vectors of parameters: $\boldsymbol{\omega}$ is the vector of weights associated with the individual Gaussian functions, $\boldsymbol{\mu}^i$ is the vector of means, and $\boldsymbol{\sigma}^i$ is the vector of standard deviations. The cardinality of all these vectors is equal to the number of Gaussian functions constituting the Gaussian kernel. The advantages of Gaussian kernel PDFs are that they allow a reasonably easy sampling, and yet provide a high variety of possible shapes in comparison to a single Gaussian function. An example of how such a Gaussian kernel PDF may look like is presented in Fig. 3.

For constructing a solution an ant performs n construction steps. At each construction step $i = 1,\ldots,n$, the ant chooses a value for decision variable X_i. This is done by sampling a Gaussian kernel PDF $G^i(x)$, which is derived from the k solutions stored in archive T. In order to define $G^i(x)$, we must define the contents of the three vectors $\boldsymbol{\omega}$, $\boldsymbol{\mu}^i$, and $\boldsymbol{\sigma}^i$ (see above). First, the values of the i-th variable of all the solutions in the archive become the elements of the vector $\boldsymbol{\mu}^i$:

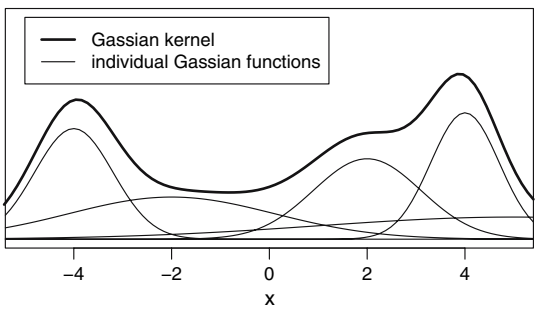

Fig. 3 Example of five Gaussian functions and their superposition—the resulting Gaussian kernel PDF (illustration limited to the range $x \in [-5,5]$)

$$\boldsymbol{\mu}^i = \{\mu_1^i, \ldots, \mu_k^i\} = \{s_1^i, \ldots, s_k^i\} \tag{4}$$

The vector of weights $\boldsymbol{\omega}$ is created in the following way. Each solution that is added to the archive T is evaluated and ranked (ties are broken randomly). The solutions in the archive are sorted according to their rank—i.e., solution s_l has rank l. The weight ω_l of the solution s_l is calculated according to the following formula:

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}}, \tag{5}$$

which essentially defines the weight to be a value of the Gaussian function with argument l, mean 1.0, and standard deviation qk, where q is a parameter of the algorithm. When q is small, the best-ranked solutions are strongly preferred, and when it is large, the probability becomes more uniform. The influence of this parameter on $ACO_{\mathbb{R}}$ is similar to adjusting the balance between the *iteration-best* and the *best-so-far* pheromone updates used in ACO for combinatorial optimization (see, for example, [36]).

In order to define the final shape of $G^i(x)$, it remains to specify the vector $\boldsymbol{\sigma}^i$ of the standard deviations. Before presenting in detail how this is done, we first explain the way in which $G^i(x)$ is sampled. In practice, the sampling process is accomplished as follows. First, the elements of the weight vector $\boldsymbol{\omega}$ are computed following Eq. 6. Then, the sampling is done in two phases. Phase one consists of choosing one of the Gaussian functions that compose the Gaussian kernel PDF. The probability p_l of choosing the lth Gaussian function is given by

$$p_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r}. \tag{6}$$

Phase two consists of sampling the chosen Gaussian function (i.e., at construction step i—function g_l^i). This may be done using a random number generator that is able to

generate random numbers according to a parametrized normal distribution, or by using a uniform random generator in conjunction with, for instance, the Box-Muller method [10]. This two-phase sampling is equivalent to sampling the Gaussian kernel PDF $G^i(x)$ as defined in Eq. 4.

It is clear that at construction step i, the standard deviation needs only to be known for the single Gaussian function $g_l^i(x)$ chosen in phase one. Hence, we do not calculate the whole vector of standard deviations $\boldsymbol{\sigma}^i$, but only the entry σ_l^i that is needed.

The choice of the lth Gaussian function is done only once per ant and iteration. This means that an ant uses the Gaussian functions associated with the chosen solution s_l—that is, functions g_l^i, $i = 1,...,n$—for constructing the whole solution in a given iteration. This allows exploiting the correlation between the variables, which is explained in detail in Sect. 3. Of course, the actual Gaussian function sampled differs at each construction step, as for step i, $\mu_l^i = s_l^i$, and σ_l^i is calculated dynamically, as follows.

In order to establish the value of the standard deviation σ_l^i at construction step i, we calculate the average distance from the chosen solution s_l to other solutions in the archive, and we multiply it by the parameter ξ:

$$\sigma_l^i = \xi \sum_{e=1}^{k} \frac{|x_e^i - x_l^i|}{k-1}. \tag{7}$$

The parameter $\xi > 0$, which is the same for all the dimensions, has an effect similar to that of the pheromone evaporation rate in ACO for combinatorial optimization. The higher the value of ξ, the lower the convergence speed of the algorithm. While the pheromone evaporation rate in ACO influences the long term memory—i.e., lower quality solutions are forgotten faster—ξ in $ACO_{\mathbb{R}}$ influences the way the long term memory is used—i.e., lower quality solutions have on average a lower influence when constructing new solutions.

As mentioned before, this whole process is done for each dimension $i = 1,...,n$ in turn, and each time the average distance σ_l^i is calculated only with the use of the single dimension i. This ensures that the algorithm is able to adapt to linear transformation of the considered problem (e.g., moving from a sphere model to an ellipsoid, or rotating an ellipsoid).

2.3 Exploiting correlations between decision variables

ACO algorithms in general do not exploit correlation information between different decision variables. In $ACO_{\mathbb{R}}$, due to the use of the solution archive, it is in fact possible to take into account the correlation between the decision variables. Consider Fig. 4, where the two-dimensional Ellipsoid test function

$$f_{\mathrm{EL}}(\mathbf{x}) = \sum_{i=1}^{n} \left(100^{\frac{i-1}{n-1}} x_i\right)^2, n = 2 \tag{8}$$

is shown—not rotated (left), and then randomly rotated (right). The test function is presented from the view point of the $ACO_{\mathbb{R}}$ algorithm, namely as a set of points representing different solutions found by the ants and stored in the solution archive. The darker a point, the higher the quality of the correpsonding solution (and the higher its rank). While in the left plot the variables are not correlated (i.e., for good solutions, the value of one coordinate does not depend on the value of the other coordinate), on the right plot they are highly correlated.

The default coordinate system that corresponds to the set of the original decision variables X_i, $i = 1,2$, is marked in bold. It is clear that the axes of that coordinate system align well with the scaling of the test function in the left plot. The Gaussian kernel PDFs for both dimensions are indicated on the right and above the plot. Clearly, a new solution generated by these Gaussian kernel PDFS has a high probability to be in the promising region. In contrast, in the case of the rotated Ellipsoid function presented in the right plot, the PDFs created with the default coordinate system cover basically the whole search space (here $\mathbf{D} = [-2,2]^2$). This means that sampled solutions would be scattered all over the search space. In order to avoid this, the other coordinate system indicated in the right plot should be used. When using this other coordinate system, the sampling would be as efficient as in the case of the non-rotated Ellipsoid function on the right.

In fact, $ACO_{\mathbb{R}}$ dynamically adapts the coordinate system used by each ant in order to minimize the correlation between different decision variables. The adaptation of the coordinate system is accomplished by expressing the set of decision variables $\mathbf{X}$ with temporary variables Z_i, $i = 1,...,n$ that are linear combinations of X_i, $i = 1,...,n$. The following paragraphs shortly present how this is done.

One possible technique for adapting the coordinate system to the distribution of the solutions in the archive is known as principal component analysis (PCA) (see, for example, [22]). PCA works by performing a statistical analysis of the solutions in the archive in order to distinguish the principal components. However, due to the fact that PCA is deterministic, for most non-trivial problems it is not robust enough and often leads to stagnation.

The mechanism that we designed instead, is relatively simple. Each ant at each step of the construction process chooses a *direction* in the search space. The direction is chosen by randomly selecting a solution s_u that is reasonably

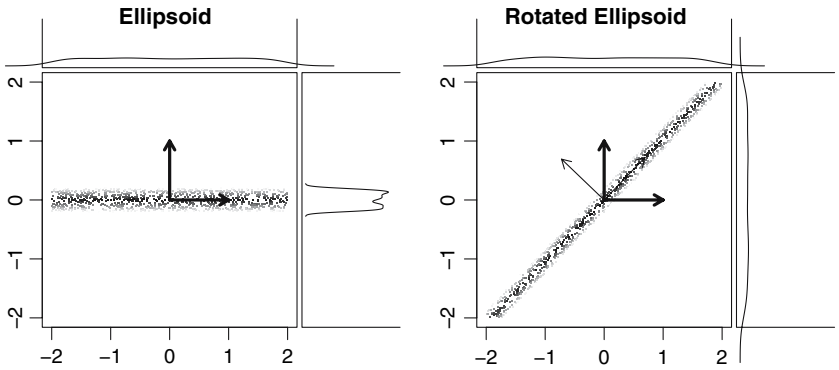

Fig. 4 Example of the Ellipsoid function: not rotated (*left graphic*), and rotated by 45° (*right graphic*). The function is shown by means of 10,000 points of which only the best 1,000 points are visible. Note that the darker a point the higher its rank (or quality). The original coordinate system is marked in *bold*. In addition, the optimal one is indicated in the *right plot*. Also, the examples of the Gaussian kernel PDFs as generated using the default coordinate system are given on the *right* and above each plot

far away from the solution s_l chosen earlier as the mean of the PDF. Then, the vector $\mathbf{s_l}\ \mathbf{s_u}$ becomes the chosen direction. The probability of choosing solution s_u at step i (having chosen earlier solution s_l as the mean of the PDF) is the following:

$$p(s_u|s_l)_i = \frac{d(s_u, s_l)_i^4}{\sum_{r=1}^{k} d(s_r, s_l)_i^4}, \tag{9}$$

where the function $d(\cdot,\cdot)_i$ returns the Euclidean distance concerning the $(n–i + 1)$-dimensional search sub-space[4] between two members of the solution archive T. Once this vector is chosen, the new orthogonal basis for the ant's coordinate system is created using the Gram-Schmidt process [18]. It 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 archive are rotated and recalculated according to this new orthogonal base resulting in the set of new temporary variables Z_i, $i = 1,...,n$.

At the end of the solution construction process, the chosen values of the temporary variables Z_i, $i = 1,...,n$ are converted back into the original coordinate system, giving rise to a set of values for the original decision variables X_i, $i = 1,...,n$.

2.4 Improvement mechanisms

The fact that general optimization methods such as $ACO_{\mathbb{R}}$ do not consider any gradient information—even though it might be available—may give them a disadvantage when compared to methods that use this information. In order to avoid this disadvantage, some iterations of gradient-based algorithms might be applied to all solutions generated by the ants.

3 An application of $ACO_{\mathbb{R}}$: the NN training test case

As a test case for the evaluation of $ACO_{\mathbb{R}}$ we chose the training of feed-forward NNs for the purpose of pattern classification. First, we will shortly present the concept of feed-forward NNs for pattern classification. Then, we are going to present the problem instances chosen for the experimental evaluation, the tuning of the algorithms, and the results obtained. In addition to the experimental evaluation of our (re-)implemented algorithms, we will also provide a comparison to some results from the literature. In particular, we compare our approaches to a basic genetic algorithm as well as to hybrids of this genetic algorithm concerning the BP and LM algorithm.

3.1 Feed-forward neural networks for pattern classification

A data set 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 NNs [6].

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 Fig. 5). 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

[4] At step i, only dimensions i through n are used.

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. 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 (that is, the NN configuration problem) 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 $\{X_1,...,X_n\}$ with continuous domains. As these domains are not restricted, each real number is feasible (i.e., $X_i \in \mathbb{R}$). Furthermore, the problem is unconstrained, which means that the variable settings do not depend on each other. Sought is a solution that minimizes the *classification error percentage* (CEP), that is, the percentage of wrongly classified patterns. Note that we will use the CEP in order to evaluate the results of our algorithm. However, when performing the NN training, the use of CEP as an objective function has disadvantages. For example, CEP induces a fitness landscape that is characterized by many plateaus, that is, areas in the search space in which the solutions have the same CEP value. In general, this is not a desirable property of an objective function. Therefore, for the NN training process we use a different objective function called *square error percentage* (SEP):

$$\mathrm{SEP} = 100\frac{o_{\max} - o_{\min}}{n_0 n_p}\sum_{p=1}^{n_p}\sum_{i=1}^{n_0}(t_i^p - o_i^p)^2, \qquad (10)$$

where $o_{\max}$ and $o_{\min}$ are respectively the maximum and minimum values of the output signals of the output neurons (depending on the neuron transfer function), 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.

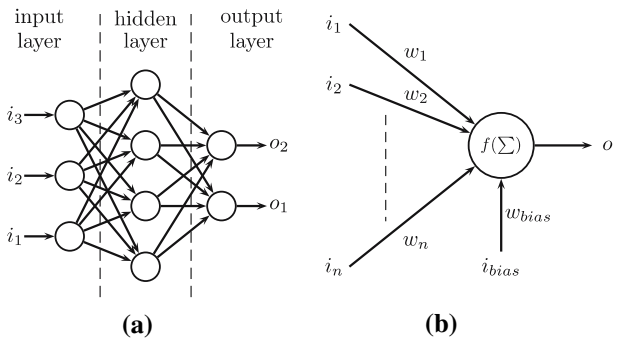

Fig. 5 **a** Feed-forward NN with one hidden layer of neurons. Note that each neuron of a certain layer is connected to each neuron of the next layer. **b** One single neuron (from either the hidden layer, or the output layer). The neuron receives inputs (i.e., signals i_l, weighted by weights w_l) from each neuron of the previous layer. Additionally, it receives a so-called bias input i_{bias} with weight w_{bias}. The transfer function $f(\Sigma)$ of a neuron transforms the sum of all the weighted inputs into an output signal, which serves as input for all the neurons of the following layer. Input signals, output signals, biases and weights are real values

3.2 Algorithms compared

In addition to the $\text{ACO}_{\mathbb{R}}$ algorithm outlined in Sect. 2, we will deal with the following algorithms in our experimental study:

- **BP:** This is the standard backpropagation algorithm [30]; without acceleration techniques and without regularization techniques such as *weight decay* or *early stopping* for avoiding overfitting. The reason for not using these techniques is that we wanted to compare basic algorithm versions.
- **LM:** The standard Levenberg–Marquardt algorithm [20].
- $\text{ACO}_{\mathbb{R}}$-BP (**or simply acobp**): This algorithm is obtained by applying to each solution generated by $\text{ACO}_{\mathbb{R}}$ one improving iteration of BP.
- $\text{ACO}_{\mathbb{R}}$-LM (**or simply acolm**): This algorithm is obtained by applying to each solution generated by $\text{ACO}_{\mathbb{R}}$ one improving iteration of LM.
- **Random search (RS):** This algorithm generates at each iteration a random solution (that is, a random weight settings). Since we used a sigmoidal neuron transfer function, it was not restrictive to limit the range of weight values to the interval [–5,5] for RS.

Finally, note that all our algorithms were implemented using the R programming language[5] (a free alternative to S+).

3.3 Experimental setup

In order to ensure a reasonably fair comparison, we allowed for each application of the 6 algorithms the same number of solution evaluations (namely 1,000 evaluations). Note that this does not mean that they spend exactly the same amount of computation time: apart from evaluating solutions, each of these algorithms (with the exception of RS) conducts calculations that the other algorithms do not

[5] http://www.r-project.org

need. However, the resulting computation times are in the same order of magnitude.

For evaluating the algorithms, we performend a so-called *k-fold cross-validation*. Hereby, the set of pattern is divided into k subsets. Then, k experiments are performed in which one of the k subsets is used as the test set and the remaining k–1 subsets are joined to form the training set. Then the average CEP value across all k experiments is computed. The aim of k-fold cross-validation is to average out the effects of the training/test set division. The disadvantage of this method is that the training algorithm has to be re-run from scratch k times, which uses quite a lot of computation time. In this work we perform a fourfold cross-validation.

3.4 Problem instances

Due to their practical importance, we chose to evaluate the performance of $ACO_{\mathbb{R}}$ on classification problems arising in the medical field. More specifically, we chose three problems from the well-known PROBEN1 benchmark set [29], namely Cancer1, Diabetes1, and Heart1. Each of these problems consists of a number of patterns together with their correct classification, that is, Cancer1 consists of 699 patterns from a breast cancer database, Diabetes1 consists of 768 patterns concerning diabetes patients, and Heart1 is the biggest of the three data sets, consisting of 920 patterns describing a heart condition. Each pattern of the three problems is either classified as pathological, or as normal. Furthermore, each pattern consists of a number of measurements (i.e., numerical values): 9 measurements in the case of Cancer1, 8 in the case of Diabetes1, and 35 in the case of Heart1. The goal consists in generating a classifier that takes the measurements of a pattern as input, and provides its correct classification as output.

Concerning the structure of the feed-forward NNs that we used, we took inspiration from the literature. More specifically we used the same NN structures that were used in [1]: one hidden layer with six neurons. Table 1 gives an overview of the number of neurons (columns two, three, and four) and the resulting number of NN weights (last table column), for each of the three problem instances. The number of NN weights is obtained by the following formula:

$$n_h(n_i + 1) + n_0(n_h + 1), \tag{11}$$

where n_i, n_h, and n_0 are respectively the numbers of input, hidden, and output neurons. Note that the additional input for each neuron of the hidden layer and the output layer represents the bias inputs.

Table 1 Overview of the NN structures used for the three problem instances

Data set	Input layer	Hidden layer	Output layer	No. of weights
Cancer1	9	6	2	74
Diabetes1	8	6	2	68
Heart1	35	6	2	230

Columns two, three, and four show the number of neurons of the input, hidden, and output layer, respectively. In the last table column is given the resulting number of NN weights

3.5 Parameter tuning

All our algorithms (with the exception of RS) require certain parameter values to be determined before they can be applied. While algorithms such as BP or LM have very few parameters, $ACO_{\mathbb{R}}$ (as well as its hybridized versions) have more. In general, in order to ensure a fair comparison of algorithms, an equal amount of effort has to be spent in the parameter tuning process for each of them. It has also been shown in the literature that the stopping condition for the parameter tuning runs should be identical to the one used in the actual experiments, as otherwise the danger of choosing suboptimal parameter values increases [34]. We have hence used a common parameter tuning methodology for all our algorithms, with the same stopping condition that we used for the final experiments (that is, 1,000 solution evaluations).

The methodology that we used is known as the F-RACE methodology [4, 5]. In particular we used the RACE package for R. It allows running a race of different configurations of algorithms against each other on a set of test instances. After each round, the non-parametric Friedman test is used to compare the performance of different configurations. Configurations are being dropped from the race as soon as sufficient statistical evidence has been gathered against them. For more information on the F-RACE methodology, we refer the interested reader to [4].

Since we wanted to tune each algorithm for each problem instance seperately, we had to generate from each problem instance an artificial set of tuning instances. This was done by splitting the training set of the first cross-validation experiment (consisting of 75% of the total amount of pattern) into a training set for tuning (two-third of these pattern), and a test set for tuning (one-third of these pattern). Each tuning instance was generated by doing this division randomly.

For the tuning we determined ten different configurations of parameter values for each of our algorithms. Then, we applied F-RACE to each problem instance, allowing not more than 100 experiments in each race. Each of the parameter tuning races returned one configuration that

Table 2 Summary of the final parameter values that we chose for our algorithms

Alg.	Cancer1				Diabetes1				Heart1			
	k	ξ	η	β	k	ξ	η	β	k	ξ	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

Not included in the table are the parameters common to all $ACO_{\mathbb{R}}$ versions, namely q (see Eq. 5) and m (the number of ants per iteration). For these parameters we used the settings $q = 0.01$, and $m = 2$. Note that η is the step-size parameter of BP, and β is the adaptation-step parameter of LM; and remember that k is the archive size of $ACO_{\mathbb{R}}$, while ξ is the parameter that controls the convergence speed of $ACO_{\mathbb{R}}$

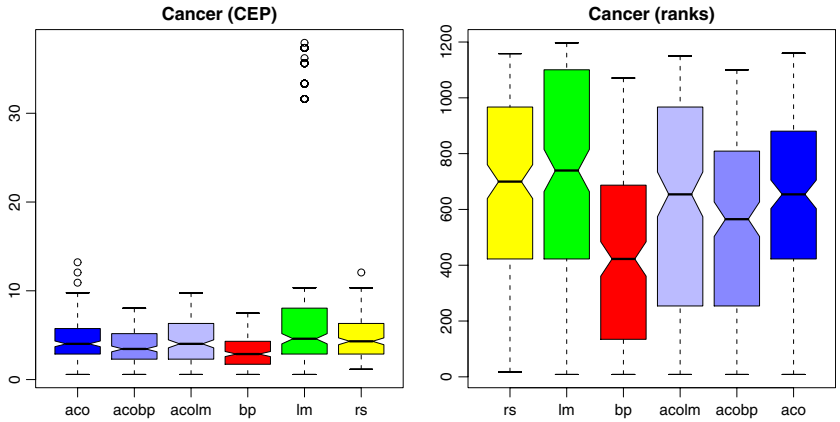

Fig. 6 Box-plots for Cancer1. The *boxes* are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95% confidence interval

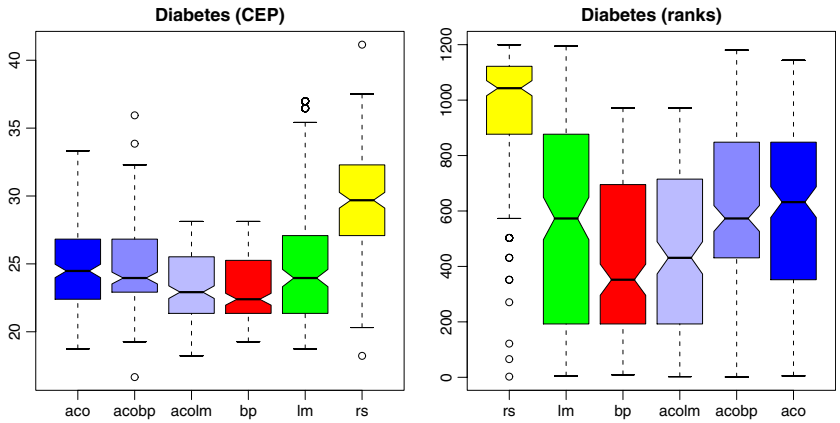

Fig. 7 Box-plots for Diabetes1. The *boxes* are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95% confidence interval

performed best.[6] The final parameter value settings that we used for our final experiments are summarized in Table 2. As a last remark, note that the inclusion of parameter ξ (controlling the convergence speed of $ACO_{\mathbb{R}}$) in the parameter tuning ensures that the $ACO_{\mathbb{R}}$ algorithms converge within the given limit of 1,000 solution evaluations.

[6] Due to the limited resources for tuning, the chosen configuration for each race is not necessarily significantly better than all the others. The limit of 100 experiments per race did sometimes not allow reaching that level of assurance. However, the chosen configuration was definitely not significantly worse than any of the others.

3.6 Results

Each cross-validation experiment was performed by applying each algorithm 50 times to each of the three test problems. Figures 6, 7, and 8 present respectively the results of the fourfold cross-validation obtained for the cancer, diabetes, and heart test problems in the form of box-plots. 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 for a given distribution [25]. Each figure contains two plots. The left plot shows the distributions of the

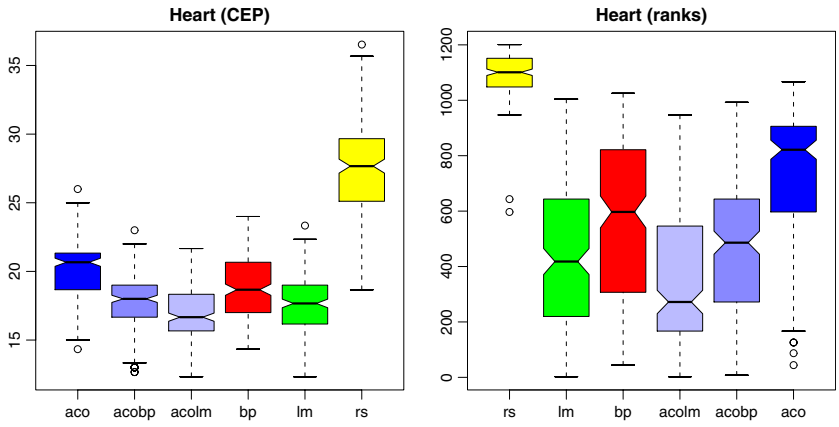

Fig. 8 Box-plots for Heart1. The *boxes* are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95% confidence interval

Table 3 Statistical significance information obtained by applying the pair-wise Wilcoxon rank sum test (with P value adjustments using the Holm method)

	$ACO_{\mathbb{R}}$	$ACO_{\mathbb{R}}$-BP	$ACO_{\mathbb{R}}$-LM	BP	LM
Cancer1					
$ACO_{\mathbb{R}}$-BP	–				
$ACO_{\mathbb{R}}$-LM	–	–			
BP	+	+	+		
LM	+	+	+	+	
RS	+	+	+	+	–
Diabetes1					
$ACO_{\mathbb{R}}$-BP	–				
$ACO_{\mathbb{R}}$-LM	+	+			
BP	+	+	–		
LM	–	–	–	+	
RS	+	+	+	+	+
Heart1					
$ACO_{\mathbb{R}}$-BP	+				
$ACO_{\mathbb{R}}$-LM	+	+			
BP	+	+	+		
LM	+	–	+	+	
RS	+	+	+	+	+

We applied a 5% confidence level and translated the P values accordingly into + (in case a P value is smaller than 0.05), and – (in case a P value is greater than 0.05). Therefore, + means that the corresponding two algorithms are different with statistical significance

actual classification error percentage (CEP) values obtained by the algorithms (for 50 independent runs; and averaged over the results of the fourfold cross-validation). In the right plot, the CEP values were exchanged for the corresponding rank of the result, that is, having 6 algorithms and running 50 trials for each of the 4 cross-validation experiments, the possible result ranks vary from 1 to 1,200. The plots of the rank distributions usually allow for a clearer identification of differences between the algorithms than the plots of the CEP value distributions. Additionally, Table 3 shows information regarding the statistical significance of the algorithms being different to each other.

Cancer1 (see Fig. 6 and Table 3) appears to be a problem instance in which the two different classes can be seperated quite easily. All algorithms obtain reasonably good results, including the RS method. However, the best performing algorithm is BP, followed by the three ACO variants. The two worst performing algorithms are LM and RS. From the fact that the results obtained by RS do not deviate much from the results obtained by other—more complex—algorithms, it may be concluded that there are a lot of reasonably good solutions scattered all 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.

Diabetes1 (see Fig. 7 and Table 3) is a data set that evidently poses more problems than Cancer1. All algorithms clearly outperform RS. In addition, we note that BP, $ACO_{\mathbb{R}}$-LM, and LM are the best-performing algorithms, beating $ACO_{\mathbb{R}}$ and $ACO_{\mathbb{R}}$-BP. In general, the overall performance of the algorithms in terms of the CEP value is relatively poor. This unsatisfactory overall performance of the algorithms may again indicate that the training set does not fully represent all the possible patterns.

The Heart1 problem (see Fig. 8 and Table 3) is—with 230 weights—the largest problem that we tackled. It is also the one on which the performance of the algorithms differed most notably. All tested algorithms clearly outperform RS. In fact, the differences between the algorithms (with the exception of the pair BP/LM) are statistically significant. $ACO_{\mathbb{R}}$-LM is the best-performing algorithm. It is interesting that BP, which seemed to have slight advantages for the first two test problems, did not do so well on Heart1. There might be serveral reasons for that. First, when using the SEP fitness function in combination with sigmoidal neuron transfer functions, the gradient descent weight updates sometimes go to zero for wrong

Table 4 Classification error percentage (CEP) values averaged for each algorithm over all runs for all cross-validation experiments

	Cancer1		Diabetes1		Heart1	
	Training	Test	Training	Test	Training	Test
$ACO_{\mathbb{R}}$	3.05	4.02	23.09	24.48	17.58	20.67
$ACO_{\mathbb{R}}$-BP	1.90	3.45	19.62	23.96	10.97	18
$ACO_{\mathbb{R}}$-LM	1.14	4.02	21.01	22.92	3.55	16.51
BP	2.67	2.87	21.18	22.40	15.65	18.67
LM	1.71	4.60	21.01	23.96	4.03	17.62
RS	4.19	4.31	28.30	29.69	26.52	27.64

The results obtained on the training set in comparison to the results obtained on the test set give information on the overfitting of the algorithms

Table 5 Pair-wise comparison of the results of $ACO_{\mathbb{R}}$ with recent results obtained by a genetic algorithm (see [1])

	GA	$ACO_{\mathbb{R}}$
Cancer1	16.76 (6.15)	**2.39 (1.15)**
Diabetes1	36.46 (0.00)	**25.82 (2.59)**
Heart1	41.50 (14.68)	**21.59 (1.14)**

For each problem-algorithm pair we give the mean of the CEPs (obtained in 50 independent runs), and their standard deviation (in brackets). The best result of each comparison is indicated in bold

outputs. Second, the fact that the Heart1 problem is bigger than the other test problems might increase the probability that the gradient-based algorithms get stuck in local optima of the search landscape. Third, it is known from the *no free lunch* theory (see [37]) that it is not possible to create an algorithm that is best on all possible problem instances. Very interesting is the performance of the hybrid versions of $ACO_{\mathbb{R}}$, namely $ACO_{\mathbb{R}}$-BP and $ACO_{\mathbb{R}}$-LM. The $ACO_{\mathbb{R}}$-BP hybrid outperforms both $ACO_{\mathbb{R}}$ and BP with statistical significane. $ACO_{\mathbb{R}}$-LM outperforms respectively $ACO_{\mathbb{R}}$ and LM with statistical significance.

Summarizing, we note that the performance of $ACO_{\mathbb{R}}$ alone does often not quite reach the performance of the derivative based algorithms and the $ACO_{\mathbb{R}}$ hybrids. Furthermore, the results show that hybridizing $ACO_{\mathbb{R}}$ with BP or LM generally helps to improve the results of the pure $ACO_{\mathbb{R}}$ algorithm. This was especially the case for Heart1, where $ACO_{\mathbb{R}}$-LM was the overall winner. There also seems to be a promising tendency in the results: The bigger the problem instance, the better was the performance of $ACO_{\mathbb{R}}$-LM in comparison to the other algorithms.

Finally, we studied if (and to what extend) the algorithms suffer from overfitting. Table 4 shows for all algorithms the CEP values averaged over all runs and all cross-validation experiments. In general, $ACO_{\mathbb{R}}$, BP, and RS do not suffer from strong overfitting, while $ACO_{\mathbb{R}}$-BP suffers from moderate overfitting on all three test problems. The strongest overfitting effects can be seen in the results of LM and $ACO_{\mathbb{R}}$-LM when applied to problem instances Cancer1 and Heart1. This suggests that a different division of the data set into a training set and a test set might be necessary in this case.

3.7 Comparison to a basic GA

Appart from the comparison to specialized algorithms for NN training, it is also interesting to compare the performance of the $ACO_{\mathbb{R}}$ based algorithms to another general optimizer. The GA-based algorithms from [1] were applied to the same three problem instances used in our study. Appart from a stand-alone GA, the authors of [1] also tested hybrid versions, namely GA-BP and GA-LM. Moreover, the GA-based algorithms were applied with the same stopping criterion (that is, 1,000 function evaluations), and the chosen training/test set division was the same.

Tables 5 and 6 summarize the results obtained by the $ACO_{\mathbb{R}}$ and GA based algorithms.[7] Clearly the stand-alone $ACO_{\mathbb{R}}$ performs better than the stand-alone GA for all the test instances. $ACO_{\mathbb{R}}$-BP and $ACO_{\mathbb{R}}$-LM perform respectively better than GA-BP and GA-LM when applied to the two more difficult problem instances Diabetes1 and Heart1 and worse on Cancer1. For the Heart1 problem instance, 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).

4 Conclusions

We have presented an ant colony optimization algorithm (i.e., $ACO_{\mathbb{R}}$) for continuous optimization and applied this

[7] Note that Alba and Chicano did not perform a fourfold cross-validation. They only performed the first one of our four cross-validation experiments. Therefore, the results of our ACO algorithms in these tables refer to the results of the first of our four cross-validation experiments.

Table 6 Pair-wise comparison of the results of the $ACO_{\mathbb{R}}$ based hybrid algorithms with recent results obtained by two GA-based hybrid algorithms (see [1])

	GA-BP	$ACO_{\mathbb{R}}$-BP	GA-LM	$ACO_{\mathbb{R}}$-LM
Cancer1	**1.43 (4.87)**	2.14 (1.09)	**0.02 (0.11)**	2.08 (0.68)
Diabetes1	36.36 (0.00)	**23.80 (1.73)**	28.29 (1.15)	**24.26 (1.40)**
Heart1	54.30 (20.03)	**18.29 (1.00)**	22.66 (0.82)	**16.53 (1.37)**

For each problem-algorithm pair we give the mean of the CEPs (obtained in 50 independent runs), and their standard deviation (in brackets). The best result of each comparison is indicated in bold

algorithm and its hybrid versions to the training of feed-forward neural networks for pattern classification. The performance of the algorithms was evaluated on real-world test problems and compared to specialized algorithms for feed-forward neural network training (namely Backpropagation and the Levenberg–Marquardt algorithm). In addition we compared our algorithms to another general optimizer, namely a genetic algorithm. The performance of the stand-alone $ACO_{\mathbb{R}}$ was generally worse (with statistical significane) than the performance of specialized algorithms for neural network training. However, the hybrid between $ACO_{\mathbb{R}}$ and the Levenberg-Marquardt algorithm (i.e., $ACO_{\mathbb{R}}$-LM) was especially on the large problem instance able to outperform the backpropagation and the Levenberg-Marquardt algorithms that are traditionally used for neural network training. Finally, when compared to another general-purpose algorithm, namely a genetic algorithm from the literature, the ant colony optimization based algorithms have in general advantages. Further research is needed to see how our algorithms perform on more complex problems, but the initial results are promising.

The current version of $ACO_{\mathbb{R}}$ is still a basic ACO variant. In the future, we plan to focus on adding mechanisms to $ACO_{\mathbb{R}}$ that allow a more efficient exploration of the search space in order to improve the global optimization capabilities of this algorithm. This will hopefully allow us to create an algorithm that achieves results that are at least comparable to the results of state-of-the-art algorithms for training neural networks.

Acknowledgments This work was supported by the Spanish CICYT project OPLINK (grant TIN-2005-08818-C04), and by the *Ramón y Cajal* program of the Spanish Ministry of Science and Technology of which Christian Blum is a 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.

References

1. Alba E, Chicano JF (2004) Training neural networks with GA hybrid algorithms. In: Deb K et al. (ed) Proceedings of the genetic and evolutionary computation conference—GECCO 2004, volume 3102 of Lecture Notes in Computer Science. Springer, Berlin, pp 852–863
2. Alba E, Marti R (eds) (2006) Metaheuristic procedures for training neural networks. Springer, Berlin
3. Bilchev B, Parmee IC (1995) The ant colony metaphor for searching continuous design spaces. In: Proceedings of the AISB workshop on evolutionary computation, volume 993 of Lecture Notes in Computer Science, pp 25–39
4. Birattari M (2005) The problem of tuning metaheuristics as seen from a machine learning perspective. PhD thesis, volume 292 of Dissertationen zur Künstlichen Intelligenz. Akademische Verlagsgesellschaft Aka GmbH, Berlin, Germany
5. Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Langdon WB et al. (eds) Proceedings of the genetic and evolutionary computation conference. Morgan Kaufman, San Francisco, pp 11–18
6. Bishop CM (2005) Neural networks for pattern recognition. MIT Press, Cambridge
7. Blum C, Socha K (2005) Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In: Nedjah N, Mourelle LM, Vellasco MMBR, Abraham A, Köppen M (eds) Proceedings of the Fifth International Conference on Hybrid Intelligent Systems (HIS). IEEE Computer Society, pp 233–238
8. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York
9. Peter AN (2000) Bosman and Dirk Thierens. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In: Pelikan M, Mühlenbein H, Rodriguez AO (eds) Proceedings of OBUPM Workshop at GECCO-2000. Morgan-Kaufmann Publishers, San Francisco, pp 197–200
10. Box GEP, Muller ME (1958) A note on the generation of random normal deviates. Ann Math Stat 29(2):610–611
11. Cotta C, Alba E, Sagarna R, Larrañaga P (2001) Adjusting weights in artificial neural networks using evolutionary algorithms. In: Larrañaga P, Lozano JA (eds) Estimation of distribution algorithms: a new tool for evolutionary computation. Kluwer Academic Publishers, Boston, pp 361–378
12. Deneubourg J-L, Aron S, Goss S, Pasteels J-M (1990) The self-organizing exploratory pattern of the argentine ant. J Insect Behav 3:159–168
13. Dorigo M (1992) Optimization, Learning and Natural Algorithms (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy
14. Dorigo M, Maniezzo V, Colorni A (1996) Ant System: Optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybernetics – Part B 26(1):29–41
15. Dorigo M, Stützle T (2004) Ant Colony Optimization. MIT Press, Cambridge
16. Dréo J, Siarry P (2002) A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In: Dorigo M, Di Caro G, Sampels M (eds) Proceedings of ANTS 2002 – from ant colonies to artificial ants: third international workshop on ant algorithms, vol 2463 of lecture notes in computer science, Springer, Berlin, pp 216–221

17. Garcia Pedrajas N, Hervás Martinez C, Muñoz Pérez J (2003) COVNET: A cooperative coevolutionary model for evolving artificial neural networks. IEEE Trans Neural Networks 14(3):575–596
18. Golub GH, van Loan CF (1989) Matrix computations, 2nd edn. The John Hopkins University Press, Baltimore
19. Guntsch M, Middendorf M (2003) Solving multi-objective permutation problems with population based ACO. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L (eds) Proceedings of the second international conference on evolutionary multi-criterion optimization (EMO 2003), vol 2636 of lecture notes in computer science. Springer, Berlin, pp 464–478
20. Hagan MT, Menhaj MB (1994) Training feedforward networks with the marquardt algorithm. IEEE Trans Neural Netw 5(6):989–993
21. Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159–195
22. Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning. Springer, Berlin
23. Larrañaga P, Lozano JA (eds) (2001) Estimation of distribution algorithms: a new tool for evolutionary computation. Kluwer Academic Publishers, Boston
24. Mandischer M (2002) A comparison of evolution strategies and backpropagation for neural network training. Neurocomputing 42(1):87–117
25. McGill R, Tukey JW, Larsen WA (1978) Variations of box plots. Am Stat 32:12–16
26. Mendes R, Cortez P, Rocha M, Neves J (2002) Particle swarms for feedforward neural network training. In: Proceedings of the 2002 international joint conference on neural networks (IJCNN'02), vol 2. IEEE press, pp 1895–1899
27. Monmarché N, Venturini G, Slimane M (2000) On how pachycondyla apicalis ants suggest a new search algorithm. Future Generation Comput Syst 16:937–946
28. Montana D, Davis L (1989) Training feedforward neural networks using genetic algorithms. In: Proceedings of the eleventh international joint conference on artificial intelligence (IJCAI). Morgan Kaufmann, San Mateo, pp 762–767
29. Prechelt L (1994) Proben1—a set of neural network benchmark problems and benchmarking rules. Technical Report 21, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany
30. Rumelhart D, Hinton G, Williams R (1986) Learning representations by backpropagation errors. Nature 536:323–533
31. Socha K (2004) Extended ACO for continuous and mixed-variable optimization. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T (eds) Proceedings of ANTS 2004 – fourth international workshop on ant algorithms and swarm intelligence. Lecture Notes in Computer Science. Springer, Berlin
32. Socha K, Blum C (2006) Metaheuristic procedures for training neural networks. chapter ant colony optimization. Springer, Berlin (in press)
33. Socha K, Dorigo M (2006) Ant colony optimization for continuous domains. Eur J Oper Res (in press)
34. Socha K (2003) The influence of run-time limits on choosing ant system parameters. In: Cantu-Paz E et al. (eds) Proceedings of GECCO 2003—genetic and evolutionary computation conference, vol 2723 of LNCS. Springer, Berlin, pp 49–60
35. Stanley KO, Miikulainen R (2002) Evolving neural networks through augmenting topologies. Evol Comput 10(2):99–127
36. Stützle T, Hoos HH (2000) $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. Future Generation Computer Systems 16(8):889–914
37. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
38. Yao X (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447