Chapter 8

ANT COLONY OPTIMIZATION

Krzysztof Socha[1] and Christian Blum[2]
[1]*IRIDIA, Université Libre de Bruxelles, Brussels, Belgium;*
[2]*ALBCOM, LSI, Universitat Politècnica de Catalunya, Barcelona Spain*

Abstract: Ant colony optimization (ACO) is a metaheuristic that was originally introduced for solving combinatorial optimization problems. In this chapter we present the general description of ACO, as well as its adaptation for the application to continuous optimization problems. We apply this adaptation of ACO to optimize the weights of feed-forward neural networks for the purpose of pattern classification. As test problems we choose three data sets from the well-known PROBEN1 medical database. The experimental results show that our algorithm is comparable to specialized algorithms for feed-forward neural network training. Furthermore, the results compare favourably to the results of other general-purpose methods such as genetic algorithms.

Key words: Ant colony optimization, continuous optimization, pattern classification, feed-forward neural network training.

1. INTRODUCTION

In the early 90's, ant colony optimization (ACO) (Dorigo, 1992; Dorigo *et al.*, 1991; Dorigo *et al.*, 1996) was introduced as a novel nature-inspired metaheuristic for solving hard combinatorial optimization (CO) problems. According to Papadimitriou and Steiglitz (1982), a CO problem $P=(S,f)$ is an optimization problem in which there is given a finite set of solutions S (also called *search space*) and an objective function $f: S \rightarrow R^+$ [2] that assigns a positive cost value to each of the solutions. The goal is either to find a

[2] R^+ denotes the space of nonnegative real values.

solution of minimum cost value[3], or—as in the case of approximate solution techniques—a good enough solution in a reasonable amount of time. ACO algorithms follow—as do all metaheuristics—the latter goal.

Examples of metaheuristics other than ACO are tabu search (Glover, 1989; Glover, 1990; Glover and Laguna, 1997), simulated annealing (Kirkpatrick *et al.*, 1983; Černý, 1985), and evolutionary computation (Fogel *et al.*, 1966; Rechenberg, 1973; Holland, 1975; Goldberg, 1989). For more general literature on metaheuristics, see (Glover and Kochenberger, 2002; Blum and Roli, 2003; Hoos and Stützle, 2004). Some metaheuristics have been designed with continuous optimization in mind, e.g. evolutionary strategies (ES). However, most metaheuristics were originally introduced to solve CO problems, and only recently their adaptation to solve continuous optimization problems enjoys increasing attention. In continuous optimization we generally want to find a vector $x^* \in K$ (where K is an n-dimensional subspace of R^n) such that $f(x^*) \leq f(x)$, for all $x \in K$ and $f : K \rightarrow R^+$. Examples of such adapted metaheuristics are simulated annealing algorithms (Siarry *et al.*, 1997), or differential evolution (Storn and Price, 1997; Chelouah and Siarry, 2000; Chelouah and Siarry, 2003) from the evolutionary computation field. Among the CO oriented metaheuristics, tabu search algorithms as, for example, (Battiti and Tecchioli, 1996) were among the first to be applied to continuous problems. Some of the above cited methods are hybrids that make use of the well-known Nelder-Mead simplex algorithm for continuous optimization (Nelder and Mead, 1965).

Among the most recent applications of metaheuristics to continuous problems are ant-related algorithms (Mathur *et al.*, 2000; Dréo and Siarry, 2002; Socha, 2004). Only the most recent one of these attempts – namely (Socha, 2004) – can be labelled ant colony optimization. The other attempts rather loosely follow the idea of ant-based algorithms. In this chapter we first present the general idea of ACO. Then, we describe the attempts of creating ant algorithms for continuous optimization. Finally, we describe our approach, which is an extended version of the algorithm proposed by Socha (2004). We apply this algorithm to the training of feed-forward neural networks (NNs). To test our algorithm we apply it to three benchmark classification problems from the medical field: the diagnosis of breast cancer, the diagnosis of diabetes, and the diagnosis of heart disease.

[3] Note that minimizing over an objective function f is the same as maximizing over $-f$. Therefore, every CO problem can be described as a minimization problem.

The remaining part of this chapter is organized as follows. Section 2 presents the ant colony optimization metaheuristic. In particular, subsection 2.1 gives an overview of the basics of ACO for combinatorial optimization, and subsection 2.2 summarizes the existing approaches of applying ant algorithms to continuous optimization problems. Section 3 describes our ACO algorithm for continuous optimization, henceforth denoted by $ACO_{\mathbb{R}}$. Following this, Section 4 presents in detail the specific problem of training feed-forward NNs, the experimental setup, and the results obtained. Finally, Section 5 concludes the chapter.

2. ANT COLONY OPTIMIZATION: THE GENERAL IDEA

ACO algorithms have their origins in a field known as swarm intelligence (SI) (Bonabeau *et al.*, 1999). SI algorithms take their inspiration from the collective behaviour of, for example, social insects, flocks of birds, or fish schools. Examples include algorithms for clustering and data mining inspired by ants' cemetery building behaviour, dynamic task allocation algorithms inspired by the behaviour of wasp colonies, particle swarm optimization (PSO) algorithms, and many more. The inspiring source of ACO is the foraging behaviour of real ants. When searching for food, ants initially explore the area surrounding their nest in a random manner. 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 ant deposits a chemical pheromone trail on the ground. The quantity of pheromone deposited, which may depend on the quantity and quality of the food, will guide other ants to the food source. As it has been shown in (Deneubourg *et al.*, 1990), indirect communication between the ants via pheromone trails enables them to find shortest paths between their nest and food sources. This characteristic of real ant colonies is exploited in artificial ant colonies in order to solve optimization problems.

```
while termination conditions not met do
   ScheduleActivities
      AntBasedSolutionConstruction()
      PheromoneUpdate()
      DaemonActions()          {optional}
   end ScheduleActivities
end while
```

Figure 8-1. Framework of the ant colony optimization (ACO) metaheuristic.

2.1 Ant Colony Optimization for CO Problems

The central component of an ACO algorithm is a parameterized probabilistic model, which is called the *pheromone model.* The pheromone model consists of a vector of model parameters T called *pheromone trail parameters.* The pheromone trail parameters $T_i \in T$, which are usually associated with components of solutions, have values τ_i, called *pheromone values.* The pheromone model is used to probabilistically generate solutions to the problem under consideration by assembling them from a finite set of solution components. At run-time, ACO algorithms update the pheromone values using previously generated solutions. The update aims to concentrate the search in regions of the search space containing high quality solutions. In particular, the reinforcement of solution components depending on the solution quality is an important ingredient of ACO algorithms. It implicitly assumes that good solutions consist of good solution components. To learn which components contribute to good solutions can help assembling them into better solutions. In general, the ACO approach attempts to solve an optimization problem by repeating the following two steps:

- candidate solutions are constructed using a pheromone model, that is, a parameterized probability distribution over the solution space;
- the candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

The ACO metaheuristic framework is shown in Figure 8-1. It consists of three algorithmic components that are gathered in the ScheduleActivities construct. The ScheduleActivities construct does not specify how these three activities are scheduled and synchronized. This is up to the algorithm designer. In the following we explain these three algorithm components in more detail.

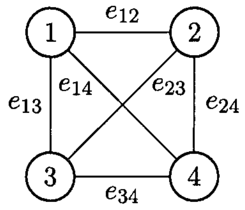

Figure 8-2. A small TSP problem instance on 4 cities in form of a graph $G=(V,E)$. The edges e_{ij} that connect the cities have associated distances d_{ij}.

AntBasedSolutionConstruction(): Artificial ants are constructive heuristics that assemble solutions as sequences of solution components taken from a finite set of solution components $C=\{c_1,\ldots,c_n\}$. Solution construction starts with an empty partial solution $s^p=<>$. Then, at each construction step the current partial solution s^p is extended by adding a feasible solution component from the set $N(s^p) \in C \setminus s^p$, which is defined by the solution construction mechanism. The process of constructing solutions can be regarded as a walk (or a path) on the so-called *construction graph* (see Figure 8-3 for an example) $G_C=(C,L)$ whose vertices are the solution components C and the set L are the connections. The allowed walks on G_C are hereby implicitly defined by the solution construction mechanism that defines set $N(s^p)$ with respect to a partial solution s^p. The choice of a solution component from $N(s^p)$ is at each construction step done probabilistically with respect to the pheromone model T, which consists of *pheromone trail parameters* $T_i \in T$ that are associated to components $c_i \in C$.[4] The values of these parameters—the *pheromone values*—are denoted by τ_i. In most ACO algorithms the probabilities for choosing the next solution component—also called the *transition probabilities*—are defined as follows:

$$p(c_i \mid s^p) = \frac{\tau_i^{\alpha} \cdot \eta(c_i)^{\beta}}{\sum\limits_{c_j \in N(s^p)} \tau_j^{\alpha} \cdot \eta(c_j)^{\beta}} \quad , \forall c_i \in N(s^p) \quad , \tag{1}$$

where η is a weighting function, which is a function that, sometimes depending on the current partial solution, assigns at each construction step a heuristic value $\eta(c_i)$ to each feasible solution component $c_i \in N(s^p)$. The values that are given by the weighting function are commonly called the *heuristic information*. Furthermore, α and β are positive parameters whose values determine the relation between pheromone information and heuristic information. In Figure 8-3 is given an example of the solution construction for the small travelling salesman (TSP) problem instance that is shown is given in Figure 8-2.

[4] Note that the description of the ACO metaheuristic as given for example in (Dorigo and Stützle, 2004) allows also connections of the construction graph to be associated with a pheromone trail parameter. However, for the purpose of this introduction it is sufficient to assume that pheromone trail parameters are associated with components.

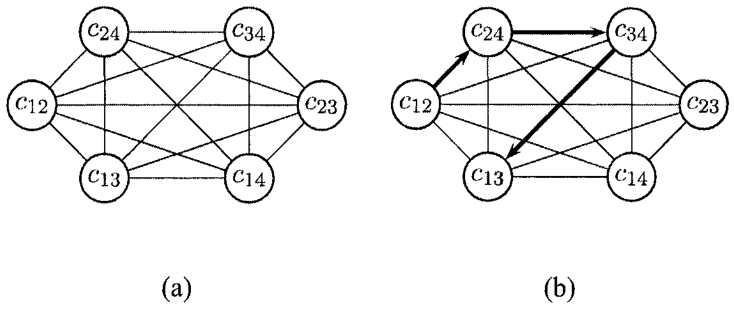

Figure 8-3. (a) shows the construction graph $G_C=(C,L)$ with respect to the TSP instance shown in Figure 8-2. The set of solution components C consists of a solution component c_{ij} for each edge e_{ij}, and the pheromone model consists of a pheromone trail parameter T_{ij} for each solution component c_{ij}. As heuristic information we choose the inverse of the distances between the cities. Therefore, it holds that $\eta(c_{ij})=1/d_{ij}$ for all c_{ij}. The construction mechanism is as follows. In the first construction step, any solution component can be chosen. For the remaining construction steps the set of solution components is restricted so that the sequence of solution components always corresponds to a path in G (respectively, to a Hamiltonian cycle in G after the last construction step). (b) shows a path on the construction graph that corresponds to the construction of solution $s=<c_{12},c_{24},c_{34},c_{13}>$.

PheromoneUpdate(): Different ACO algorithms—such as, for example, Ant Colony System (ACS) (Dorigo and Gambardella, 1997) and *MAX–MIN* Ant System (*MM*AS) (Stützle and Hoos, 2000)—mainly differ in the update of the pheromone values they apply. In the following, we outline a common pheromone update rule in order to provide the general idea. This pheromone update consists of two parts. First, a *pheromone evaporation*, which proportionally decreases all the pheromone values, is performed. From a practical point of view, pheromone evaporation is needed to avoid an overly rapid convergence of the algorithm toward a sub-optimal region. It implements a useful form of *forgetting*, favouring the exploration of new areas in the search space. Second, one or more solutions from the current and/or from earlier iterations are used to increase the values of pheromone trail parameters on solution components that are part of these solutions:

$$\tau_i \leftarrow (1-\rho)\tau_i + \rho \sum_{\{s \in S_{upd} | c_i \in s\}} F(s) \quad , \qquad (2)$$

for $i=1,...,n$. Here, S_{upd} is the set of solutions that are used for the update. Furthermore, $\rho \in (0,1]$ is a parameter called evaporation rate, and $F : S \rightarrow R^+$ is a function such that $f(s) < f(s') \Rightarrow F(s) \geq F(s'), \ \forall s \neq s' \in S$. $F(\cdot)$ is commonly called the *quality function*. Instantiations of this update rule are obtained by different specifications of S_{upd}, which—in many cases—is a subset of $S_{iter} \cup \{s_{bs}\}$, where S_{iter} is the set of solutions that were constructed in the current iteration, and s_{bs} is the best-so-far solution, that is, the best solution found since the first algorithm iteration. A well-known example is the *AS-update* rule, that is, the update rule of Ant System (AS) (Dorigo *et al.*, 1996). The AS-update rule is obtained from update rule 2 by setting

$$S_{upd} \leftarrow S_{iter} \quad .$$

This update rule is well-known due to the fact that AS was the first ACO algorithm to be proposed in the literature. An example of a pheromone update rule that is more used in practice is the *IB-update* rule (where IB stands for *iteration-best*). The IB-update rule is given by:

$$S_{upd} \leftarrow \arg\max\{F(s) \mid s \in S_{iter}\}$$

The IB-update rule introduces a much stronger bias towards the good solutions found than the AS-update rule. However, this increases the danger of premature convergence. An even stronger bias is introduced by the *BS-update* rule, where BS refers to the use of the best-so-far solution s_{bs}. In this case, S_{upd} is set to $\{s_{bs}\}$. In practice, ACO algorithms that use variations of the IB-update or the BS-update rule include mechanisms to avoid premature convergence and tend to achieve better results than algorithms that use the AS-update rule. Examples are ACS and *MM*AS as mentioned above.

DaemonActions(): Daemon actions can be used to implement centralized actions which cannot be performed by single ants. Examples are the application of local search methods to the constructed solutions, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon may decide to deposit extra pheromone on the solution components that belong to the best solution found so far.

After the initial proof-of-concept application to the travelling salesman problem (TSP) (Dorigo *et al.*, 1991; Dorigo *et al.*, 1996), ACO was applied to many other CO problems. Examples include the applications to assignment problems (Costa and Hertz, 1997; Maniezzo and Colorni, 1999; Maniezzo, 1999; Socha et al., 2003; Stützle and Hoos, 2000), scheduling problems (Stützle, 1998; den Besten *et al.*, 2000; Gagné *et al.*, 2002; Merkle *et al.*, 2002; Blum and Sampels, 2004), and vehicle routing problems (Gambardella *et al.*, 1999; Reimann *et al.*, 2004). Among other applications, ACO algorithms are currently state-of-the-art for solving the sequential ordering problem (SOP) (Gambardella and Dorigo, 2000), the resource constraint project scheduling (RCPS) problem (Merkle *et al.*, 2002), and the open shop scheduling (OSS) problem (Blum, 2005). For an overview of applications of ACO we refer the interested reader to (Dorigo and Stützle, 2004).

2.2 Ant-Related Algorithms for Continuous Optimization

As indicated in the previous section, ACO has proven to be an efficient and versatile tool for solving various combinatorial optimization problems. In recent years some attempts were also made to use them for tackling continuous optimization problems. However, a direct application of the ACO metaheuristic to continuous domains is not straightforward. Hence, the early proposals often drew inspiration from ACO, but they did not follow exactly the same methodology. This has changed since the publication of the work by Socha (2004), which is the first real ACO algorithm for continuous optimization. In the following paragraphs we highlight the main characteristics of the various ant algorithms proposed for continuous domains.

One of the first attempts to apply ant-based ideas to continuous optimization problems was Continuous ACO (CACO) (Bilchev and Parmée, 1995). In CACO the ants start from a *nest* situated somewhere in the search space. The artificial pheromone information is kept as numerical values that are each assigned to a vector. At each iteration of the algorithm, the ants choose probabilistically the vector from which they then continue the search. This is followed by some random moves. The vectors are updated with the best results found. Although the authors of CACO state to have taken the inspiration for their algorithm from the original ACO formulation, there are some important differences. There is a new notion of the *nest* introduced, which does not exist in the ACO metaheuristic. Also, CACO does not perform an incremental construction of solutions, which is one of the main characteristics of the ACO metaheuristic.

Another ant-related approach to continuous optimization is the API algorithm (Monmarché *et al.*, 2000). The API algorithm does not use any artificial pheromone information, neither do the authors claim that API is based on the ACO metaheuristic. In API, the ants perform their search independently, but starting from the same nest, which is periodically moved to other locations. The ants use a mechanism labelled *tandem running* as a type of recruitment strategy. API is the only ant algorithm published so far–apart from the later work of Socha (2004)—that allows tackling both discrete and continuous optimization problems.

The third ant-based approach to continuous optimization is Continuous Interacting Ant Colony (CIAC) (Dréo and Siarry, 2002). In contrast to a standard ACO algorithm, CIAC uses two types of communication between ants: artificial pheromone information (i.e., spots of pheromone deposited in the search space) and direct communication between ants. Furthermore, CIAC is not performing an incremental construction of solutions, which is a vital part of standard ACO algorithms.

Finally, Socha (2004) proposed ACO*, an ACO algorithm for continuous domains that follows closely the spirit of ACO algorithms for CO problems. In particular, ACO* is also based on step-by-step construction of solutions. The main idea of this algorithm is as follows. In ACO algorithms for CO problems, each solution construction step concerns the (probabilistic) choice of a solution component from a set of allowed solution components (see Section 2.2). Hereby, each solution component corresponds to the assignment of a certain value to one of the decision variables. The choice of a solution component is at each step performed probabilistically according to Equation 1, which defines a discrete probability distribution. In contrast, in ACO* the construction of a solution works by choosing for each of the continuous variables a domain value by sampling a so-called probability density function. Therefore, the main idea of ACO* is to replace discrete distributions with continuous distributions (see Figure 8-4). In that sense, ACO* is closely related to so-called estimation of distribution algorithms (EDAs). See, for example, (Kern et al., 2004).

3. ACO$_{\mathbb{R}}$ FOR CONTINUOUS OPTIMIZATION

In this section, we present the ACO$_{\mathbb{R}}$ algorithm—an extension of the idea of ACO* (Socha, 2004). In the following we assume to tackle a continuous optimization problem of the following form. Given are *n* decision variables $X=\{X_1,\dots,X_n\}$ with continuous domains $\{D_1,\dots,D_n\}$. For constructing a solution, the algorithm chooses for each of the *n* decision variables a domain value by sampling probability density functions (PDFs).

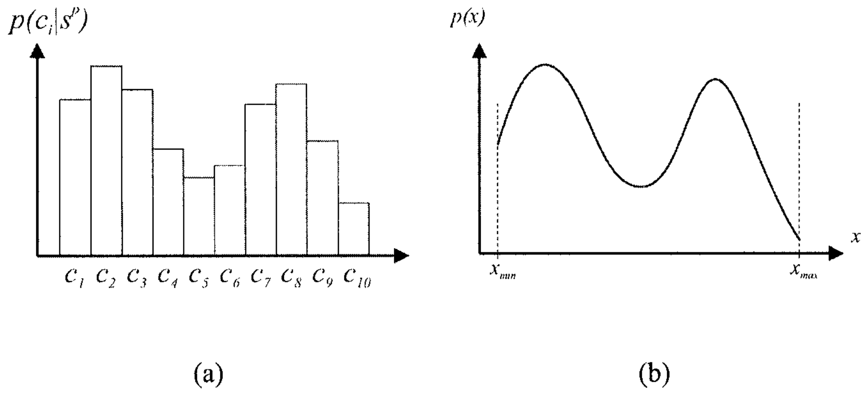

Figure 8-4. (a) The discrete probability distribution that is employed at each construction step when combinatorial problems are concerned, and (b) a continuous probability density function for continuous problems. Hereby, x_{min} and x_{max} respresent the minimal, respectively the maximal allowed domain values.

These PDFs are, for each solution construction (and for each decision variable), 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 randomly generated solutions. At each iteration a set of m solutions is generated by m ants and added to P. Then, the solutions of this extended population are ranked according to their objective function values, and the m worst solutions are removed. This mechanism biases the search process towards the best solutions found during the search process, and keeps the population size fixed to m at all times. Note that the population P of solutions takes over the role of the pheromone information that is used in ACO algorithms for CO problems as a storage of search experience. A similar approach has been used before by Guntsch and Middendorf (2002) in case of Population-Based ACO for CO problems.

For constructing a solution, an ant acts as follows. First, it transforms the original set of decision variables X into a set of temporary variables $Z=\{Z_1,\ldots,Z_n\}$. 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 P 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.

At each of the n construction steps $i=1,...,n$, the ant chooses a value for the corresponding decision variable Z_i. For performing this choice it uses a so-called Gaussian kernel PDF, which is a weighted superposition of several Gaussian functions. For a decision variable Z_i the corresponding Gaussian kernel G_i is given as follows:

$$G_i(z) = \sum_{j=1}^{k} \omega_j \cdot g_j(z) = \sum_{j=1}^{k} \omega_j \cdot \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{(z-\mu_j)^2}{2\sigma_j^2}} \quad \text{, for all } z \in R \quad (3)$$

where the j-th Gaussian function g_j is derived from the j-th member of population P. Remember that k is the number of Gaussian functions composing the Gaussian kernel PDF. Note that $\boldsymbol{\omega}$, $\boldsymbol{\mu}$, and $\boldsymbol{\sigma}$ are vectors of size k. $\boldsymbol{\omega}$ is the vector of weights, whereas $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the vectors of means and standard deviations respectively. Figure 8-5 presents an example of a Gaussian kernel PDF consisting of five separate Gaussian functions.

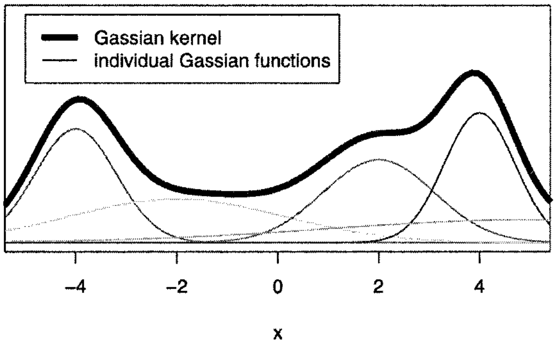

Figure 8-5. An example of a Gaussian kernel PDF consisting of five separate Gaussian functions.

Sampling directly the Gaussian kernel PDF as defined in Equation (3) is problematic. It can, however, be accomplished by the following procedure, which can be proven to be equivalent to sampling G_i directly.

Before starting the construction of a solution, each ant chooses exactly one of the k Gaussian functions, and uses this Gaussian function, henceforth denoted by g_{j^*}, for all n construction steps. The Gaussian function g_{j^*} is chosen with following probability distribution:

$$p_j = \frac{\omega_j}{\sum_{l=1}^{k} \omega_l} \quad , \tag{4}$$

where ω_j is the weight of the j-th Gaussian function g_j, which is obtained as follows. All solutions in P are ranked with respect to their quality; 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:

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

which essentially defines the weight to be a value of the Gaussian function with the argument r, with mean 1.0 and standard deviation qk, where q is also 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.

The sampling of the chosen Gaussian function g_{j*} may be done using a random number generator that is able to generate random numbers according to a parameterized normal distribution, or by using a uniform random generator in conjunction with (for instance) the Box-Muller method (Box and Muller, 1958). However, before doing so the mean μ_{j*} and the standard deviation σ_{j*} of the chosen Gaussian function g_{j*} have to be determined. As mean μ_{j*} we choose the value of the i-th decision variable in the j^*-th solution. It remains to specify the standard deviation σ_{j*}. In order to establish the value of this standard deviation we calculate the average distance of the other population members from the j^*-th solution (in dimension i) and multiply it by the parameter ρ, which determines the speed of convergence:

$$\sigma_{j*} = \frac{1}{k-1} \rho \sum_{l=1}^{k} | z_i^l - z_i^{j*} | \tag{6}$$

Parameter ρ has a role similar to the pheromone evaporation rate ρ in ACO for CO problems. The higher the value of ρ>0, 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., each 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 convergence, but also allows the handling of problems that are scaled differently in different directions.

Next, we describe how the set of temporary decision variables $\boldsymbol{Z}$ is created from the original set X. 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 pheromone is represented (i.e., as the population of solutions), it is in fact possible to take into account the correlation between the decision variables. An obvious choice for adapting the coordinate system to the distribution of population P is the Principal Component Analysis (PCA) (Hastie *et al.*, 2001). 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 new direction. This direction is chosen by randomly selecting a solution s_u from P that is reasonably far away from the j^*-th solution chosen for defining the Gaussian function g_{j^*}. Then, the vector from the u-th solution to the j^*-th solution becomes the new direction. The probability of choosing the u-th solution is the following:

$$p(s_u \mid s_{j^*}) = \frac{d(s_u, s_{j^*})^4}{\sum_{l=1}^{k} d(s_l, s_{j^*})^4} \quad , \tag{7}$$

where function $d(.,.)$ returns the distance between two members of the population P. Once this new direction is chosen, the new orthogonal basis for the ant's coordinate system is created using the Gram-Schmidt process (Golub and van Loan, 1989). It takes as input all the (already orthogonal) directions chosen in earlier construction 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 $\boldsymbol{Z}$. 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 $\boldsymbol{Z}$ are converted back into the original coordinate system X.

Finally, we deal with the subject of constraint handling. Note that the way of generating new solutions as explained above, might lead to unfeasible solutions. In the literature on evolutionary algorithms we find several ways to deal with unfeasible solutions, including rejection, repair, or penalization. All these methods can also be applied in $ACO_{\mathbb{R}}$.

4. EXPERIMENTAL SETUP AND RESULTS

After the presentation of $ACO_{\mathbb{R}}$ we will now outline in detail the application of $ACO_{\mathbb{R}}$ to the training of feed-forward NNs for the purpose of pattern classification, as well as the experimental setup and the results that we obtained.

4.1 The Problem

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[5] benchmark set (Prechelt, 1994), 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.

Feed-forward neural networks (NNs) are popular classification tools. Each feed-forward NN consists of an input layer of neurons. In case of the classification problem the input layer consists of as many neurons as there are measurements in the patterns, that is, for each measurement there exists exactly one input neuron. Furthermore, a feed-forward NN consists of an arbitrary number of hidden layers of neurons, and an output layer (for an example, see Figure 8-6). The output layer consists of as many neurons as the data set has classes. In our case, the output layer consists of 2 output 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,

[5] It is available online at: ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz.

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 (winner takes all).

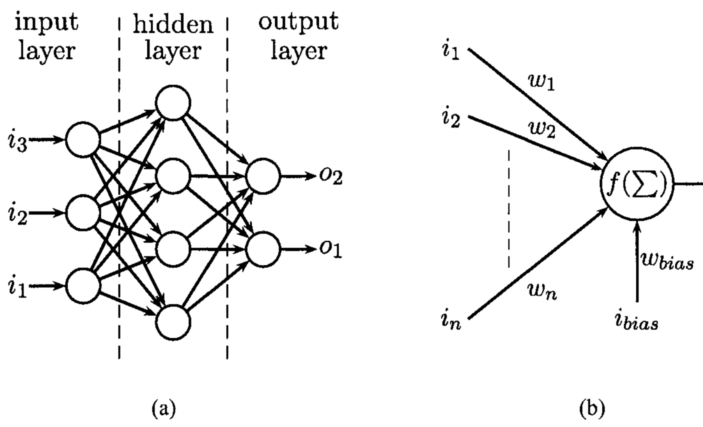

Figure 8-6. (a) shows a feed-forward NN with one hidden layer. Note that all the neurons of each layer are connected to all the neurons of the next layer. (b) shows one single neuron (from either the hidden layer, or the output layer). The neuron receives inputs (i.e., signals i_l, weighted by weights 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()* 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 and output signals, biases, and weights are real values.

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.

Concerning the hidden neuron layers of the feed-forward NNs that we used, we took inspiration from the literature. More specifically we used the same structure of hidden layers that were used in (Alba and Chicano, 2004). For an overview of the feed-forward NNs that we used see Table 8-1. The number of weights to be optimized is—for each of the three data sets—given by the following formula:

$$n_h(n_i + 1) + n_o(n_h + 1) \quad , \tag{8}$$

where n_i, n_h, and n_o 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. The last column of Table 8-1 provides the number of weights to be optimized.

Table 8-1. Summary of the feed-forward NNs that we used for the three data sets. In the last table column is given the number of weights to be optimized for each tackled problem

Problem	*Input Layer(n_i)*	*Hidden Layer (n_h)*	*Output Layer (n_o)*	*Weights*
Cancer1	9	6	2	74
Diabetes1	8	6	2	68
Heart1	35	6	2	230

Note that the training of a feed-forward NN is an unconstrained continuous optimization problem (i.e, the domains of the decision variables are unconstrained). Remember that at the start of our algorithm, the population P of solutions is initialized by uniform random sampling. Considering the neuron transfer function that we used (i.e., the sigmoid function), the primary influence of a weight comes from its sign rather than its value. Hence, we restricted the random sampling for generating the initial solution to the interval *[-1,1]*.[6]

4.2 Training and Solution Evaluation

Neural networks for pattern classification are usually expected to exhibit a generalization capability, that is, new patterns that were not used for the training of the neural network should also be classified correctly. Having this objective in mind, the training of neural network classifiers works generally as follows. First, the set of patterns is divided into training set and test set. In our case we chose randomly 75% of all available patterns of a problem as training set (denoted by P^*), and the remaining 25% of the patterns as test set (denoted by P').

For the training of the weights of a feed-forward NN, a function is needed that distinguishes between different solutions. In other words, we need a function that measures the classification power of a solution, that is, a weight setting, with respect to the training set. For this purpose, we have used the function that is routinely used for this purpose, namely the Square Error Percentage (SEP):

[6] This restriction applies only to the initial interval used. During the search, the $ACO_{\mathbb{R}}$ algorithm can sample values outside of this initial interval.

$$SEP = 100 \frac{o_{\max} - o_{\min}}{n_o \, | P^* |} \sum_{p=1}^{|P^*|} \sum_{i=1}^{n_o} \left(t_i^p - o_i^p \right)^2 \quad , \tag{9}$$

where o_{max} and o_{min} are respectively the maximum and minimum values of the output signals of the output neurons (depeding on the neuron transfer function), n_0 is the number of output neurons, and $t^p{}_i$ and $o^p{}_i$ represent respectively the expected and actual values of output neuron i for pattern p.

Finally, in order to assess the quality of the final solution found by a given algorithm, we used the Classification Error Percentage (CEP) as the performance measure. CEP represents the percentage of incorrectly classified patterns from the test set.

4.3 Algorithms Used for Comparison

The goal of our experimentation was to evaluate whether ACO$_{\mathbb{R}}$ may be used for training feed-forward NNs, and if so, we were interested in how it would compare to other algorithms. In order to be able to draw any meaningful conclusions, it is required to have some reference algorithm to which to compare the performance of ACO$_{\mathbb{R}}$. In order to ensure a fair comparison, we have re-implemented some algorithms traditionally used for training NNs—namely the back-propagation (BP) algorithm and the Levenberg-Marquardt (LM) algorithm. We used the R programming language (a free alternative to S+) for implementing these algorithms.[7]

Back-Propagation is a gradient-descent algorithm traditionally used for training NNs (Rumelhart *et al.*, 1986). It is a first-order minimization algorithm—i.e. it is based on first-order derivatives (i.e., the gradient). It uses the estimation of the gradient of the instantaneous sum-squared error for each network layer:

$$\Delta \vec{w} = -\eta \nabla E(\vec{w}) \quad , \tag{10}$$

where $\vec{w}$ is the vector of all weights, η is the learning rate, and E is the gradient. The algorithm we have implemented is the basic version of back-propagation without heuristic improvements that were developed over time.

Levenberg-Marquardt is a variation of Newton's method that was initially designed for minimizing functions that are either sums of squares,

[7] http://www.R-project.org

or, in general, other non-linear functions (Hagan and Menhaj, 1994; Papliński, 2004). In Newton's method, minimization is based on utilizing the second order derivatives as well as on the use of a batch training mode rather than the pattern mode (which is used, for example, in back-propagation). The batch training mode is based on derivatives of instantaneous errors. The LM algorithm uses an approximation of the Hessian matrix by adding a small constant μ multiplied by the identity matrix I to the product of the transposed Jacobian matrix J^T and the Jacobian matrix J:

$$\Delta\vec{w} = -\sum_{1}^{P} \nabla E(w)\left[J(w)^T J(w) + \mu I\right]^{-1} \qquad (11)$$

Both algorithms (i.e., BP and LM) require gradient information. Hence, they require the neuron transfer function to be differentiable. Consequently, these algorithms may not be used in case, when the neuron transfer function is not differentiable or is unknown. In contrast, $ACO_{\mathbb{R}}$ is a general heuristic optimization that can be applied when the neuron transfer function is non-differentiable. On the other side, in case, when the neuron transfer function is differentiable, the drawback of general optimization algorithms such as $ACO_{\mathbb{R}}$ is that they do not exploit available additional information as, for example, gradient information.

In order to see how the additional gradient information influences the performance of $ACO_{\mathbb{R}}$, we have also implemented hybridized versions of $ACO_{\mathbb{R}}$, namely $ACO_{\mathbb{R}}$-BP and $ACO_{\mathbb{R}}$-LM, which are hybrids of the $ACO_{\mathbb{R}}$ algorithm and respectively the BP and LM algorithms. In these hybrids, each solution generated by the $ACO_{\mathbb{R}}$ algorithm is improved by running a single improving iteration of either BP or LM, respectively.

Finally, we wanted to study how all the algorithms tested compare to a simple random restart search method. In order to accomplish that, we have implemented random search (RS)—i.e. 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].

4.4 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 is required in the parameter tuning process for each of the algorithms. Also, it has 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 (be that time, number of iterations, etc.), as otherwise the danger of choosing suboptimal parameter values increases (Socha, 2003). We have hence used a common parameter tuning methodology for all our algorithms, with the same stopping condition that we planned to use for the final experiments. The methodology that we used is known as F-RACE methodology (Birattari *et al.*, 2002; Birattari, 2004). In particular we used the RACE package[8] 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 (Birattari, 2004). Since for the problems we investigated we did not have several instances available (i.e., we wanted to tune the algorithms for each of the three considered data sets separately), we have created a set of instances for each race by dividing randomly (several times) the training set of each problem instance into a training set for tuning (two thirds of the training set) and a test set for tuning (one third of the training set). Table 8-2 provides details on the number of patterns used respectively for learning and validation during the parameter tuning runs, as well as for training and testing the chosen configurations.

Table 8-2. Summary of the number of patterns used for training and testing, both for parameter tuning as well as for the final performance evaluation. The patterns used for parameter tuning (learning and testing) were randomly chosen from the training set that we used later in the performance evaluation

Algorithm	*Total number of patterns*	*Parameter tuning*		*Performance evaluation*	
		Training set for tuning	*Test set for tuning*	*Training Set*	*Test Set*
Cancer1	699	350	175	525	174
Diabetes1	768	384	192	576	192
Heart1	920	460	230	690	230

For the tuning, we determined 10 different configurations of parameter settings for each of our algorithms. Then, we applied the F-RACE to each instance set (i.e., per algorithm, per problem), allowing not more than 100

[8] http://cran.r-project.org/src/contrib/Descriptions/race.html

experiments in the race. Each of the parameter tuning races returned one configuration that performed best[9]. The final parameter value settings that we used for our final experiments are summarized in Table 8-3.

Table 8-3. Summary of the parameters chosen for our algorithms. Not included in the table are the parameters common to all $ACO_{\mathbb{R}}$ 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)

	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

4.5 Results

In order to compare the performance of the algorithms, we applied each algorithm 50 times to each of the three test problems. As stopping condition we used the number of fitness function evaluations. Following the work of Alba and Chicano (2004), we used 1000 function evaluations as the limit. We used the training and testing approach—no cross-validation.

Figures 8-7, 8-8, and 8-9 present respectively the results obtained for the cancer, diabetes, and heart test problems in the form of box-plots. Each figure contains two graphics; the left one presents the distributions of the actual CEP values obtained by the algorithms (over 50 independent runs); the right one presents the distributions of rankings achiseved by the algorithms. Any solution generated by any of the algorithms is ranked. Having 6 algorithms and running 50 trials each, the possible rankings vary from 1 to 300. The distribution of those rankings is then plotted per algorithm—this allows for a clear identification of those better performing ones, regardless of how small the difference may be in terms of objective function value. 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 (McGill et. al, 1978). In other words, this means that if the notches of two distributions do not overlap, they are significantly different with 95% confidence.

[9] 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.

Cancer (Fig. 8-7) appears to be the easiest problem among the three. 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.

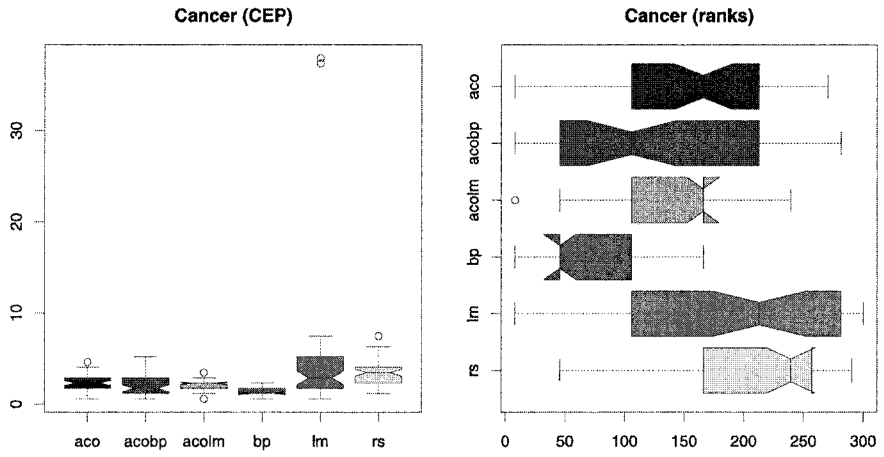

Figure 8-7. Performance comparison of the algorithms on the Cancer1 problem. The graphic on the left represents the actual CEP values, while the right one represents the ranks among all the solutions generated.

Diabetes (Fig. 8-8) is a problem that is more difficult than Cancer. All our algorithms clearly outperform RS. However, the overall performance of the algorithms 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.

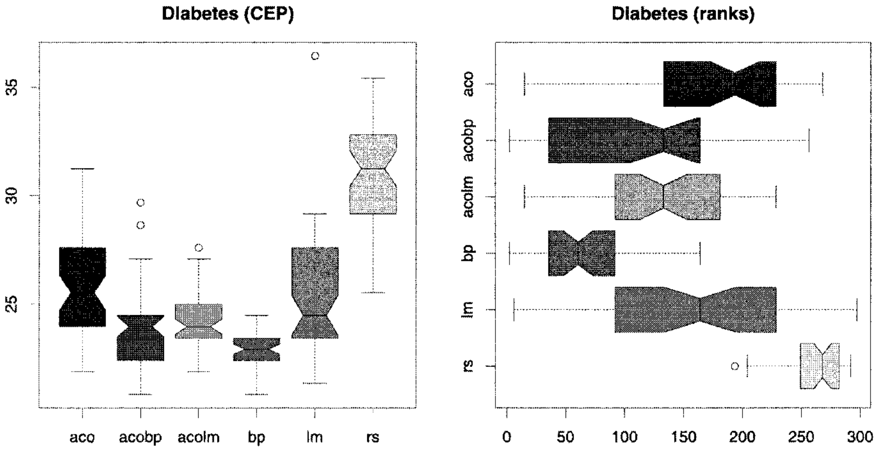

Figure 8-8. Performance comparison of the algorithms on the Diabetes1 problem. The graphic on the left represents the actual CEP values, while the right one represents the ranks among all the solutions generated.

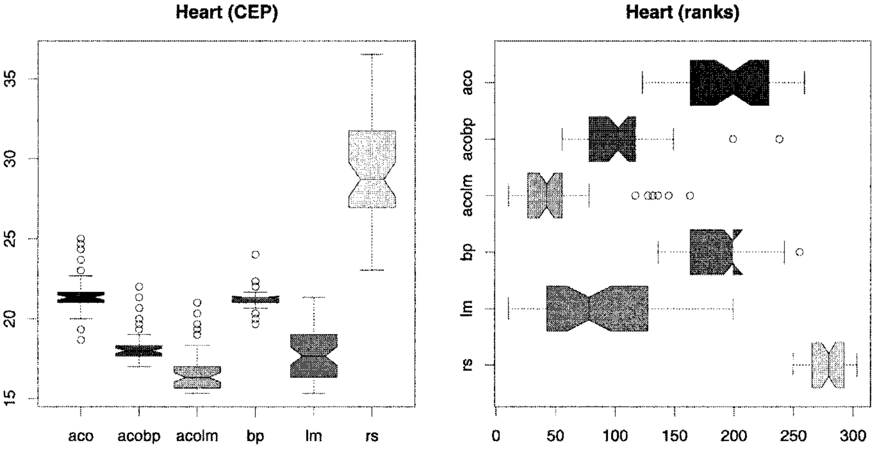

Figure 8-9. Performance comparison of the algorithms on the Heart1 problem. The graphic on the left represents the actual CEP values, while the right one represents the ranks among all the solutions generated.

The Heart problem (Fig. 8-9) with 230 weight values is 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 Heart. $\mathrm{ACO}_{\mathbb{R}}$ 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 $\mathrm{ACO}_{\mathbb{R}}$ — $\mathrm{ACO}_{\mathbb{R}}$-BP and $\mathrm{ACO}_{\mathbb{R}}$ -LM. The $\mathrm{ACO}_{\mathbb{R}}$-BP hybrid clearly outperforms both $\mathrm{ACO}_{\mathbb{R}}$ and BP. $\mathrm{ACO}_{\mathbb{R}}$-LM outperforms respectively $\mathrm{ACO}_{\mathbb{R}}$ and LM. Additionally, $\mathrm{ACO}_{\mathbb{R}}$-LM performs best overall.

Summarizing, we note that the performance of $\mathrm{ACO}_{\mathbb{R}}$ alone does often not quite reach the performance of the derivative based algorithms and the $\mathrm{ACO}_{\mathbb{R}}$ hybrids. Its performance is, however, not much worse. Furthermore, the results show that hybridizing $\mathrm{ACO}_{\mathbb{R}}$ with BP or LM helps to improve the results of the pure $\mathrm{ACO}_{\mathbb{R}}$ algorithm. This was especially the case for Heart, where $\mathrm{ACO}_{\mathbb{R}}$-LM was the overall winner. We want to remind at this point that $\mathrm{ACO}_{\mathbb{R}}$ is much more general than for example BP and LM, because it does not require derivative information. Hence, it may be applied when the neuron transfer function of a NN is non-differentiable or unknown, while algorithms such as BP or LM could not be used in this case.

Table 8-4. Pair-wise comparison of the results of $\mathrm{ACO}_{\mathbb{R}}$-based algorithms with recent results obtained by a set of GA based algorithms (Alba and Chicano, 2004). The results can be compared thanks to maintaining the same experimental setup. 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	*ACO_R*	*GA-BP*	*ACO_R-BP*	*GA-LM*	*ACO_R-LM*
Cancer	16.76	**2.39**	**1.43**	2.14	**0.02**	2.08
	(6.15)	**(1.15)**	**(4.87)**	(1.09)	**(0.11)**	(0.68)
Diabetes	36.46	**25.82**	36.46	**23.80**	28.29	**24.26**
	(0.00)	**(2.59)**	(0.00)	**(1.73)**	(1.15)	**(1.40)**
Heart	41.50	**21.59**	54.30	**18.29**	22.66	**16.53**
	(14.68)	**(1.14)**	(20.03)	**(1.00)**	(0.82)	**(1.37)**

Finally it is interesting to compare the performance of the $\mathrm{ACO}_{\mathbb{R}}$ based algorithms to some other general optimization algorithms. Alba and Chicano (2004) 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.

Table 8-4 summarizes the results obtained by the $\mathrm{ACO}_{\mathbb{R}}$ and GA based algorithms. Clearly the stand-alone $\mathrm{ACO}_{\mathbb{R}}$ performs better than the stand-alone GA for all the test problems. $\mathrm{ACO}_{\mathbb{R}}$-BP and $\mathrm{ACO}_{\mathbb{R}}$-LM perform respectively better than GA-BP and GA-LM on both of the more difficult problems—Diabetes and Heart—and worse on Cancer. For the Heart 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. CONCLUSIONS

We have presented an ant colony optimization algorithm (i.e., $ACO_{\mathbb{R}}$) for the training of feed-forward neural networks in classification problems. $ACO_{\mathbb{R}}$ is a generic approach that can be flexibly used either as a stand-alone method, or hybridized with more problem specific algorithms. The performance of the algorithm was evaluated on real-world test problems and compared to specialized algorithms for feed-forward neural network training (back propagation and Levenberg-Marquardt), and also to genetic algorithm based algorithms.

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 between $ACO_{\mathbb{R}}$ and the Levenberg-Marquardt algorithm (i.e., $ACO_{\mathbb{R}}$-LM) was in some cases able to outperform the back propagation and the Levenberg-Marquardt algorithms that are traditionally used for neural network training. Finally, when compared to other general-purpose algorithms, namely genetic algorithm based algorithms from the literature, our results showed that the ant colony optimization based algorithms may provide superior performance for some of the test problems.

ACKNOWLEDGEMENTS

This work was supported by the Spanish CICYT project no. TIC-2002-04498-C05-03 (TRACER), 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.

REFERENCES

Alba, E., and Chicano, J.F, 2004, Training Neural Networks with GA Hybrid Algorithms, in: *Proceedings of Genetic and Evolutionary Computation - GECCO 2004, Part 1*, Lecture Notes in Computer Science, vol. 3102, K. Deb et al, eds., Springer-Verlag, Berlin, Germany, pp. 852-863.

Battiti, R., and Tecchiolli, G., 1996, The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization, *Annals of Operations Research* **63**:153-188.

Bilchev, G., and Parmee, I. C., 1995, The ant colony metaphor for searching continuous design spaces, in: *Proceedings of the AISB Workshop on Evolutionary Computation*, Lecture Notes in Computer Science, vol. 993, T.~C. Fogarty, ed., Springer-Verlag, Berlin, Germany, pp. 25-39.

Birattari, M., 2004, *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*, Ph.D. thesis, ULB, Brussels, Belgium.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K., 2002, A Racing Algorithm for Configuring Metaheuristics, in: *Proceedings of Genetic and Evolutionary Conference*, W. B. Langdon *et al.* eds., Morgan Kaufmann, San Francisco, CA, USA, pp. 11-18.

Blum, C., 2005, Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling, *Computers &* Operations *Research* **32**(6):1565-1591.

Blum, C., and Roli, A., 2003, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Computing Surveys* **35**(3):268-308.

Blum, C., and Sampels, M., 2004, An ant colony optimization algorithm for shop scheduling problems, *Journal of Mathematical Modelling and Algorithms* **3**(3):285-308.

Blum, C., 2005, Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling, *Computers &* Operations *Research* **32**(6):1565-1591.

Bonabeau, E., Dorigo, M., and Theraulaz, G., 1999, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, NY.

Box, G. E. P., and Muller, M. E, 1958, A note on the generation of random normal deviates. *Annals of Mathematical Statistics* **29**(2):610-611.

Černý, V., 1985, A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm, *Optimization Theory and Applications* **45**:41-51.

Chelouah, R., and Siarry, P., 2000, A continuous genetic algorithm designed for the global optimization of mulitmodal functions, *Journal of Heuristics* **6**:191-213.

Chelouah, R., and Siarry, P., 2000, Tabu search applied to global optimization, *European Journal of Operational Research* **123**:256-270.

Chelouah, R., and Siarry, P., 2003, Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions, *European Journal of Operational Research* **148**:335-348.

Costa, D., and Hertz, A., 1997, Ants can color graphs, *Journal of the Operational Research Society* **48**:295-305.

den Besten, M. L., Stützle, T., and Dorigo, M., 2000, Ant colony optimization for the total weighted tardiness problem, in: *Proceedings of PPSN-VI, Sixth International Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, vol. 1917, M.~Schoenauer et al., eds., Springer Verlag, Berlin, Germany, pp. 611-620.

Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J.-M., 1990, The self-organizing exploratory pattern of the argentine ant, *Journal of Insect Behaviour* **3**:159-168.

Dorigo, M., 1992, *Optimization, Learning and Natural Algorithms* (in Italian), PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M., and Gambardella, L. M., 1997, Ant Colony System: A cooperative learning approach to the travelling salesman problem, *IEEE Transactions on Evolutionary Computation* **1**(1):53-66.

Dorigo, M., Maniezzo, V., and Colorni, A., 1991, Positive feedback as a search strategy, Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M., Maniezzo, V., and Colorni, A., 1996, Ant System: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics — Part B* **26**(1):29-41.

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

Dréo, J., and Siarry, P., 2002, A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions, in: *Proceedings of ANTS 2002—From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms*, Lecture Notes in Computer Science, vol. 2463 of LNCS, M. Dorigo et al., eds., Springer Verlag, Berlin, Germany, pp. 216-221.

Fogel, L. J., Owens, A. J., and Walsh, M. J., 1966, *Artificial Intelligence through Simulated Evolution*, Wiley.

Gagné, C., Price, W. L., and Gravel, M., 2002, Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times, *Journal of the Operational Research Society* **53**:895-906.

Gambardella, L. M., and Dorigo, M., 2000, Ant Colony System hybridized with a new local search for the sequential ordering problem, *INFORMS Journal on Computing* **12**(3):237-255.

Gambardella, L. M., Taillard, É. D., and Agazzi, G., 1999, MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows, in: *New Ideas in Optimization*, D. Corne et al., eds., McGraw Hill, London, UK, pp. 63-76.

Glover, F., 1989, Tabu search—Part I, *ORSA Journal on Computing* **1**(3):190-206.

Glover, F., 1990, Tabu search—Part II, *ORSA Journal on Computing* **2**(1):4-32.

Glover, F., and Kochenberger, G., 2002, *Handbook of Metaheuristics*, Kluwer Academic Publishers, Norwell, MA.

Glover, F., and Laguna, M., 1997, *Tabu Search*, Kluwer Academic Publishers.

Goldberg, D. E., 1989, *Genetic algorithms in search, optimization, and machine learning*, Addison Wesley, Reading, MA.

Golub, G. H., and van Loan, C. F., 1989, *Matrix Computations*, 2nd ed., the John Hopkins University Press, Baltimore, MD, USA.

Guntsch, M., and Middendorf, M., 2002, A population based approach for ACO, in: *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTim*, vol. 2279, S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, eds., Springer-Verlag, Berlin, Germany, pp. 71-80.

Hagan, M. T., and Menhaj, M. B., 1994, Training Feedforward Networks with the Marquardt Algorithm, *IEEE Transactions on Neural Networks* **5**:989-993.

Hastie, T., Tibshirani, R., and Friedman, J., 2001, *The Elements of Statistical Learning*, Springer-Verlag, Berlin, Germany.

Holland, J. H., 1975, *Adaption in natural and artificial systems*, The University of Michigan Press, Ann Harbor, MI.

Hoos, H. H., and Stützle, T., 2004, *Stochastic Local Search: Foundations and Applications*, Elsevier, Amsterdam, The Netherlands.

Kern, S., Müller, S. D., Hansen, N., Büche, D., Očenášek, J., and Koumoutsakos, P., 2004, Learning probability distributions in continuous evolutionary algorithms—A comparative review, *Natural Computing* **3**(1):77-112.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., 1983, Optimization by simulated annealing, *Science* **220**(4598):671-680.

Maniezzo, V., 1999, Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem, *INFORMS Journal on Computing* **11**(4):358-369.

Maniezzo, V., and Colorni, A., 1999, The Ant System applied to the quadratic assignment problem, *IEEE Transactions on Data and Knowledge Engineering* **11**(5):769-778.

Mathur, M., Karale, S. B., Priye, S., Jyaraman, and V. K., Kulkarni, B. D., 2000, Ant colony approach to continuous function optimization, *Industrial & Engineering Chemistry Research* **39**:3814-3822.

McGill, R., Tukey, J. W.,Larsen, and W. A., 1978, Variations of box plots, *The American Statisticia* **32**:12-16 .

Merkle, D., Middendorf, M., and Schmeck, H., 2002, Ant Colony Optimization for Resource-Constrained Project Scheduling, *IEEE Transactions on Evolutionary Computation* **6**(4):333-346.

Monmarché, N., Venturini, and G.,Slimane M., 2000, On how Pachycondyla apicalis ants suggest a new search algorithm, *Future Generation Computer Systems* **16**:937-946.

Nelder, J. A., and Mead, R., 1965, A simplex method for function minimization, *Computer Journal* **7**:308-313.

Papadimitriou, C. H., and Steiglitz, K., 1982, *Combinatorial Optimization—Algorithms and Complexity*, Dover Publications, Inc., New York.

Papliński, A.P., 2004, Lecture 7—Advanced Learning Algorithms for Multilayer Perceptrons, available online at http://www.csse.monash.edu.au/courseware/cse5301/04/L07.pdf.

Prechelt, L., 1994, Proben1—A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical Report 21, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany.

Rechenberg, I., 1973, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog.

Reimann, M., Doerner, K., and Hartl, R. F., 2004, D-ants: Savings based ants divide and conquer the vehicle routing problems, *Computers & Operations Research* **31**(4):563-591.

Rumelhart, D., Hinton, G., and Williams, R., 1986, Learning Representations by Backpropagation Errors, *Nature* **323**:533-536.

Siarry, P., Berthiau, G., Durbin, F., and Haussy, J., 1997, Enhanced simulated annealing for globally minimizing functions of many-continuous variables, *ACM Transactions on Mathematical Software* **23**(2):209.228.

Socha, K., 2003, The Influence of Run-Time Limits on Choosing Ant System Parameters, in *Proceedings of GECCO 2003—Genetic and Evolutionary Computation Conference*, Lecture Notes in Computer Science, vol. 2723, E. Cantu-Paz et al., eds., Springer-Verlag, Berlin, Germany, pp. 49-60.

Socha, K., 2004, Extended ACO for continuous and mixed-variable optimization, in: *Proceedings of ANTS 2004—Fourth International Workshop on Ant Algorithms and Swarm Intelligence*, Lecture Notes in Computer Science, M. Dorigo et al., eds., Springer Verlag, Berlin, Germany, pp. 35-46.

Socha, K., Sampels, M., and Manfrin, M., 2003, Ant algorithms for the university course timetabling problem with regard to the state-of-the-art, in: *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2003*, vol. 2611, G. Raidl et al., eds., pp 334-345.

Storn, R., and Price, K., 1997, Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* **11**:341-359.

Stützle, T., 1998, An Ant Approach to the Flow Shop Problem, in: *Proceedings of the Fifth European Congress on Intelligent Techniques and Soft Computing, EUFIT'98*, pp 1560-1564.

Stützle, T., and Hoos, H. H., 2000, *MAX-MIN* Ant System, *Future Generation Computer Systems* **16**(8):889-914.