Exploring the Impact of Different Classification Quality Functions in an ACO Algorithm for Learning Neural Network Structures

Khalid M. Salama[1] and Ashraf M. Abdelbar[2]
[1]*School of Computing, University of Kent, Canterbury, U.K.*
[2]*Mathematics and Computer Science Department, Brandon University, Brandon, Canada*

Keywords: Ant Colony Optimization (ACO), Machine Learning, Pattern Classification, Neural Networks.

Abstract: Although artificial neural networks can be a very effective classification method, one of the drawbacks of their use is the need to manually prescribe the neural network topology. Recent work has introduced the ANN-Miner algorithm, an Ant Colony Optimization (ACO) technique for optimizing the topology of arbitrary FFNN's, i.e. FFNN's with multiple hidden layers, layer-skipping connections, and without the requirement of full-connectivity between successive layers. In this paper, we explore the use of several classification quality evaluation functions in ANN-Miner. Our experimental results, using 30 popular benchmark datasets, identify several quality functions that significantly improve on the simple Accuracy quality function that was previously used in ANN-Miner.

1 INTRODUCTION

Feed-Forward Neural Networks (FFNN) are a popular and effective pattern classification technique. However, one drawback of FFNN's is the need to manually prescribe the neural network topology. Even if one's attention is restricted to three-layer FFNN's with full-connectivity between successive layers and no layer-skipping connections, then one only needs to select the number of neurons in the single hidden layer. If one allows for arbitrary feed-forward topologies, then optimizing the network topology becomes more challenging.

Recent work has introduced the ANN-Miner algorithm (Salama and Abdelbar, 2014), an Ant Colony Optimization (ACO) technique for optimizing the topology of arbitrary FFNN's, i.e. FFNN's with multiple hidden layers, layer-skipping connections, and without the requirement of full-connectivity between successive layers. In this paper, we explore the use of several classification quality evaluation functions in ANN-Miner. Our experimental results, using 30 popular benchmark datasets, identify several quality functions that significantly improve on the simple Accuracy quality function that was used in the original work on ANN-Miner (Salama and Abdelbar, 2014).

2 FEED-FORWARD NEURAL NETWORKS

Feed-forward neural networks (FFNN) are widely acknowledged as being one of the most popular methods for pattern classification. The most common FFNN topology is a three-layer topology in which neurons are arranged in an input layer, a hidden layer, and an output layer. Commonly, there are connections between every neuron in a layer to all the neurons in the succeeding layer.

Most commonly, each neuron i is a simple computational unit which accepts r inputs $o_1, \ldots, o_r$, and produces a single output o_i:

$$net_i = \sum_{i=1}^{r} w_{ij} o_j + \theta_i \,, \tag{1}$$

$$o_i = \sigma(net_i) = a \cdot \tanh(b \cdot net_i) \,, \tag{2}$$

where each input o_j is the output of a neuron in the previous layer, the weight w_{ij} represents a real-valued weight between neuron j and neuron i, θ_i represents a weight associated with neuron i itself called the neuron's "self-bias," and σ is an *activation function* that is most commonly selected to be the sigmoidally-shaped logistic function shown in Eq. (2).

A FFNN with n input neurons and m output neurons computes a mapping $R^n \mapsto R^m$. In a classification problem, the training set $\mathcal{T}$ consists of a number

Salama K. and Abdelbar A..
Exploring the Impact of Different Classification Quality Functions in an ACO Algorithm for Learning Neural Network Structures.
DOI: 10.5220/0005031301370144
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2014), pages 137-144
ISBN: 978-989-758-052-9

of labelled patterns. Each training pattern x is an n-dimensional input vector of real values, and the label is an m-dimensional output vector y. The k-th element in y that represents the class of pattern x is set to 1, while the other $(m-1)$ elements in y are set to 0. The aim is to train a FFNN, given the training set $\mathcal{T}$, to be able to correctly classify (predict the label of) a new unlabelled pattern. For that, each training pattern x is, in turn, applied to the input layer of the network, the signal is allowed to propagate through the network, and the output of the network, denoted y', is compared to the desired output y to determine the error of the network for that pattern, denoted E_x. A common error function is the simple Sum of Squared Error (SSE) function, defined as:

$$E_x = \frac{1}{2}\sum_{i=1}^{m}(y-y')^2 \,, \tag{3}$$

where the total error is simply: $E = \sum_x E_x$.

Perhaps the most popular neural network training algorithm is the gradient descent based Backward Error Propagation (BP) algorithm which is based on repeatedly applying the training set to the network (each full pass through the training set is called an *epoch*), computing the error E, and then modifying each element of the weight vector according to: $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$, where η is the learning rate parameter. Commonly, FFNN applications use a simple three-layer network topology, with full connectivity between layers.

3 ANT COLONY OPTIMIZATION

Swarm intelligence is a branch of soft computing in which a wide variety of biological collective behaviours are applied to solve optimization problems. Ant Colony Optimization (ACO) was defined as a meta-heuristic for combinatorial optimization problems (Dorigo and Stützle, 2004), inspired by the behaviour of natural ant colonies. The basic principle of ACO is that a population of artificial ants cooperate to find the best path in a graph, analogously to the way that natural ants cooperate to find the shortest path between two points like their nest and a food source.

In ACO, each artificial ant constructs a candidate solution to the target problem, represented by a combination of solution components in the search space. Ants cooperate via indirect communication, by depositing pheromone on the selected solution components for a candidate solution. The amount of pheromone deposited is proportional to the quality of that solution, which influences the probability with which other ants will use that solution's components when constructing their solution. This contributes to the global search aspect of ACO algorithms. The population of ants searches for the best solution in parallel, thus exploring possibly different regions of the search space at each iteration of the algorithm. This increases the chances of finding a near-optimal solution in the search space.

ACO has been successful in tackling the classification problem of data mining. A number of ACO-based algorithms have been introduced in the literature with different classification learning approaches. Ant-Miner (Parpinelli et al., 2002) is the first ant-based classification algorithm, which discovers a list of classification rules in the form of `IF-Conditions-Then-Class`. The algorithm has been followed by several extensions in (Parpinelli et al., 2002; Salama et al., 2011; Salama et al., 2013; Otero et al., 2009; Otero et al., 2013).

ACDT (Boryczka and Kozak, 2010; Boryczka and Kozak, 2011) and Ant-Tree-Miner (Otero et al., 2012) are two different ACO-based algorithms for inducing decision trees for classification. Salama and Freitas (2013a; 2013b) have recently employed ACO to learn various types of Bayesian network classifiers.

As for learning neural networks, the ACO meta-heuristic was utilized in two works. Liu et al. (2006) proposed ACO-PB, a hybrid of the ant colony and back-propagation algorithms to optimize the network weights. It adopts ACO to search the optimal combination of weights in the solution space, and then uses the BP algorithm to further fine-tune the ACO solution. Blum and Socha applied $ACO_{\mathbb{R}}$, an ant colony optimization algorithm for continuous optimization (Socha and Dorigo, 2008; Liao et al., 2014), to train feed-forward neural networks (Socha and Blum, 2007).

4 THE ANN-Miner ALGORITHM

As discussed previously, the three-layer fully-connected FFNN topology is the most commonly used FFNN topology. ANN-Miner (Salama and Abdelbar, 2014), a recently proposed ACO algorithm for learning FFNN topologies, allows connections to be generated between hidden neurons and other hidden neurons — under the restriction that the topology remains acyclic — as well as direct connections between input neurons and output neurons. This allows producing networks with a variable number of layers, as well as arbitrary connections that skip over layers.

As for the problem at hand, a candidate solution is a network topology, and the solution components are the possible connections (between input and hidden neurons, between hidden and output neurons, be-

tween input and output neurons, and connections between different hidden neurons). Each potential connection $c = i \rightarrow j$, connecting between neurons i and j, has two solution components in the construction graph: D_c^{true}, representing the decision to include connection $i \rightarrow j$ in the current candidate topology being constructed by the ant, and D_c^{false}, representing the decision not to include the connection. Therefore, the construction graph can be represented as a two-dimensional $2 \times |C|$ array, where 2 refers to the Boolean solution components, and C is the set of the available connections.

The number of input neurons and output neurons depends on the dataset and the representation that is used for the attributes of the dataset, while the total number of hidden neurons is an external user-supplied parameter. Suppose the total number of neurons is N, with N_i input neurons, N_o output neurons, and N_h potential hidden neurons. $N_i \times N_h$, $N_h \times N_o$ and $N_i \times N_o$ are the number of available connections between input and hidden neurons, hidden and output neurons, and input and output neurons, respectively, in the total available number of connections $|C|$. This means that, for instance, an ant can select (or unselect) a connection between any input neuron and any hidden neuron. The same applies for the two other connection types.

However, the available connections between the hidden neurons N_h are defined as follows. In order to ensure that the topology is acyclic, we impose the restriction that $i \rightarrow j$ is not available if $i \geq j$. In other words, each hidden neuron has a numeric index, and we only allow connections from a given hidden neuron n_i to a higher-numbered neuron n_j. It is well-known that any directed acyclic graph is isomorphic to a graph where the nodes are lexicographically ordered and for all arcs (u,v) in the graph u precedes v in the lexicographic order. Hence, the number of the available connection between the N_h hidden neurons is: $(N_h-1)+(N_h-2)+...+1+0 = N_h(N_h-1)/2$.

The overall process of ANN-Miner is illustrated in Algorithm 1. In the initialization step of ANN-Miner (line 3), the amount of pheromone assigned to each solution component D_c^a – where a can be true or false – in the construction graph is initialized with the value 0.5. Hence, for each connection c, the probability of including $i \rightarrow j$ (i.e. selecting D_c^{true}) in the topology equals the probability of not including $i \rightarrow j$ (i.e. selecting D_c^{false}).

In the inner **for-loop** (lines 6-12), each ant_i in the colony creates a candidate solution NN_i, i.e. a complete neural network (line 7). Then the quality of the constructed solution is evaluated (line 8). The best solution NN_{tbest} produced in the colony is selected to update the pheromone trail according to the quality

Algorithm 1: Pseudo-code of ANN-Miner.

```
1:  Begin
2:  NN_bsf = ϕ; t = 1;
3:  InitializePheromone();
4:  repeat
5:      NN_tbest = ϕ; Q_tbest = 0;
6:      for i = 1 → colony_size do
7:          NN_i = ant_i.CreateSolution();
8:          Q(NN_i) = EvalQuality(NN_i, T_v);
9:          if Q(NN_i) > Q(NN_tbest) then
10:             NN_tbest = NN_i;
11:         end if
12:     end for
13:     UpdatePheromone();
14:     if Q(NN_tbest) > Q(NN_bsf) then
15:         NN_bsf = NN_tbest;
16:     end if
17:     t = t + 1;
18: until t = max_iterations or Convergence();
19: NN_final = PostProcessing(NN_bsf);
20: return NN_final;
21: End
```

of its solution $Q(NN_{tbest})$. After that, the algorithm compares the iteration-best solution NN_{tbest} with the best-so-far solution NN_{bsf} (the **if** statement in lines 14-16) to keep track of the best solution found so far during the algorithm execution.

After the outer **repeat-until** loop terminates (lines 4-18), the best-so-far neural network undergoes a post-processing step to produce the final neural network NN_{final} to be returned by the algorithm. Basically, the algorithm learns the final weights of the connections in the neural network NN_{bsf} — which represents the best topology found during the ACO search process. The standard BP procedure, described in Section 2, is used to train NN_{bsf} and learn its final weights. The only difference is that, instead of having BP work on the conventional three-layer fully-connected network topology, it works on the arbitrary topologies constructed by ANN-Miner.

The process of an ant creating a new candidate solution (neural network) is described in Algorithm 2. The procedure starts with an empty (edge-less) neural network (line 2) to be constructed throughout the procedure. In addition, an empty array SLN, which represents the ant trail in the construction graph and its selected solution components, is initialized. This data structure is necessary for the pheromone update procedure, as described later. For each connection c in the available set of connections C, the ant selects D_c^a to decide whether to include this connection in the candidate network NN or not (line 4) — by either se-

lecting solution component D_c^{true} or D_c^{false}. The selection of the solution component at each step is based on the following probabilistic state transition formula

$$p(D_c^a) = \frac{\tau(D_c^a)}{\tau(D_c^{true}) + \tau\left(D_c^{false}\right)}, \quad (4)$$

where $p(D_c^a)$ is the probability of selecting decision D^a for connection c, and $\tau(D_c^a)$ is the current amount of pheromone associated with D_c^a. Every selected solution component D_c^a (where $a = true$ or $a = false$) is added to the data structure *SLN* (line 5). However, only if $D_c^a = D_c^{true}$, that is, the ant selected the decision to include connection c in the topology, the corresponding connection $(i \rightarrow j)_c$ is appended to the candidate network *NN* (the **if** statement in lines 6-8). After the ant visits all the available connections in the construction graph and performs the include-or-not decision, the network topology of *NN* is now complete.

Algorithm 2: Pseudo-code of solution creation.

```
1: Begin CreateSolution()
2: NN ← ϕ ; SLN ← ϕ;
3: for c = 1 → |C| do
4:     D_c^a = SelectDecisionComponent();
5:     SLN = SLN ∪ D_c;
6:     if D_c^a == D_c^true then
7:         NN = NN ∪ (i → j)_c;
8:     end if
9: end for
10: InitNetwork(NN, NN_bsf);
11: TrainNeuralNetwork(NN, T_l);
12: return NN;
13: End
```

The weights of the neural network *NN* are then initialized (line 10) as follows. For each connection $i \rightarrow j$ in the topology of the *NN* network, if the connection $i \rightarrow j$ is also present in the topology of NN_{bsf} — the best trained neural network encountered so far — then the weight w_{ji} associated with the connection $i \rightarrow j$ is initialized with the value that it has in the trained network NN_{bsf}. On the other hand, if the connection $i \rightarrow j$ is not present in N_{bsf}, then the weight w_{ji} is randomly initialized.

The idea here is to start the training of each newly-created topology by building on the weights of the best network we have generated so far NN_{bsf}. If the new network, after training, performs better than NN_{bsf}, it will replace NN_{bsf} and its connection weights will be used as initial values for performing BP on subsequent networks. Of course, some connections in *NN* may not exist in NN_{bsf}, in which case, they are randomly initialized, as mentioned above. Further, some connections in NN_{bsf} may not exist in *NN*. Such a diversion in the topologies maintains the exploration aspect of the algorithm, while the exploitation aspect is realized by building on the best weights learned in previous iterations.

After *NN* is initialized, it is trained (line 11) using the BP procedure, with a relatively large learning rate and a small number of epochs (see Table 1 for parameter settings). These are intended as "quick and dirty" parameter settings meant to allow us to obtain a complete neural network and evaluate its pattern classification quality. The classification quality evaluation is discussed in the next section.

5 EXPLORING DIFFERENT QUALITY FUNCTIONS

A key objective of a classification algorithm is to learn models with good generalization, that is, models that are able to accurately predict the class labels of *new* unknown patterns. Overfitting occurs when the induced model reflects good classification performance (fit) on the training (in-sample) data used in the learning process, yet shows bad predictive performance (generalization) involving new/testing data.

Therefore, we split the training set $\mathcal{T}$ at the beginning of the algorithm into two mutually exclusive parts: 1) the learning set $\mathcal{T}_l$, which contains 80% of the training set and is used to learn the neural network topology and weights (line 11, Algorithm 2); and 2) the validation set $\mathcal{T}_v$, which contains 20% of the training set and is used to evaluate the quality of the model (line 8, Algorithm 1).

In this paper, we investigate several quality evaluation functions to be used in line 8 of Algorithm 1. First, let us establish some notation. Let m denote the number of classes, $\hat{c}$ denote the true (correct) class for a given pattern x (the element in the label vector y where the value=1), and c denote the class that is predicted by the neural network *NN*. We will assume that the output of the network is an m-dimensional vector $y' = o_1, o_2, \ldots, o_m$ of real values. The output vector of *NN* can be transformed into a vector p of class probability scores through a simple normalization:

$$p_k = o_k / \sum_{j=1}^{m} o_j. \quad (5)$$

Hence, the predicted class c will be the label k of the output of the *NN* with the highest probability score p_k. We will use the notation $f(NN|x)$ to refer to the value of the classification measure f on *NN* given pattern x, and $Q_f(NN|\mathcal{T}_v)$ as the quality of *NN*, using

classification measure f, on the full validation set $\mathcal{T}_v$. Note that $Q_f(NN|\mathcal{T}_v)$ is the actual pheromone amount to be deposited in the pheromone update step (line 13, Algorithm 1); the higher the value of $Q_f(NN|\mathcal{T}_v)$, the better the quality of NN. Recall that the amount of pheromone deposited is proportional to $Q(NN_{tbest})$, as follows:

$$\tau(D_c^a) = \tau(D_c^a) + [\tau(D_c^a) \times Q(NN_{tbest})] \;\; \forall D_c^a \in |SLN_{tbest}|. \quad (6)$$

Note that, for quality function f, if Q_f is computed to be less than zero, it is adjusted to zero, in order not to allow negative pheromone values.

Example 1 — suppose we have a pattern x where $m = 3$ and $\hat{c} = 1$. Consider three candidate NNs with the following probability score vectors given pattern x: $NN_1(x) = (0.9, 0.1, 0)$, $NN_2(x) = (0.6, 0.4, 0)$ and $NN_3(x) = (0.6, 0.2, 0.2)$. This example will be used in the following subsections.

5.1 Accuracy

A simple, and perhaps the most widely used, classification measure is accuracy. This is the quality measure that was used in previous work on ANN-Miner (Salama and Abdelbar, 2014). For a given pattern x,

$$Acc(NN|x) = \begin{cases} 1 & \text{if} \quad \hat{c} = c, \\ 0 & \text{if} \quad \hat{c} \neq c. \end{cases} \quad (7)$$

For an entire validation set $\mathcal{T}_v$,

$$Q_{Acc}(NN|\mathcal{T}_v) = \frac{1}{|\mathcal{T}_v|} \sum_{x \in \mathcal{T}_v} Acc(NN|x). \quad (8)$$

The deficiency of the accuracy measure is that, in the three NNs of Example 1, $Acc(x)$ will be equal to 1 for all three probability vectors. However, it is obvious that NN_1 should receive a better quality preference than NN_2 and NN_3, since it produces a higher probability for the true class.

5.2 Class Probability Error

An alternative to the accuracy measure is the true Class Probability Error (CPE) measure. For a given pattern x,

$$CPE(NN|x) = 1 - p_{\hat{c}}. \quad (9)$$

Let us again consider Example 1. Here, $CPE(NN_1|x) = 0.1$, while $CPE(NN_2|x) = CPE(NN_3|x) = 0.4$. In this case, CPE prefers NN_1 over NN_2 and NN_3, but does not differentiate between NN_2 and NN_3.

For an entire validation set $\mathcal{T}_v$,

$$Q_{CPE}(NN|\mathcal{T}_v) = 1 - \frac{1}{|\mathcal{T}_v|} \sum_{x \in \mathcal{T}_v} CPE(NN|x). \quad (10)$$

5.3 Variations of the CPE Measure

We consider two variations of CPE that differ primarily in their handling of outlier patterns. First, CPE^k is defined as follows. For a single pattern x,

$$CPE^k(NN|x) = (1 - p_{\hat{c}})^k. \quad (11)$$

In this paper, we focus on the case where $k = 2$. Suppose we have a dataset $\mathcal{T}_1$ consisting of three patterns with individual pattern error values of: 0.1, 0.2, 0.3, and another dataset $\mathcal{T}_2$ with pattern error values of: 0.1, 0.15, 0.35. Under CPE, these two datasets would have equal error values. But, under CPE^2, the second dataset would have a higher error value (0.155 versus 0.14). Thus, CPE^2 is less tolerant of outliers than CPE; a small number of outlier patterns can dominate the CPE^2 function much more easily than the CPE function.

Another variation of CPE is CPE_m which is even more tolerant of outliers than CPE. For a single pattern x, $CPE_m(NN|x) = CPE(NN|x)$. However, for an entire validation set:

$$Q_{CPE_m}(NN|\mathcal{T}_v) = 1 - \text{median}\{CPE(x) : x \in \mathcal{T}_v\}. \quad (12)$$

For the two dataset scenarios described above, $\mathcal{T}_1$ (with pattern errors 0.1, 0.2, 0.3) would have a smaller error value under Q_{CPE_m} than $\mathcal{T}_2$ (with pattern errors 0.1, 0.15, 0.35), because Q_{CPE_m} would simply compare 0.2 to 0.15.

5.4 Quadratic Loss Function

The Quadratic Loss Function is another widely used error measure. For a given pattern x:

$$QLF(NN|x) = (1 - p_{\hat{c}})^2 + \sum_{k:k \neq \hat{c}}^{m} (p_k)^2. \quad (13)$$

In Example 1, the three probability vectors would have QLF values of: 0.02, 0.32, and 0.24. Thus, the QLF error measure would prefer NN_1, followed by NN_3, followed by NN_2. Thus, not only does QLF favour the models that produce a higher probability for the true class, but it also favours the models that produce the lowest probabilities for the other classes.

Note that, for a two-class problem (i.e. $m = 2$),

$$QLF(NN|x) \equiv 2 \cdot CPE^2(NN|x) \; . \quad (14)$$

For an entire validation set $\mathcal{T}_v$,

$$Q_{QLF}(NN|\mathcal{T}_v) = 1 - \frac{1}{|\mathcal{T}_v|} \sum_{x \in \mathcal{T}_v} QLF(NN|x). \quad (15)$$

5.5 Cross Entropy

For a given pattern x, the cross entropy CE measure is defined as:

$$CE(NN|x) = -\sum_{k=1}^{m} y_i \ln p_i \,, \tag{16}$$

where y is the target vector. Since $y_{\hat{c}} = 1$ and $y_k = 0$ for all $k \neq \hat{c}$, Eq. (16) reduces to:

$$CE(NN|x) = -\ln p_{\hat{c}} \,. \tag{17}$$

Thus, CE is somewhat similar to CPE: both are monotonically decreasing functions of $p_{\hat{c}}$. They differ only in their respective response curves as $p_{\hat{c}}$ varies from 0 to 1.

For an entire validation set $\mathcal{T}_v$,

$$Q_{CE}(NN|\mathcal{T}_v) = 1 - \frac{1}{|\mathcal{T}_v|} \sum_{x \in \mathcal{T}_v} CE(NN|x). \tag{18}$$

5.6 Bayesian Information Reward

For a given pattern x, we use a variation of Bayesian Information Reward BIR, defined as:

$$BIR(NN|x) = \frac{1}{m} \sum_{c=1}^{m} IR_c(NN|x) \tag{19}$$

$$IR_c(NN|x) = \begin{cases} 1 - \frac{\log(p_c)}{\log(p'_c)} & \text{if} \quad c = \hat{c} \ (\textbf{reward}), \\ \frac{\log(1-p_c)}{\log(1-p'_c)} & \text{if} \quad c \neq \hat{c} \ (\textbf{penalty}). \end{cases} \tag{20}$$

where p'_c represents the prior probability of class c, which is the ratio of the number of patterns in the learning set with class label c to the total number of patterns in the learning set, and m is the total number of the classes. Note that the first branch in the conditional Equation (24) is the reward value, where the predicted class c is the same as the true (correct) class $\hat{c}$, while the second branch is the penalty value, where the predicted c is not the same as the true class $\hat{c}$.

Not only does the Bayesian Information Reward measure respect the probability of the true class $p_{\hat{c}}$, it also respects the prior probability p'_c of the predicted class in the dataset. This makes BIR more robust to class imbalance situations, where one (or more) class values have high occurrence in a given dataset compared to the other values. That is, the lower the frequency of the correctly predicted class in the dataset, the higher the reward. Similarly, the higher the frequency of the misclassified class in the dataset, the higher the penalty.

For an entire validation set $\mathcal{T}_v$,

$$Q_{BIR}(NN|\mathcal{T}_v) = \frac{1}{|\mathcal{T}_v|} \sum_{x \in \mathcal{T}_v} [\phi_1 + BIR(NN|x)/\phi_2] \,. \tag{21}$$

Table 1: Parameter settings used in experiments.

No.	Parameter	Value
1	`max_iterations`	1000
2	`colony_size`	10
3	`conv_ iterations`	10
4	`Ant BP Learning Rate`	0.05
5	`Ant BP Epochs`	10
6	`3L-BP Learning Rate`	0.01
7	`3L-BP Epochs`	1000

The parameters ϕ_1 and ϕ_2 are used to adjust the actual amount of pheromone to be deposited. The first parameter makes sure that the Q_{BIR} value is greater than 0, while the second parameter scales the Q_{BIR} value. In our experiments, we set $\phi_1 = \phi_2 = 50/m$.

6 COMPUTATIONAL RESULTS

The experiments were carried out using the well-known *stratified* 10-times 10-fold cross-validation procedure, which works as follows. First, the target dataset is divided into 10 mutually exclusive partitions (or folds), with approximately the same number of patterns in each partition. Then, for each of the 7 different quality measures used for candidate-solution evaluation and pheromone updating in this work, a version of ANN-Miner using that measure is run 10 times, where each time a different partition is used as the test set and the other 9 partitions are merged and used as the training set $\mathcal{T}$ for the algorithm. Note that $\mathcal{T}$ is itself further divided during algorithm execution into $\mathcal{T}_l$ and $\mathcal{T}_v$ as discussed in Section 5.

The predictive performance associated with each quality measure is computed as the average value of the accuracy on the test set across the 10 runs. In addition, we ran ANN-Miner with each of the 7 quality measures 10 times – using a different random seed to initialize the search each time – for each cross-validation fold. Thus, the accuracy associated with each quality measure is actually averaged over 100 values (10 cross-validation folds times 10 runs per fold). We also report the results for the standard three-layer fully-connected Neural Network with Back-Propagation (3L-BP) as a baseline.

The performance of classification quality measures was evaluated using 30 public-domain datasets from the well-known University of California at Irvine (UCI) dataset repository. The parameter settings of ANN-Miner used are shown in Table 1. The number of the hidden neurons is set equal to the num-

Table 2: Predictive Accuracy Results.

Dataset	3L-BP	ANN-*Acc*	ANN-CPE^2	ANN-*CPE*	ANN-CPE_m	ANN-*QLF*	ANN-*CE*	ANN-*BIR*
balance	96.50	91.33	96.83	98.67	94.67	97.33	96.67	**99.10**
breast-l	**72.29**	68.80	69.09	68.29	70.22	69.84	71.64	70.84
breast-p	68.26	75.21	76.79	75.76	76.32	77.29	77.29	**77.61**
breast-t	32.64	57.55	**67.03**	60.73	58.36	63.46	57.73	64.38
breast-w	93.86	95.43	94.74	96.14	95.79	96.32	94.91	**96.49**
car	90.29	98.19	98.66	**99.01**	97.78	98.31	97.84	96.83
credit-a	84.35	82.75	85.76	83.04	84.49	**85.80**	85.19	85.76
credit-g	74.00	71.90	74.33	71.00	74.40	**74.67**	74.33	74.40
ecoli	79.53	84.86	83.67	86.08	86.33	**86.94**	85.46	85.18
glass	46.30	48.46	58.99	57.56	**60.39**	52.38	54.68	54.70
hay	60.01	75.45	74.26	77.67	76.43	77.60	71.26	**77.87**
heart-c	57.46	55.43	**58.43**	55.47	56.46	58.41	58.08	57.19
hepatitis	**83.79**	81.92	79.92	81.88	80.00	82.58	83.25	81.34
horse	76.04	78.94	81.67	79.77	**82.01**	79.14	81.42	77.32
ionosphere	89.67	93.38	92.53	93.40	91.10	91.68	91.08	**93.54**
iris	87.28	90.67	93.95	92.57	94.62	**95.28**	93.95	92.54
liver	57.40	64.08	**67.49**	64.07	64.38	64.86	64.98	67.31
monks	54.25	42.20	77.36	**77.70**	70.80	69.63	67.33	71.36
parkinsons	75.78	75.13	85.62	85.68	83.03	**85.71**	82.03	82.53
pima	55.71	42.86	76.20	75.91	**76.56**	76.17	75.51	74.60
pop	81.11	**81.85**	52.68	50.54	68.21	68.39	65.71	56.15
s-heart	**94.16**	92.74	83.71	81.85	83.70	83.33	84.07	81.90
segmentation	70.56	93.16	93.84	93.21	93.16	92.66	93.68	**94.35**
thyroid	86.10	89.78	90.37	88.85	89.80	89.78	87.99	**92.30**
transfusion	70.56	72.60	74.76	73.66	72.75	74.45	72.58	**75.67**
ttt	76.63	98.00	98.31	98.00	98.21	**98.84**	95.26	98.36
vehicle	64.90	60.90	**75.00**	72.69	71.63	71.98	72.11	71.27
voting	93.89	**94.90**	94.12	94.86	92.91	94.59	94.52	94.75
wine	94.41	94.38	95.52	93.86	94.38	**96.04**	94.97	94.44
zoo	81.25	**97.50**	96.50	96.07	96.07	94.64	96.07	96.75
Average Rank	**6.5**	**5.7**	**3.5**	**4.6**	**4.4**	**3.3**	**4.6**	**3.5**

ber of input attributes plus the number of class values (output units) for each dataset. Note that the learning rate and the number of epochs used in BP in each ant solution creation step (line 11, Algorithm 2) are indicated in parameters 4 and 5, respectively, while the learning rate and the number of epochs used in the BP post-processing step (line 19, Algorithm 1) are indicated in parameters 6 and 7, respectively.

Table 2 shows the average predictive accuracy results of the ANN-Miner algorithm with the 7 different classification quality functions, as well as the conventional BP algorithm, where the best result obtained for each algorithm is shown in boldface. The results reported are the average of the 100 runs of the stratified 10-times 10-fold cross-validation procedure. The last row of the table shows the average ranking of each algorithm. The average rank for a given algorithm g is obtained by first computing the rank of g on each dataset individually, with a rank of 1 representing the best performance and a rank of 8 representing the worst performance. The individual ranks are then

Table 3: Results of Friedman statistical significance test with Holm *post hoc* test.

Comparison	p	Holm
3L-BP vs. ANN-*QLF*	5.5E-07	0.00179
3L-BP vs. ANN-*BIR*	1.6E-06	0.00185
3L-BP vs. ANN-CPE^2	1.8E-06	0.00192
ANN-*Acc* vs. ANN-*QLF*	2.3E-04	0.00200
ANN-*Acc* vs. ANN-*BIR*	5.0E-04	0.00208
ANN-*Acc* vs. ANN-CPE^2	5.6E-04	0.00217
3L-BP vs. ANN-CPE_m	9.9E-04	0.00227
3L-BP vs. ANN-*CPE*	0.0020	0.00238
3L-BP vs. ANN-*CE*	0.0024	0.00250

averaged across all datasets to obtain the overall average rank.

As shown in Table 2, the Quadratic Loss Function (ANN-*QLF*) obtained the best overall ranking of 3.3, followed closely by Bayesian Information Reward (ANN-*BIR*) and Squared Class Probability Error (ANN-CPE^2), both of which had an overall ranking of 3.5.

The non-parametric Friedman statistical test with the Holm's post-hoc test was applied, at the conventional 0.05 significance level, to the predictive accuracy results reported in Table 2—an all-pairs multiple comparison on the 8 algorithms under evaluation. The results of the statistical significance test are reported in Table 3; for space limitations, results are reported only for the cases where there is a statistically significant difference. The results indicate that all seven measures are significantly better than the baseline 3L-BP, and four of the measures are also significantly better than ANN-*Acc* (the measure that was used in the original work on ANN-Miner): ANN-*BIR*, ANN-*QLF*, ANN-CPE^2, and ANN-CPE_m.

ACKNOWLEDGEMENT

This partial support of a Brandon University Research Grant is gratefully acknowledged.

REFERENCES

Boryczka, U. and Kozak, J. (2010). Ant colony decision trees. In *4th International Conference on Computational Collective Intelligence*, pages 4373–382.

Boryczka, U. and Kozak, J. (2011). An adaptive discretization in the ACDT algorithm for continuous attributes. In *3rd International Conference on Computational Collective Intelligence*, pages 475–484.

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

Liao, T., Socha, K., de Oca, M. M., Stuetzle, T., and Dorigo, M. (2014). Ant colony optimization for mixed-variable optimization problems. *To appear in IEEE Transactions on Evolutionary Computation*.

Otero, F., Freitas, A., and Johnson, C. (2009). Handling continuous attributes in ant colony classification algorithms. In *IEEE Symposium on Computational Intelligence in Data Mining (CIDM-09)*, pages 225–231.

Otero, F., Freitas, A., and Johnson, C. (2013). A new sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Transactions on Evolutionary Computation*, 17(1):64–74.

Otero, F. E. B., Freitas, A. A., and Johnson, C. G. (2012). Inducing decision trees with an ant colony optimization algorithm. *Applied Soft Computing*, 12(11):3615–3626.

Parpinelli, R. S., Lopes, H. S., and Freitas, A. A. (2002). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332.

Salama, K., Abdelbar, A., and Freitas, A. (2011). Multiple pheromone types and other extensions to the Ant-Miner classification rule discovery algorithm. *Swarm Intelligence*, 5(3-4):149–182.

Salama, K., Abdelbar, A., Otero, F., and Freitas, A. (2013). Utilizing multiple pheromones in an ant-based algorithm for continuous-attribute classification rule discovery. *Applied Soft Computing*, 13(1):667–675.

Salama, K. M. and Abdelbar, A. M. (2014). A novel ant colony algorithm for building neural network topologies. In *Proceedings ANTS-14*. Accepted.

Socha, K. and Blum, C. (2007). An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training. *Neural Computing & Applications*, 16:235–247.

Socha, K. and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185:1155–1173.