An Empirical Comparison of Combinations of Evolutionary Algorithms and Neural Networks for Classification Problems

Erick Cantú-Paz and Chandrika Kamath

Abstract

There are numerous combinations of neural networks (NNs) and evolutionary algorithms (EAs) used in classification problems. EAs have been used to train the networks, design their architecture, and select feature subsets. However, most of these combinations have been tested on only a few data sets and many comparisons are done inappropriately measuring the performance on training data or without using proper statistical tests to support the conclusions. This paper presents a comprehensive empirical evaluation of eight combinations of EAs and NNs on 15 public-domain and artificial data sets. Our objective is to identify the methods that consistently produce accurate classifiers that generalize well. In most cases, the combinations of EAs and NNs perform equally well on the data sets we tried and were not more accurate than hand-designed neural networks trained with simple backpropagation.

Keywords

classification, evolutionary algorithms, machine learning, network design, training algorithms, feature selection

I. Introduction

There are numerous combinations of neural networks (NNs) and evolutionary algorithms (EAs) used in classification problems. EAs have been used to train the networks, design their architecture, and select feature subsets [1], [2], [3]. Most of these combinations have been tested on few data sets and only a handful of studies have compared different combinations of EAs and NNs on the same domain [4], [5], [6]. Moreover, many comparisons are done inappropriately using the performance on training data or without using proper statistical tests to support the conclusions.

This paper presents a comprehensive empirical evaluation of eight combinations of EAs and NNs on 16 public-domain and artificial data sets. Our objective is to identify the methods that consistently produce accurate classifiers that generalize well. To perform the comparisons, we use a recently-developed statistical test that is well accepted in machine learning for comparing classifiers.

We cannot test the myriad combinations of EAs and NNs that have been proposed, but we chose combinations that are representative of the main types. First, we experimented with EAs to train neural networks. We used different EAs

E. Cantú-Paz and C. Kamath are with the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 7000 East Avenue, L-561, Livermore, CA. Email: cantupaz@acm.org and kamath2@llnl.gov

with binary and real encodings. Since it is well known that EAs can benefit from combining them with local optimizers, we hybridized the EAs with backpropagation and experimented varying the number of epochs. We also explored the possibility of altering the weights encoded in the chromosomes of EA individuals using the results of backpropagation (Lamarckian evolution) as well as exploiting the Baldwin effect, where the improved results are used in the fitness calculations, but the chromosomes remain unaltered.

We also experimented with EAs to select feature subsets and with four methods to design the architecture of the NNs with EAs. To design the networks we used direct encodings that specify each connection among neurons to build and prune the networks, as well as indirect encodings that develop the structure of the networks from information contained in the chromosomes.

In a previous study, we compared six of the algorithms used in the present paper on real-life astronomical data [7]. The algorithms we tried performed equally well on that data with one notable exception: Using genetic algorithms (GAs) to select feature subsets yielded the most accurate classifiers. Those results motivated us to explore EA and NN combinations in more detail to verify if our conclusions applied to other data sets. Somewhat surprisingly, and in contrast to other studies, the experiments in this paper show that the combinations of EAs and NNs performed equally well in most cases. Moreover, the EA and NN combinations were not more accurate than networks trained with simple backpropagation.

The following section reviews several combinations of EAs and NNs that have appeared previously in the literature. Section III describes the data sets, algorithms, and the statistical test used to compare the experimental results, which are presented in section IV. Section V concludes the paper with our observations and plans for future work.

II. Background

Evolutionary algorithms and neural networks have been combined in two major ways. First, EAs have been used to train or to aid in the training of NNs. In particular, EAs have been used to search for the weights of the network, or to reduce the size of the training set by selecting the most relevant features. The second major type of collaboration is to use EAs to design the structure of the network. The structure largely determines the efficiency of the network and the classes of problems that it can solve. It is well known that to solve non-linearly separable problems, the network must have at least one hidden layer between the inputs and outputs; but determining the number and the size of the hidden layers is mostly a matter of trial and error. EAs have been used to search for these parameters, as well as for the pattern of connections and for developmental instructions to generate a network. The interested reader may consult more extensive reviews [1], [8], [2], [3].

This section first reviews some basic concepts from evolutionary algorithms. Then, we describe how to use EAs to train NNs, to select features, and to determine the topology of the network.

A. Evolutionary Algorithms

Evolutionary algorithms are randomized search procedures that mimic the process of natural evolution to solve optimization problems. There are different types of evolutionary algorithms, such as genetic algorithms [9], evolution strategies, evolutionary programming, and genetic programming. Despite some differences, EAs share many properties. EAs operate on a population of individuals that represent possible solutions to a problem. The representation of a solution is defined by the user and may be as simple as a string of zeroes and ones or as complex as a computer program. The initial population may be created entirely at random or using some domain knowledge (in the form of previously known solutions, for example). The algorithm evaluates the individuals to determine how well they solve the problem at hand with an objective function, which is unique to each problem and must be supplied by the user. The individuals with better performance are selected into a mating pool to serve as parents of the next generation of individuals. EAs create new individuals using simple randomized operators that are inspired by sexual recombination (crossover) and mutation in natural organisms. The new solutions are evaluated, and the cycle of selection and creation of new individuals is repeated until a satisfactory solution is found or a predetermined time limit elapses.

There are numerous methods to select promising solutions into the mating pool. This paper uses binary (pairwise) tournaments, which is one of the simplest selection methods. This selection method randomly selects two individuals without replacement from the current population, and the most fit individual (according to the objective function) is incorporated into the mating pool. The random pairing of individuals is repeated twice to obtain a mating pool of the same size as the original population.

Most of the experiments of the present study use binary-encoded GAs. In GAs, crossover is the primary method to create new solutions and we use two crossover operators in the present study: multi-point and uniform crossovers. Multi-point crossover randomly chooses a pair of previously selected individuals from the mating pool and a number of crossover points along their chromosomes. Then, this operator exchanges segments of the two chromosomes delimited by the crossover points. Uniform crossover randomly selects the donor parent for each bit in the offspring.

In GAs, mutation occurs with a low frequency, but mutation is the primary search operator in evolution strategies and evolutionary programming. In the binary case, mutation consists of flipping one randomly-chosen bit from zero to one or vice versa. Some theoretical studies support the use of a mutation rate of $1/l$, where l is the length of the chromosome [10], [11]. Although our application may not satisfy the assumptions made in those studies, this choice of mutation rate has been successful in several practical situations, and we adopt it for our experiments.

B. Training Networks with Evolutionary Algorithms

Training a neural net is an optimization task with the goal of finding a set of weights that minimizes some error measure. The search space is high dimensional and, depending on the error measure and the input data, it may contain numerous local optima. Some traditional network training algorithms, such as backpropagation (BP), use some form of

gradient search, and may get trapped in local optima. In contrast, EAs do not use any gradient information, and are likely to avoid getting trapped in a local optimum by sampling simultaneously multiple regions of the space.

A straightforward combination of genetic algorithms and neural networks is to use the EA to search for weights that make the network perform as desired. The architecture of the network is fixed by the user prior to the experiment. In this approach, each individual in the GA represents a vector with all the weights of the network.

In the simplest variation of this method, the weights found by the EA are used in the network without any further refinement [12], [13], [14], [15]. This is particularly useful when the activation function of the neurons is non-differentiable and traditional gradient-based training algorithms cannot be used.

An alternative is to use backpropagation or other methods to refine the weights represented in each individual [16], [17]. The motivation of this approach is that GAs quickly identify promising regions of the search space, but they do not fine-tune parameters very fast. So, EAs are used to find a promising set of initial weights from which a gradient-based method can quickly reach an optimum. This approach extends the processing time per individual, but sometimes the overall training time can be reduced because fewer individuals may need to be considered before reaching an acceptable solution.

EA can also be used to refine weights found by a traditional NN learning algorithm [18]. In general, seeding the initial population is an effective way to bias the EA toward good solutions.

These approaches are straightforward and have produced good results, but suffer from several problems. First, since adjacent layers in a network are usually fully connected, the total number of weights is $O(n^2)$, where n is the number of units. Longer individuals usually require larger populations, which in turn result in higher computational costs. For small networks, the GA can be used to search for good weights efficiently, but this method may not scale up to larger domains.

Another drawback is the so-called permutations problem [19]. The problem is that by permuting the hidden nodes of a network, the representation of the weights in the chromosome would change, although the network is functionally the same. Some permutations may not be suitable for GAs because crossover might easily disrupt favorable combinations of weights. To ameliorate this problem, Thierens [20] suggested placing incoming and outgoing weights of a hidden node next to each other. An analysis by Hancock [21] suggested that the permutation problem is not as difficult as it is often presented, and Thierens [22] presented an encoding that avoids the permutations problem altogether.

The most common approach is to use the EA to find the initial weights and then refine the weights with backpropagation. There are two variations of these hybrid methods, which are inspired by the Lamarckian model of evolution and the Baldwin effect. In the Lamarckian algorithm, the weights of the trained network are encoded back in the chromosomes. While in natural evolution it is impossible to change a genome to reflect acquired or learned characteristics, the Lamarckian mechanism has demonstrated its usefulness in some EA applications. In a Baldwinian algorithm, the fitness

is calculated using a BP-trained network, but the weights modified by BP are not encoded back into the chromosomes. The Baldwin effect has also been shown useful in EA applications.

Of course, the success of Lamarckian or Baldwinian approaches depends on the problem. In evolving neural networks the results have been mixed: Ku and Mak [23] show improved performance using a Lamarckian strategy while Gruau and Whitley [24] obtained equally effective results with a Lamarckian and Baldwinian algorithms. Similarly, in general function optimization, the results are mixed: Whitley, Gordon and Mathias [25] observed that using the Baldwin effect was slower but more effective than a Lamarckian approach; Houck, Jones, Kay, and Wilson [26] showed that pure Lamarckian and Baldwinian approaches were outperformed by updating the chromosomes of only a fraction of the population (i.e., using a "partial" Lamarckianism); and Julstrom [27] showed examples where using the Baldwin effect was actually worse than straightforward Darwinian search and the best results were obtained with Lamarckian strategies. In the present paper, we investigate both Lamarckian and Baldwinian approaches.

C. Feature Selection with Evolutionary Algorithms

Besides searching for weights, EAs may be used to select the features that are input to the NNs. The training examples may contain features that are irrelevant or redundant, but it is generally unknown a priori which features are relevant. Avoiding irrelevant and redundant features is desirable not only because they increase the size of the network and the training time, but also because they may reduce the accuracy of the network.

Selecting a subset of features with EAs is straightforward, using binary-encoded GAs and the so-called wrapper approach [28]: The chromosome of the individuals contains one bit for each feature, and the value of the bit determines whether the feature will be used in the classification. The individuals are evaluated by training the networks (that have a predetermined structure) with the feature subset indicated by the chromosome. The resulting accuracy is used to calculate the fitness. Since the pioneering work of Siedlecki and Sklanski [29], genetic algorithms have been used for many feature selection problems using neural networks [30], [31], [32], [33] and other classifiers such as decision trees [34], k-nearest neighbors [35], [36], rules [37], and Naive Bayes [38], [39].

D. Using EAs to Design the Topology

As mentioned before, the topology of a network is crucial to its performance. If a network has too few nodes and connections, it may not be able to learn the required concept. On the other hand, if a network has too many nodes and connections, it may overfit the training data and have poor generalization. Miller, Todd, and Hedge [40] identified two major approaches to use EAs to design the topology of NNs: use a direct encoding to specify every connection of the network or evolve an indirect specification of the connectivity.

D.1 Direct Encodings

The key idea behind direct encodings is that a neural network may be regarded as a directed graph where each node represents a neuron and each edge is a connection. A common method of representing directed graphs is with a binary connectivity matrix: the i, j-th element of the matrix is one if there is an edge between nodes i and j, and zero otherwise. Binary-encoded GAs appear well suited for direct encoding, because the connectivity matrix can be represented in a GA simply by concatenating its rows or columns [40], [41]. The algorithm uses the connectivity matrix to build a network which is then trained, and the performance of the network is used to calculate the fitness. Using this method, Whitley, Starkweather, and Bogart [42] showed that the GA can find topologies that learn faster than the typical fully-connected feedforward network. The GA can be explicitly biased to favor smaller or sparsely connected networks, which can be trained faster. However, since each connection is explicitly coded, the length of the individuals is $O(n^2)$ (where n is the number of neurons), and the algorithm is not scalable to large problems.

Instead of using direct encodings to build a network, it is possible to use direct encodings to prune an excessively large network to try to improve its generalization. Numerous algorithms have been used to prune neural networks [43]. Pruning begins by training a fully-connected neural network. Most pruning methods delete a single weight at a time in a greedy fashion, which may result in suboptimal networks. Additionally, many pruning methods fail to account for interactions between multiple weights. This may be problematic if, for example, a network has two weights that should be deleted, but deleting only one weight results in a decrease in performance. Greedy methods would not prune the weights, but an algorithm that considers weight interactions and more than one weight at a time may have better chances of reducing the size of the network significantly without affecting the classification accuracy. For these reasons, GAs seem promising for NN pruning.

Genetic algorithms have been used to prune networks with good results [42], [44], [45]. Whitley and Bogart [46] suggest to retrain the network for a few epochs after pruning. We performed experiments to confirm this idea, but we found only limited advantages of retraining. It is also possible to prune entire (input and hidden) nodes, but in the present paper we experiment only with the more common approach of pruning individual weights.

D.2 Indirect Encodings

A simple indirect encoding method is to commit to a particular topology (feedforward, recurrent, etc.) and a particular learning algorithm, and then use an EA to find the parameter values that complete the network specification. For example, with a fully-connected feedforward network, the EA may search for the number of layers and the number of units per layer. Another example would be to code the parameters of a particular learning algorithm, such as the momentum and the learning rate of backpropagation [41], [47]. By specifying only the parameters for a given topology, the coding is very compact and well suited for an evolutionary algorithm. However, this method is constrained by the initial choice of topology and learning algorithm.

A more sophisticated approach to indirect representations is to use a grammar to encode rules that govern the development of a network. Kitano [48] introduced the earliest grammar-based approach. He used a connectivity matrix to represent the network, but instead of encoding the matrix directly in the chromosome, he used a graph-rewriting grammar to generate the matrix. The chromosomes contain rules that rewrite scalar matrix elements into 2×2 matrices.

In this grammar, there are 16 terminal symbols that are 2×2 binary matrices. There are 16 non-terminal symbols, and the rules have the form $n \rightarrow m$, where n is one of the scalar non-terminals, and m is a 2×2 matrix of non-terminals. There is an arbitrarily designated start symbol, and the number of rewriting steps is fixed by the user.

Only the 16 right-hand sides of the rules are contained in the chromosome, the left side is implicit in the position of the rule. To evaluate the fitness of individuals, the rules are decoded and the connectivity matrix is developed by applying all the rules that match non-terminal symbols. Then, the connectivity matrix is interpreted and the network is constructed and trained with backpropagation.

Perhaps the major drawback of this approach is that the number of units must be 2^i (where i is any non-negative integer), because after each rewriting step the size of the matrix doubles in each dimension.

Other examples of grammar-based developmental systems are the work of Boers and Kuiper [49] with Lindenmayer systems, Gruau's "cellular encoding" method [50], and the system of Nolfi, Elman, and Parisi [51] that simulates cell growth, migration, and differentiation.

Siddiqi and Lucas [5] compared Kitano's method against direct encoding and found that, in contrast to Kitano's original results, the direct encoding method performed at least as well as the grammar-based encoding. Grönroos [6] also compared Kitano's method against a direct encoding and Nolfi et al.'s indirect encodings. Grönroos experimented with four artificial and four real-world problems. His experiments favored Kitano's method, but also suggested that the evolutionary algorithms did not find networks that were more accurate than hand-designed networks. However, Grönroos did not had the computational resources to do appropriate statistical testing of his results.

E. Experimental Comparisons

As we mentioned in the introduction, only a handful of studies compare different combinations of EAs and NNs on the same problems. Only a few more compare the performance of the EA against traditional network training algorithms. Most of the publications introduce an algorithm and present results to demonstrate the feasibility of the new method. These publications report valuable innovative research, but at some point the new algorithms must be carefully validated and compared to existing approaches.

Usually, when combinations of EAs and NN are introduced, the algorithms are evaluated informally and there are almost no follow up studies or independent verification of the results. The usual approach is to evaluate the newly proposed method on very few data sets—and frequently only on a single one—severely limiting the generality of the results. Unfortunately, most comparisons are done improperly, either by presenting "convergence graphs" that show

the value of some statistic (usually *not* the classification accuracy) as a function of computational effort (measured as generations, epochs, etc.) or by presenting a performance metric calculated on the training sets. While there might be some value in those results, the accuracy of the final network must be evaluated on an independent test set that has not been considered by the algorithm at any moment during construction or training of the networks.

Since EAs and NNs are stochastic algorithms, multiple experiments should be performed and compared with appropriate statistical tests to verify the validity of the comparisons. This may sound obvious, but it is still common practice to compare results informally using convergence graphs or the results of single experiments.

Schaffer, Whitley, and Eshelman [8] noted that "studies rigorously comparing different approaches are as yet very rare." By examining the proceedings of a recent workshop on combinations of EAs and NNs [52] we observed that comparisons remain very rare. Of nine papers relevant to classification, we found that only three used generally accepted methods to estimate generalization (either crossvalidation experiments or training/testing partitions adequate for the particular application). One of these three papers performed experiments on five data sets, another used three data sets, and the last paper used a single data set of a particular application. Of these three papers, none verified the results with statistical tests to support their conclusions, although one of the papers presented the variances of the experiments allowing the reader to perform tests. The remaining papers relevant to classification were mostly proof-of-concept papers with single runs on single data sets, with no results of accuracy or no experiments at all.

Experimental deficiencies are not exclusive to research on combinations of EA and NNs. Prechelt [53] noted that the situation was not much better in top neural network journals. Analyzing 190 papers, Prechelt found that one third did not present a quantitative comparison with an existing method. His survey also found that 29% of new algorithms were not evaluated on any real problem and that only 8% present results on more than one real-world problem. We believe that the present paper addresses some of these concerns by using an accepted method to compare eight EA+NN combinations against hand-designed neural nets on 12 real-world and four synthetic data sets.

III. Methods

This section details the data sets and algorithmic details used in this study as well as the method used to compare algorithms. We defer details of the fitness functions and the parameters used in each algorithm to the next section.

The data sets used in the experiments are briefly described in Table I. The data sets are available in the UCI repository [54], except for the last four synthetic data sets. Random21 and Redundant21 are synthetic data sets with 21 features each. The target concept of these data sets is to define whether their first nine features are closer to (0,0,...,0) or (9,9,...,9) in Euclidean distance. The features were generated uniformly at random in the range [3,6]. All the features in Random21 are random. The first, fifth, and ninth features are repeated four times each in Redundant21. The accuracy of neural networks may be degraded if trained with irrelevant or redundant features, and we intend to use these synthetic data sets to test the robustness of the algorithms. We took the definition of Redundant21 from the feature selection

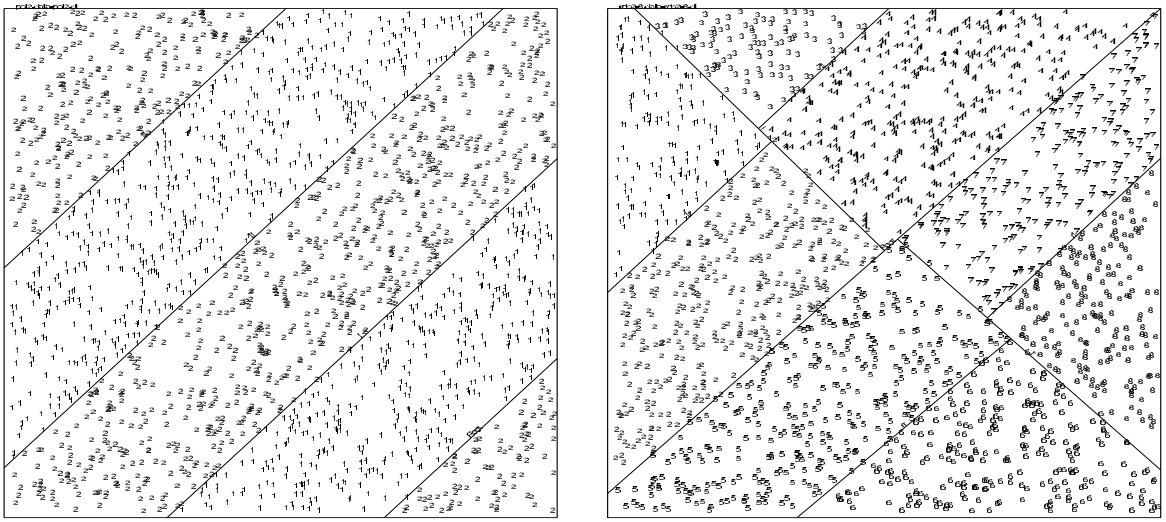

Fig. 1. The POL2 (left) and RCB2 (right) artificial data sets.

			Features			Neural Network			
Domain	**Cases**	**Class**	**Cont.**	**Disc.**	**Miss**	**Input**	**Output**	**Hidden**	**Epochs**
Breast Cancer	699	2	9	–	N	9	1	5	20
Credit-Australian	653	2	6	9	Y	46	1	10	35
Credit-German	1000	2	7	13	N	62	1	10	30
Pima Diabetes	768	2	9	–	Y*	8	1	5	30
Heart-Cleveland	303	2	6	7	N	26	1	5	40
Housing	506	3	12	1	N	13	3	2	70
Ionosphere	351	2	34	–	N	34	1	10	40
Iris	150	3	4	–	N	4	3	5	80
Kr-vs-kp	3196	2	–	36	N	74	1	15	20
Sonar	208	2	60	–	N	60	1	10	60
Wine	178	3	13	–	N	13	3	5	15
POL2	2000	2	2	–	N	2	1	15	30
RCB2	2000	8	2	–	N	2	8	10	25
Random21	2500	2	21	–	N	21	1	1	100
Redundant21	2500	2	21	–	N	21	1	1	100

TABLE I

DESCRIPTION OF THE DATA SETS USED IN THE EXPERIMENTS. FOR EACH DATA SET, THE TABLE SHOWS THE NUMBER OF INSTANCES; THE NUMBER OF CLASSES; THE NUMBER OF CONTINUOUS AND DISCRETE FEATURES; WHETHER THE DATA HAS MISSING VALUES; THE NUMBER OF INPUT, HIDDEN, AND OUTPUT UNITS; AND THE NUMBER OF EPOCHS OF BACKPROPAGATION USED TO TRAIN THE NETWORKS.

study by Inza et al. [38].

POL2 and RCB2-8 are the two-dimensional data sets depicted in figure 1. We used these data sets in previous experiments with oblique decision trees [55]. The concept represented by the POL2 data is a set of four parallel oblique lines (hence its name), it contains 2000 instances divided into two classes. The "rotated checker board" (RCB2) data also has 2000 instances, but in this case they are divided into eight classes.

Instances with missing values in Credit-Australian were deleted. Following the usual practice, the missing values in Pima-Diabetes (denoted with zeroes) were not removed and were treated as if their values were meaningful. The classes in Housing were obtained by discretizing the attribute "mean value of owner-occupied homes" as follows: class = 1 if

log(median value) ≤ 9.84, class = 2 if $9.84 <$ log(median value) ≤ 10.075, and class = 3 otherwise.

For all experiments, each numeric feature in the data was linearly normalized to the interval $[-1, 1]$. The discrete features and the class labels were encoded with the usual 1-in-C coding if there are $C > 2$ values (one of the C outputs is set to 1 and the rest to -1). Binary values were encoded as a single -1 or 1 value.

The programs were written in C++ and compiled with g++ version 2.96 with -O2 optimizations. The experiments were executed on a single processor of a Linux (Red Hat 7.1) workstation with dual 2.4 GHz Intel Xeon processors and 512 Mb of memory. The programs used a Mersenne Twister random number generator.

The experiments used feedforward networks with one hidden layer. All neurons are connected to a "bias" unit with constant output of 1.0. Unless specified otherwise, the inputs are connected to all the hidden units, which in turn are connected to all the outputs. In feedforward operation, the units compute their net activation as net $= \sum_{i=1}^{d} x_i w_i + w_0$, where d is the number of inputs to the neuron, x_i is an input, w_i is the corresponding weight, and w_0 is the weight corresponding to the "bias" unit. Each unit emits an output according to $f(\text{net}) = \tanh(\beta \text{net})$. Unless specified otherwise, we set $\beta = 1.0$ in all experiments. Some of the experiments used simple backpropagation with momentum with a learning rate of 0.15 and a momentum term of 0.9. The network sizes and the number of training epochs varied for each data set and are specified in Table I. These backpropagation and network parameters were taken from a study by Opitz and Maclin [56]. In each epoch, the examples were presented to the network in a different random order.

The usual method to compare classification algorithms is to perform k-fold crossvalidation experiments to estimate the accuracy of the algorithms and use t-tests to confirm if the results are significantly different. In crossvalidation, the data D is divided into k non-overlapping sets, $D_1, ..., D_k$. At each iteration i (from 1 to k), the network is trained with $D \backslash D_i$ and tested on D_i. However, it has been shown that comparing algorithms using t-tests on crossvalidation experiments results in an increased type-I error: The results are incorrectly deemed significantly different more often than expected given the level of confidence used in the test [57].

To amerliorate this problem, we followed the procedure recommended by Dietterich [57] and Alpaydin [58] and used 5 iterations of 2-fold crossvalidation (5x2cv). In each iteration, the data were randomly divided in halves. One half was input to the algorithms, and the other half was used to test the final solution. The accuracy results presented in the next section are the average and standard deviations of the ten tests.

Note that the algorithms further split the half of the data input to them into training/validation sets or use it in internal crossvalidations experiments to guide the search or to decide when to stop. Presenting the results of these internal crossvalidations or the accuracy on the validation set as the final result of the algorithms is common, but incorrect. The final results should always be tested on unseen data. The accuracy results that we present are obtained by testing the final solutions on the half of the data that has not been considered at all by the algorithms.

Having an outer 5x2 crossvalidation loop allows us to partition the data to do proper comparisons on unseen testing

data and also to use the combined F test proposed by Alpaydin [58] that ameliorates the problems of the crossvalidated t-test and has high power. Let $p_i^{(j)}$ denote the difference in the accuracy of two classifiers in fold j of the i-th iteration of 5x2cv, $\bar{p} = (p_i^{(1)} + p_i^{(2)})/2$ denote the mean, and $s_i^2 = (p_i^{(1)} - \bar{p})^2 + (p_i^{(2)} - \bar{p})^2$ the variance, then

$$f = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} \left(p_i^{(j)}\right)^2}{2 \sum_{i=1}^{5} s_i^2}$$

is approximately F distributed with 10 and 5 degrees of freedom. We rejected the null hypothesis that the two algorithms have the same error rate with a 0.05 significance level if $f > 4.74$. All the algorithms used the same training and testing data in the two folds of the five crossvalidation experiments.

IV. Experiments

We present experiments that correspond to the combinations of EAs and NNs discussed in section II: using EAs to train neural networks, select feature subsets, and design network topologies.

A. Training Networks with EAs

We trained neural networks with the following methods:

1. Using binary-encoded GAs and real-encoded EAs to search for weights.
2. Using a binary-encoded GA to search for initial weights and refine the weights using backpropagation.
   (a) Varying the number of backpropagation epochs (1, 2, and 5 epochs).
   (b) Varying the fraction of individuals refined with BP.
   (c) Using Baldwinian and Lamarckian evolution models.

The first objective of these experiments is to test whether there is an advantage of using real vs. binary codings in training neural nets on our selection of data sets. We recognize that the comparison is necessarily incomplete, as we cannot test the myriad algorithmic variations and operators designed to operate on binary and real values. However, we present results with algorithms that represent extremes in the sophistication of EAs in use: a simple GA with a straightforward binary encoding and a state-of-the-art real parameter GA.

The second objective is to test the advantage of hybridizing the EAs using backpropagation to refine the weights. We varied the number of BP epochs to measure how much local search is required to reach good solutions. It has been suggested that refining a small fraction of individuals is enough to benefit from the hybridization with local optimizers. We performed experiments where 5% of the individuals (the usually recommended fraction) are refined by BP and experiments where the entire population is refined. We also tested whether encoding the trained weights back into the individuals (Lamarckian evolution) results in more accurate networks than simply using the trained network to assess the fitness without modifying the chromosomes (Baldwin effect).

To avoid overfitting, we follow the usual procedure of splitting the input data into evaluation and validation sets. The evaluation set with 70% of the items will be used to evaluate the fitness of the individuals. The validation set consists of the remainding 30% of the data and will be used to estimate the generalization ability of the best result found during each generation. The output of the GA will be the solution that exhibited the highest accuracy on the validation set. The results presented in the tables are the averages of the 5x2cv accuracies measured on the test sets that have not been considered during training.

A.1 Binary and Real Encodings

In the binary-encoded GA, each weight was represented with 16 bits and ranged in [-1, 1]. The population size for the GAs was set to $n = \lfloor 3\sqrt{l} \rfloor$, where l is the length of the chromosomes in bits, following the gambler's ruin model for population sizing that asserts that the population size required to reach a solution of a particular quality is $O(\sqrt{l})$ [59]. The initial population was initialized uniformly at random. The results reported are from GAs that used uniform crossover. Extensive testing with single- and multi-point crossovers did not yield significant differences. The mutation rate was set to $1/l$, following theoretical studies [10], [11] that assert that, under some conditions, this is an optimal setting. Although our problems may not satisfy the assumptions of the analyses, this choice of mutation rate has been successful in several practical situations. As in all experiments, pairwise tournament selection without replacement was used to select promising solutions. The entire population is replaced every generation and there is no elitism. The GA was stopped after a limit of 100 generations and the solution with the highest estimated accuracy on the validation set was returned.

For the real encoding, we chose the EA described by Deb, Anand, and Joshi [60]. This EA is based on a parent-centric recombination operator (PCX) and a steady-state elitist replacement method (G3). The parent-centric recombination means that offspring are likely to be close to the parents, in contrast with other recombination operators where the offspring are close to the centroid of the parents. The G3 method always selects the best individual to participate as a parent. Two other parents are selected randomly. After offspring are created, the method replaces two randomly selected parents with the best two solutions from a combined population formed of the two parents and the newly generated offspring. We refer to this EA as G3PCX. Deb et al. [60] compared the performance of their system against five real parameter optimizers: two real-encoded GAs, a correlated self-adaptive evolution strategy, differential evolution, and the quasi-Newton method. G3PCX consistently reached target solutions using fewer function evaluations than the other methods, and G3PCX was shown to scale well to increasing problem size. For our experiments, we modified the code provided by the authors to use a Mersenne Twister random number generator, to allocate memory dynamically, and to use the termination criteria described below.

For G3PCX, we set the population size to $n = \lfloor 30\sqrt{l} \rfloor$, where l is the number of weights in the network. The algorithm was stopped after n iterations with no change in the best solution found, or after a limit of $50n$ function evaluations.

Domain	Neural Net		G3PCX		Binary	
Breast Cancer	96.39	0.58	**98.94**	2.35	**98.88**	0.32
Credit-Australian	82.53	9.49	82.05	2.86	83.28	1.69
Credit-German	70.12	1.39	**30.00**	1.46	70.94	1.49
Diabetes-Pima	73.30	3.47	75.49	1.3	73.83	2.45
Heart-Cleveland	78.17	3.16	**90.42**	2.12	87.72	3.42
Housing	64.62	12.76	61.22	4.70	67.94	7.31
Ionosphere	84.77	3.80	**64.10**	2.04	74.10	1.94
Iris	94.53	2.12	89.73	11.7	88.67	6.09
Kr-vs-kp	74.30	6.47	80.12	4.04	**90.14**	0.60
Sonar	69.61	3.12	67.40	5.44	**73.65**	2.55
Wine	95.16	1.76	**84.94**	11.46	92.47	4.55
POL2	90.72	2.53	**87.74**	1.65	90.81	1.81
RCB2-8	92.61	0.88	**53.96**	6.88	96.41	1.41
Random21	91.70	3.76	97.52	0.68	93.29	4.24
Redundant21	91.75	3.95	98.88	0.61	98.52	0.50
Mean Accuracy	83.35		77.50		85.38	
Median Time	4.9		8.6		41	
Mean Time	10.1		11.8		277	

TABLE II

MEANS AND STANDARD DEVIATIONS OF THE CLASSIFICATION ACCURACIES OBTAINED BY A HAND-DESIGNED NEURAL NETWORK, THE G3PCX REAL-ENCODED ALGORITHM, AND A BINARY-ENCODED SIMPLE GA. RESULTS THAT ARE SIGNIFICANTLY DIFFERENT ($\alpha = 0.05$) FROM THE HAND-DESIGNED NEURAL NETWORK (FIRST COLUMN) ARE HIGHLIGHTED IN **bold**. TIMES ARE IN CPU SECONDS.

Since the G3PCX algorithm uses steady-state replacement, the iterations in the G3PCX algorithm do not correspond to generations in the simple GA, and we calibrated the termination criteria experimentally. The PCX parameters σ_ζ and σ_η were both set to 0.2. All other parameters were set to their default values.

In both the binary- and real-encoded GAs, the fitness of each individual was calculated by placing the weights encoded by the individual into a neural net and calculating the accuracy on the evaluation set (70% of the training data). The architecture of the neural net for each problem is fixed before the experiments as described in Table I. The network with the best accuracy in the validation data (30% of the training data) is returned as the result of an experiment. As explained previously, the results reported are the averages of 5x2cv accuracies measured with previously unseen test sets.

The results of the binary GA and the real-encoded G3PCX are presented in Table II along with the results of a hand-designed neural network trained with backpropagation using the number of epochs specified in Table I. **Bold** typeface is used to highlight the results that are significantly different from the hand-designed neural network (first column) according to the combined F test at a 0.05 significance level. The table shows that only very few results of the EAs are significantly different than backpropagation: The GA found significantly better results in three cases, while the G3PCX was divided with two better and five worse results than the neural net. In the remainder of the experiments, we use only the binary-encoded GA.

A.2 Refining Weights with Backpropagation

The second training method described in section II-B is to refine the weights encoded in the individuals using backpropagation. This method has numerous options. The user must decide how many epochs of backpropagation will be used, how many individuals will undergo refinement, and what to do with the refined weights. There is no guidance on how to choose these parameters, so we experimented using one, two and five training epochs; we applied BP to 5% and 100% of the population; and we tested both Lamarckian and Baldwinian variations. To remain consistent, we used the same network architecture and GA parameters as in the previous section.

At the end of each experiment in the Baldwinian model, the best set of weights found by the GA was used to initialize a final network that was trained using the entire training data and tested on the previously unseen testing data. In the case of Lamarckian evolution, the weights refined with BP were encoded back in the chromosomes of the individuals, and therefore the final network was not trained further.

Table III has the results of Baldwinian and Lamarckian experiments on 5% of the population as well as the binary GA results from Table II to facilitate comparisons. As before, bold denote results that are significantly different from the hand-designed neural networks and daggers (†) denote results that are significantly different than the simple GA without BP.

Recall that the binary GA performed significantly better than the hand-designed NN on the Breast Cancer, Kr-vs-kp and Sonar data sets. Both Baldwinian and Lamarckian methods perform significantly better than the hand-designed neural net in the same three data sets, and there are only a few other significant differences. In particular, with Credit-Australian, the Baldwinian method found significantly more accurate networks when one epoch was used, but the differences were not significantly different with more BP epochs or with Lamarckian evolution. In fact, the results indicate that additional epochs of BP tend to *decrease* the accuracy of the networks on the testing data and that Lamarckian evolution performs slightly worse than the Baldwinian method.

The decrease in accuracy with additional BP epochs is coupled with a slight—but consistent—reduction in the number of generations until termination. This suggests that BP might have effectively refined the weights of a few (5%) individuals that quickly dominated the population before allowing sufficient time to explore the search space. One possibility to avoid this dominance of a few is to refine all the individuals.[1]

The results of refining all the individuals in the population are presented in Table IV. In these results, the trend of diminishing accuracy with increased refinement epochs is much less noticeable. With POL2 refining all the population caused the accuracy to fall $\approx 10\%$, and in RCB2-8 the accuracy improved noticeably. However, in general the accuracies

[1]There are, of course, many alternatives to deal with the so-called premature convergence problem. The problem and some alternatives are described by Goldberg [61]. Effective solutions seek a balance between selection and exploration either by altering the selection method or the genetic operators used to explore new solutions. To facilitate comparisons with the rest of the experiments of this section, we decided to change the fraction of individuals refined with BP, which is one of the parameters we are studying, and leave the GA parameters and operators unchanged.

		Baldwinian Evolution			Lamarckian Evolution		
Domain	**0 BP**	**1 BP**	**2 BP**	**5 BP**	**1 BP**	**2 BP**	**5 BP**
Breast Cancer	**98.88** 0.32	**98.48** 0.54	**98.91** 0.33	**99.03** 0.43	**98.68** 0.50	**98.74** 0.48	**99.08** 0.48
Credit-Australian	83.28 1.69	**84.12** 3.14	83.75 1.55	83.32 1.91	81.86 3.13	81.15 1.56	81.11 3.56
Credit-German	70.94 1.49	71.15 1.95	71.98 2.12	70.12 1.43	71.92 1.24	71.67 1.33	69.85 1.93
Diabetes-Pima	73.83 2.45	73.98 3.38	74.92 2.04	74.64 1.95	74.38 1.45	73.31 0.90	73.65 2.03
Heart-Cleveland	87.72 3.42	88.68 1.11	89.25 1.19	89.21 1.99	86.45 2.48	88.82 1.83	87.98 1.75
Housing	67.94 7.31	68.77 2.55	70.91 2.71	69.54 2.56	† 71.50 2.23	69.21 3.27	67.94 3.98
Ionosphere	74.10 1.94	† 68.43 2.87	† 69.43 3.45	† 67.86 4.55	† 65.12 3.06	† 65.88 4.04	† 64.65 4.95
Iris	88.67 6.09	89.47 6.49	† 91.07 4.38	87.20 9.05	89.20 12.74	88.00 13.17	88.13 10.67
Kr-vs-kp	**90.14** 0.60	**91.42** 0.88	**91.71** 1.20	**89.66** 1.06	† **80.88** 2.65	† **79.80** 3.52	† **78.29** 1.89
Sonar	**73.65** 2.55	**74.44** 3.78	**74.12** 2.15	**72.83** 3.32	**72.40** 2.72	**73.31** 2.96	**72.96** 4.32
Wine	92.47 4.55	91.91 3.55	90.90 5.45	89.55 4.11	91.69 3.19	92.25 3.27	91.01 3.52
POL2	90.81 1.81	**85.28** 9.42	**86.40** 4.13	† 79.33 6.96	**85.80** 8.94	**83.43** 6.77	**81.87** 8.31
RCB2-8	96.41 1.41	† **52.56** 12.93	† **52.47** 17.42	† **64.99** 11.28	† **62.49** 9.89	† **63.01** 12.32	† **63.11** 14.23
Random21	93.29 4.24	93.47 1.48	92.61 1.87	93.88 3.73	94.64 1.71	93.63 1.98	94.04 1.51
Redundant	98.52 0.60	98.22 1.15	97.72 2.18	98.66 0.75	98.68 0.75	97.96 0.93	98.13 1.16
Mean Accuracy	85.38	82.03	82.41	81.99	81.71	81.34	80.79
Median Time	41	77	89	136	101	114	145
Mean Time	277	4132	5963	11329	4138	5944	11328

TABLE III

MEANS AND STANDARD DEVIATIONS OF ACCURACIES OBTAINED BY INITIALIZING THE NEURAL NETS WITH THE WEIGHTS ENCODED IN THE CHROMOSOMES AND APPLYING BACKPROPAGATION FOR 1, 2, AND 5 EPOCHS. RESULTS USING BALDWINIAN AND LAMARCKIAN VARIATIONS ON *5%* OF THE MEMBERS OF THE POPULATION ARE PRESENTED. RESULTS THAT ARE SIGNIFICANTLY DIFFERENT ($\alpha = 0.05$) FROM THE HAND-DESIGNED NEURAL NETWORK ARE IN **bold** AND DAGGERS (†) INDICATE RESULTS SIGNIFICANTLY DIFFERENT THAN THE GA WITHOUT BP (FIRST COLUMN). TIMES ARE GIVEN IN CPU SECONDS.

are not much different than with refining only 5% of the individuals, and since refining the entire population is much more costly, this option should probably be avoided.

B. Feature Selection

The next combination of GAs and NNs that we considered is the use of GAs to select a subset of features that will be used to train the networks, as described in section II-C.

We use a binary-encoded GA, because binary codings are natural for this problem. The chromosomes in the GA had one bit for each feature. The population was initialized uniformly at random with $\lfloor 3\sqrt{l} \rfloor$ individuals, but a minimum population size of 20 individuals was enforced. The GA used uniform crossover with probability 1.0 and mutation with rate $1/l$. The networks were trained for the number of epochs indicated in Table I. The algorithm was stopped after the best solution found did not change in five generations, or until a limit of 50 generations was reached. Across the data sets we used, the largest number of generations until termination was 15 (averaged over the 5x2cv trials), indicating that the GAs found good solutions early in their runs.

We are always interested in networks that classify accurately data that were not used in training. In the case of feature selection, we evaluate the fitness of candidate feature subsets using an estimate of the generalization given by a single five-fold crossvalidation.

Since the fitness is an estimate of the generalization of the networks, it may seem appropriate to report the best fitness found by the GA as the result of the algorithm. However, as it has been recently demonstrated [62], [63] this approach has a risk of overfitting the training data. Instead, we use the best feature subset found by the GA to train a final

Domain	0 BP	Baldwinian Evolution 1 BP	Baldwinian Evolution 2 BP	Baldwinian Evolution 5 BP	Lamarckian Evolution 1 BP	Lamarckian Evolution 2 BP	Lamarckian Evolution 5 BP
Breast Cancer	**98.88** 0.32	**98.83** 0.45	**98.86** 0.50	**98.60** 0.62	**98.88** 0.49	**98.94** 0.57	**98.86** 0.38
Credit-Australian	83.28 1.69	82.02 2.95	81.75 1.60	82.60 1.33	79.88 2.94	80.03 1.60	79.11 2.75
Credit-German	70.94 1.49	68.23 2.15	68.11 1.83	67.88 2.03	68.92 2.34	67.66 1.84	66.58 2.10
Diabetes-Pima	73.83 2.45	73.83 2.45	74.53 1.65	72.92 2.10	73.83 1.57	73.07 3.53	72.63 1.56
Heart-Cleveland	87.72 3.42	88.58 1.27	88.45 1.39	88.32 2.29	86.87 2.06	88.98 1.61	87.66 1.57
Housing	67.94 7.31	68.14 3.23	66.84 3.28	65.26 5.71	65.89 2.97	65.18 4.54	63.32 5.48
Ionosphere	74.10 1.94	73.88 2.15	73.15 2.73	72.89 2.98	74.25 1.62	74.03 2.15	73.77 3.67
Iris	88.67 6.09	91.33 6.20	89.87 4.05	91.07 5.40	92.40 4.85	93.20 3.84	92.00 3.72
Kr-vs-kp	**90.14** 0.60	**89.48** 0.60	**89.71** 1.01	**86.46** 0.66	72.30 4.05	69.48 2.52	73.12 2.56
Sonar	**73.65** 2.55	**72.50** 5.27	**72.12** 3.96	**71.73** 4.39	**72.40** 3.57	**71.63** 3.80	**70.48** 5.21
Wine	92.47 4.55	93.60 3.10	92.02 3.71	92.36 3.29	89.55 4.35	87.98 2.56	87.08 5.73
POL2	90.81 1.81	**81.88** 7.37	**75.77** 5.46	† **68.69** 7.52	† **72.21** 16.9	† **76.01** 14.2	† **76.69** 9.94
RCB2-8	96.41 1.41	89.21 13.23	88.64 8.88	89.54 9.99	97.02 5.8	† 87.67 2.26	90.95 1.24
Random21	93.29 4.24	92.98 3.54	89.74 2.85	86.63 4.76	87.05 3.14	85.23 3.23	86.69 2.32
Redundant	98.52 0.60	89.74 5.28	† 82.91 5.28	† 84.34 9.11	90.15 4.54	† 84.15 7.31	85.11 4.54
Mean Accuracy	85.38	83.62	82.16	81.29	81.44	80.22	80.27
Median Time	41	1099	1325	1869	1172	1519	2144
Mean Time	277	67082	90104	131053	65664	90890	142144

TABLE IV

MEANS AND STANDARD DEVIATIONS OF ACCURACIES OBTAINED BY INITIALIZING THE NEURAL NETS WITH THE WEIGHTS ENCODED IN THE CHROMOSOMES AND TRAINING THE NETWORKS WITH BACKPROPAGATION FOR 1, 2, AND 5 EPOCHS. RESULTS USING BALDWINIAN AND LAMARCKIAN VARIATIONS ON *all* THE MEMBERS OF THE POPULATION ARE PRESENTED. RESULTS THAT ARE SIGNIFICANTLY DIFFERENT ($\alpha = 0.05$) FROM THE HAND-DESIGNED NEURAL NETWORK ARE IN **bold** AND DAGGERS (†) INDICATE RESULTS SIGNIFICANTLY DIFFERENT THAN THE GA WITHOUT BP (FIRST COLUMN). TIMES ARE GIVEN IN CPU SECONDS.

network on the entire training data and test the network on the testing data that has not been used by the algorithm until this time. Table V shows the mean and standard deviations of the tests. POL2 and RCB2 are not used in the feature selection experiments, because both of their two features are necessary for a correct classification.

The results indicate that the GA feature selection produced significantly more accurate results than with all the features in four data sets and significantly worse results in only one case (Wine). In almost all cases, the GA successfully decreased the dimensionality of the data by selecting approximately half of the original features without significant decreases in accuracy.

The synthetic Random21 and Redundant21 data sets were included in the tests because they have known irrelevant and redundant features, so it is interesting to examine in more detail the results with these data. Recall that Random21 has nine relevant and 12 random features. Table VI shows the features selected in each of the 5x2cv folds on the Random21 data. Only in two cases, the GA missed one of the relevant features (and this resulted in substantially lower accuracy in those folds), while it selected the irrelevant features very infrequently. Even with all the irrelevant features in Random21, backpropagation performed well: The results of feature selection are not significantly different than using all the features.

In Redundant21, features 1, 5, and 9 are repeated four times each. For example, feature 1 also appears in positions 10, 13, 16 and 19. Missing one feature would cause the accuracy to degrade, but selecting a feature more than once does not cause problems. Table VII shows that in all cases the GA selected all the relevant features. Note that there is no preference to select feature 1 over feature 10, 13, 16 or 19. As long as one of these features is selected, the target

Domain	Accuracy		Num. of Features Original	Selected	
Breast Cancer	96.48	1.38	9	6.3	1.15
Credit-Australian	84.71	2.33	46	23.5	2.75
Credit-German	71.00	3.16	62	29.6	3.80
Diabetes-Pima	75.70	2.48	8	4.5	0.97
Heart-Cleveland	84.72	12.49	26	12.6	2.32
Housing	**68.65**	2.50	13	8.3	1.94
Ionosphere	87.00	1.82	34	16.2	1.39
Iris	93.60	2.41	4	2.3	0.67
Kr-vs-kp	**95.04**	1.98	74	35.1	2.84
Sonar	**72.98**	5.03	60	31.9	5.97
Wine	**86.06**	10.59	13	8.5	1.58
Random21	96.61	4.71	21	13	1.24
Redundant21	98.65	1.08	21	13.9	1.52
Mean Accuracy	85.44		30.07	15.82	
Median Time	1250				
Mean Time	3172				

TABLE V

RESULTS OF FEATURE SELECTION EXPERIMENTS. THE TABLE PRESENTS THE 5X2CV MEAN ACCURACY AND STANDARD DEVIATION, THE NUMBER OF ORIGINAL INPUTS, AND THE MEAN AND STANDARD DEVIATION OF THE NUMBER OF FEATURES SELECTED BY THE GA. RESULTS THAT ARE SIGNIFICANTLY DIFFERENT ($\alpha = 0.05$) FROM THE HAND-DESIGNED NEURAL NETWORK ARE HIGHLIGHTED IN **bold**. TIMES ARE GIVEN IN CPU SECONDS.

	Features																				
Selected	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	**Accuracy**
12	1	1	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	1	0	0	0	87.04
12	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	99.52
15	1	1	1	0	1	1	1	1	1	0	1	1	0	0	1	0	1	1	1	1	0	88.48
12	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	99.28
14	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	0	0	0	1	97.44
12	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	1	99.04
12	1	1	1	1	1	1	1	1	1	0	0	1	0	1	0	1	0	0	0	0	0	99.04
13	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	1	99.36
15	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	1	0	0	0	98.48
13	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	0	0	0	1	0	0	98.48
13.9	10	10	10	9	10	10	9	10	10	3	5	7	3	3	5	3	2	3	2	2	4	96.61

TABLE VI

FEATURES SELECTED IN EACH OF THE 5X2CV FOLDS ON THE RANDOM21 DATA, WHERE ONLY THE FIRST 9 FEATURES ARE RELEVANT. THE LAST LINE PRESENTS THE AVERAGE NUMBER OF FEATURES AND THE NUMBER OF TIMES EACH FEATURE WAS SELECTED AND THE AVERAGE ACCURACY.

concept will be completely specified.

C. Using EAs to Design the Networks

This section presents the results of experiments with four methods to use GAs to design neural networks.

	Features																				
Selected	1	2	3	4	5	6	7	8	9	1	5	9	1	5	9	1	5	9	1	5	9	**Accuracy**
15	0	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	0	1	0	0	98.40
13	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	0	0	1	0	1	1	99.20
15	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	1	0	0	0	99.36
12	0	1	1	1	0	1	1	1	0	1	0	1	1	1	0	1	1	0	0	0	0	98.72
14	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1	1	0	0	0	1	0	98.56
11	1	1	1	1	0	1	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	99.36
14	1	1	1	1	1	1	1	1	1	0	1	0	0	1	0	0	1	1	1	0	0	98.96
15	1	1	1	1	0	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	1	98.72
14	1	1	1	1	0	1	1	1	1	0	0	1	1	0	1	1	1	1	0	0	0	95.76
16	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	99.52
13	7	10	10	10	5	10	10	10	7	3	5	7	7	5	6	6	7	4	2	5	3	98.65

TABLE VII

FEATURES SELECTED IN EACH OF THE 5X2CV FOLDS ON THE REDUNDANT21 DATA. THE LAST LINE PRESENTS THE AVERAGE NUMBER OF FEATURES AND THE NUMBER OF TIMES EACH FEATURE WAS SELECTED AND THE AVERAGE ACCURACY.

C.1 Connectivity Matrix

In the first set of network design experiments, we used the GA to search for a connectivity matrix as described in section II-D. For each data set, we fixed the number of hidden units to those indicated in Table I. The neurons are numbered consecutively starting with the inputs and followed by the hidden units and outputs. The connectivity matrix is encoded by concatenating its rows. We allow direct connections between the inputs and the outputs, and therefore the string length is $l = (hidden + outputs) * inputs + hidden * outputs$ bits. For these longer problems we use multi-point crossover with probability 1.0 with $l/10$ crossover points and a mutation rate of $1/l$. The populations contained $\lfloor 3\sqrt{l} \rfloor$ individuals with an enforced minimum of 20 individuals. The GA was terminated after five generations of no improvement of the best solution or if a limit of 50 generations was reached.

After building a network using the connectivity matrix specified in the chromosomes, the fitness was calculated with five-fold crossvalidation experiments. In each fold, the network was trained with backpropagation using the number of epochs specified in Table I. The best connectivity matrix found during the GA run was used to build a final network, which was trained with the entire training data and tested on the previously unseen test data.

The results of these experiments are labeled **Matrix** in Table VIII. The experiments show that the accuracy was significantly lower than the hand-designed network in three artificial data sets (POL2, Random21 and Redundant21) and higher only in Kr-vs-kp.

C.2 Pruning

For the pruning experiments, the representation of the connectivity matrix and the GA parameters are the same as in the previous experiment. A fully-connected neural network was trained at the start of each experiment using the parameters shown in Table I. To calculate the fitness of each individual, a copy of the initial fully-connected network

Domain	Matrix		Pruning		Parameters		Grammar	
Breast Cancer	96.77	1.10	96.31	1.21	96.69	1.13	96.71	1.16
Credit-Australian	84.34	0.87	86.01	1.42	84.12	3.22	76.04	14.39
Credit-German	71.76	2.06	69.70	2.42	71.70	5.75	72.10	1.69
Diabetes-Pima	75.44	1.65	73.88	2.44	75.88	2.16	74.45	1.83
Heart-Cleveland	76.78	7.87	89.50	3.36	65.89	13.55	72.8	12.56
Housing	66.60	2.57	70.47	1.43	73.28	1.42	59.44	5.84
Ionosphere	87.06	2.14	83.66	1.90	85.58	3.08	88.03	1.55
Iris	92.40	2.67	92.40	1.40	91.73	8.26	92.93	3.08
Kr-vs-kp	**96.45**	1.00	**92.44**	0.98	**98.07**	0.95	**96.23**	1.68
Sonar	71.34	4.09	73.94	3.98	72.59	4.16	70.57	6.44
Wine	90.56	3.14	93.37	2.33	94.04	2.37	94.49	2.72
POL2	**68.67**	12.02	**72.02**	7.98	94.03	3.49	**68.43**	15.75
RCB2-8	79.24	5.42	**59.01**	12.14	92.01	3.51	**75.55**	4.29
Random21	**71.03**	6.43	94.05	3.32	80.08	4.77	95.28	1.52
Redundant21	**73.40**	9.02	94.86	4.19	**72.68**	17.05	96.95	1.13
Mean Accuracy	80.12		82.26		83.22		82.4	
Median Time	1499		29		2688		4155	
Mean Time	3227		73		13423		11246	

TABLE VIII

MEAN 5X2CV ACCURACIES OBTAINED BY DIFFERENT METHODS OF EVOLVING THE STRUCTURE OF A NEURAL NETWORK. RESULTS THAT ARE SIGNIFICANTLY DIFFERENT ($\alpha = 0.05$) FROM THE HAND-DESIGNED NEURAL NETWORK ARE HIGHLIGHTED IN **bold**. TIMES ARE GIVEN IN CPU SECONDS.

was pruned by setting to zero the weights corresponding to entries with zero in the connectivity matrix encoded in the chromosome. The fitness was the accuracy of the pruned network on the training data. The pruned networks were not retrained, but additional experiments reported elsewhere show that retraining with 1, 2 or 5 epochs of backpropagation has little effect on the accuracy [64].

The connectivity matrix encoded in the best individual found by the GA was used to prune the initial fully-connected network, which was tested on the previously unseen test data. The results in Table VIII show that in most cases pruning results in networks with an accuracy that is not significantly different than the fully-connected networks. Pruning resulted in worse results in two cases, and only in one case pruning improved the accuracy.

C.3 Finding Network Parameters

In our next application of GAs to network design, the GA was used to find the number of hidden units, the parameters for BP, and the range of initial weights as described in section II-D. The length l of each individual was 36 bits. The learning rate and the coefficient β for the activation function were encoded in five bits each and ranged in [0,1]. The number of hidden units was also encoded in five bits and could take values in [0,31]. Initial experiments used seven bits to encode the number of hidden units (allowing up to 127 hidden units). With five bits (and up to 31 hidden units), we observed a notable reduction in execution time without a degradation in accuracy. This is expected since the largest hand-designed network for these data sets has 15 hidden units (see Table I). The number of epochs was encoded in six bits. The upper and lower ranges for the initial weights were encoded in ten bits each and their ranges were [-10,0] and

[0,10], respectively.

After extracting the parameters from a chromosome, a network was built, initialized, and trained according to the parameters. As in previous experiments, the 5-fold crossvalidated accuracy estimate was used as the fitness of the networks.

The population of the GA contained 25 individuals and was initialized uniformly at random. Various population sizes ranging from 10 to 100 were tried on selected data sets with no noticeably improvement in accuracy. The GA used two-point crossover with probability 1.0 and the mutation rate was set at $1/l = 0.04$. As in all experiments, pairwise tournament selection without replacement is used. The GA runs were terminated after the best solution does not improve for five generations or a limit of 50 generations is reached.

The best parameters found by the GA were used to build and train a final network. The final network was trained on the entire training data and tested on the previously unseen testing data. As in all the experiments, the results reported are the averages on the testing data. The accuracy results are labeled **Parameters** in Table VIII. The results are significantly different from the hand-designed network in only two cases.

C.4 Graph Rewriting Grammar

We implemented Kitano's graph rewriting grammar method as described in section II-D. We limited the number of rewriting steps to 8, resulting in networks with at most 256 units. Since the chromosomes encode four 2×2 binary matrices for each of the 16 rules, the string length is 256 bits. The GAs used populations with 64 individuals, multi-point crossover with probability 1.0 and $l/10$ crossover points, and mutation with a rate of $0.004 \approx 1/l$.

After using the grammar encoded in the chromosomes to generate a connectivity matrix, a network was built from the matrix and trained with backpropagation. As before, the fitness of the each individual was determined by estimating the accuracy of the network with five-fold crossvalidation. The GA was stopped after five generations of no improvement or a limit of 50 generations. The best grammar found was used to produce a final network, which was trained on the entire training set and tested with the unseen test data.

The results labeled **Grammar** in Table VIII show that this network design method does not result in many significantly different results. Only in one data set the grammar-based method resulted in a better accuracy than the hand-designed network, and in two cases the grammar was significantly less accurate.

V. Conclusions

There are numerous combinations of EAs and NNs, but these methods have not been tested systematically and compared carefully to each other. This paper represents the most comprehensive study to date, presenting a comparison of eight combinations of EAs and NNs applied to 15 classification problems. We experimented with real- and binary-encoded EAs to train the networks and we tested the effect of refining weights with backpropagation and Lamarckian

and Baldwinian approaches. In addition, we experimented using EAs for feature subset selection and with four methods to design the structure of the networks.

In a few data sets, we found that some methods perform quite differently than the others. However, most of the time, the EA and NN combinations that we tried performed equally well and their accuracy was not significantly different from the accuracy reached by backpropagation. The results also suggest that the methods that use GAs to design the structure of networks perform slightly worse (but not significantly) than using GAs for training networks.

One conclusion of our study is that simple methods perform well and often better than more complex approaches. In particular, networks trained with simple backpropagation and simple binary-encoded GAs were competitive with the most complex methods examined here. Feature subset selection also demonstrated to be very useful. These algorithms are easy to implement and, because they have fewer parameters than the more sophisticated algorithms, are also easier to use in practice.

Innovative combinations of EAs and NNs are still being proposed. Evaluations of these new methods should be done carefully and systematically and should include a variety of data sets and different algorithms.

There are other interesting combinations of GAs and NNs that we did not include in this study, but appear promising and should be investigated in future work. For example, ensemble methods that combine several NNs are well known to improve the classification accuracy, and several combinations of EAs and ensembles of NNs have been proposed. Other interesting combinations use the EAs to evolve the structure and weights of the NN simultaneously.

Acknowledgments

We are grateful to the anonymous reviewers for their comments that helped improve the presentation of the paper. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

[1] J. Branke, "Evolutionary algorithms in neural network design and training – A review," in *Proceedings of the First Nordic Workshop on Genetic Algorithms and their Applications*, Jarmo T. Alander, Ed., Vaasa, Finland, 1995, pp. 145–163.

[2] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[3] P. A. Castillo, M. G. Arenas, J. J. Castillo-Valdivieso, J. J. Merelo, A. Prieto, and G. Romero, "Artificial neural networks design using evolutionary algorithms," in *Proceedings of the Seventh World Conference on Soft Computing*, 2002.

[4] S.G. Roberts and M. Turenga, "Evolving neural network structures," in *International Conference on Genetic Algorithms and Neural Networks*, D.W. Pearson, N.C. Steele, and R.F. Albrecht, Eds., New York, 1995, pp. 96–99, Springer-Verlag.

[5] A. A. Siddiqi and S. M. Lucas, "A comparison of matrix rewriting versus direct encoding for evolving neural networks," in *Proceedings of the 1998 International Conference on Evolutionary Computation*, Piscataway, NJ, 1998, pp. 392–397, IEEE Press.

[6] M. A. Grönross, "Evolutionary design of neural networks," M.S. thesis, University of Turku, Finland, 1998.

[7] E. Cantú-Paz and C. Kamath, "Evolving neural networks to identify bent-double galaxies in the FIRST survey," *Neural Networks*, vol. 16, no. 3–4, pp. 507–517, 2003.

[8] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *International Workshop on Combinations of Genetic Algorithms and Neural Networks*. 1992, pp. 1–37, IEEE Computer Society Press.

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

[10] T. Bäck, *Evolutionary algorithms in theory and practice*, Oxford University Press, New York, 1996.

[11] H. Mühlenbein, "How genetic algorithms really work: I.Mutation and Hillclimbing," In Männer and Manderick [65], pp. 15–25.

[12] T. P. Caudell and C. P. Dolan, "Parametric connectivity: Training of constrained networks using genetic algorithms," In Schaffer [66], pp. 370–374.

[13] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, San Mateo, CA, 1989, pp. 762–767, Morgan Kaufmann.

[14] D. Whitley and T. Hanson, "Optimizing neural networks using faster, more accurate genetic search," In Schaffer [66], pp. 391–397.

[15] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biological Cybernetics*, vol. 63, pp. 487–493, 1990.

[16] H. Kitano, "Empirical studies on the speed of convergence of neural network training using genetic algorithms," *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 789–795, 1990.

[17] A. Skinner and J. Q. Broughton, "Neural networks in computational material science: training algorithms," *Modelling and Simulation in Material Science and Enginnering*, vol. 3, pp. 371–390, 1995.

[18] N. Kadaba and K. E. Nygard, "Improving the performance of genetic algorithms in automated discovery of parameters," in *Machine Learning: Proceedings of the Seventh International Conference*, B. Porter and R. Mooney, Eds., San Mateo, CA, 1990, pp. 140–148, Morgan Kaufmann.

[19] N. J. Radcliffe, *Genetic neural networks on MIMD computers*, Unpublished doctoral dissertation, University of Edinburgh, Scotland, 1990.

[20] D. Thierens, J. Suykens, J. Vandewalle, and B. De Moor, "Genetic weight optimization of a feedforward neural network controller," in *Proceedings of the Conference on Neural Nets and Genetic Algorithms*. 1991, pp. 658–663, Springer Verlag.

[21] P. J. B. Hancock, "Recombination operators for the design of neural nets by genetic algorithm," In Männer and Manderick [65], pp. 441–450.

[22] D. Thierens, *Analysis and design of genetic algorithms*, Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1995.

[23] K. W. C. Ku and M. W. Mak, "Exploring the effects of Lamarckian and Baldwinian learning in evolving recurrent neural networks," in *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*, Piscataway, NJ, 1997, pp. 617–622, IEEE.

[24] F. Gruau and D. Whitley, "Adding learning to the cellular development of neural networks: Evolution and the baldwin effect," *Evolutionary computation*, vol. 1, no. 3, pp. 213–233, 1993.

[25] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in *Parallel Problem Solving fron Nature, PPSN III*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds., Berlin, 1994, pp. 6–15, Springer-Verlag.

[26] C. R. Houck, J. A. Joines, M. G. Kay, and J. R. Wilson, "Empirical investigation of the benefits of partial lamarkianism," *Evolutionary Computation*, vol. 5, no. 1, pp. 31–60, 1997.

[27] B. Julstrom, "Comparing darwinian, baldwinian, and lamarckian search in a genetic algorithm for the 4-cycle problem," in *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference (GECCO'99)*, Scott Brave and Annie S. Wu, Eds., 1999, pp. 134–138.

[28] R. Kohavi and G. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[29] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," *Pattern Recognition Letters*, vol. 10, pp. 335–347, 1989.

[30] F. Z. Brill, D. E. Brown, and W. N. Martin, "Genetic algorithms for feature selection for counterpropagation networks," Tech. Rep. No. IPC-TR-90-004, University of Virginia, Institute of Parallel Computation, Charlottesville, 1990.

[31] T. W. Brotherton and P. K. Simpson, "Dynamic feature set training of neural nets for classification," in *Evolutionary Programming IV*, J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, Eds., Cambridge, MA, 1995, pp. 83–94, MIT Press.

[32] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," *IEEE Intelligent Systems*, vol. 13, pp. 44–49, 1998.

[33] M. Ozdemir, M. J. Embrechts, F. Arciniegas, C. M. Breneman, L. Lockwood, and K. P. Bennett, "Feature selection for in-silico drug design using genetic algorithms and neural networks," in *IEEE Mountain Workshop on Soft Computing in Industrial Applications*. 2001, pp. 53–57, IEEE Press.

[34] J. Bala, K. De Jong, J. Huang, H. Vafaie, and H. Wechsler, "Using learning to facilitate the evolution of features for recognizing visual concepts," *Evolutionary Computation*, vol. 4, no. 3, pp. 297–311, 1996.

[35] J. D. Kelly and L. Davis, "Hybridizing the genetic algorithm and the K nearest neighbors classification algorithm," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker, Eds., San Mateo, CA, 1991, pp. 377–383, Morgan Kaufmann.

[36] W. F. Punch, E. D. Goodman, M. Pei, L. Chia-Shun, P. Hovland, and R. Enbody, "Further research on feature selection and classification using genetic algorithms," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed., San Mateo, CA, 1993, pp. 557–564, Morgan Kaufmann.

[37] H. Vafaie and K. A. De Jong, "Robust feature selection algorithms," in *Proceedings of the International Conference on Tools with Artificial Intelligence*. 1993, pp. 356–364, IEEE Computer Society Press.

[38] I. Inza, P. Larrañaga, R. Etxeberria, and B. Sierra, "Feature subset selection by Bayesian networks based optimization," *Artificial Intelligence*, vol. 123, no. 1-2, pp. 157–184, 1999.

[39] E. Cantú-Paz, "Feature subset selection by estimation of distribution algorithms," in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, Eds., San Francisco, CA, 2002, pp. 303–310, Morgan Kaufmann Publishers.

[40] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," In Schaffer [66], pp. 379–384.

[41] R. Belew, J. McInerney, and N. Schraudolph, "Evolving networks: Using the genetic algorithm with connectionist learning," in *Proceedings of the Second Artificial Life Conference*, New York, NY, 1991, pp. 511–547, Addison-Wesley.

[42] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing*, vol. 14, pp. 347–361, 1990.

[43] R. Reed, "Pruning algorithms—a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[44] P. J. B. Hancock, "Pruning neural networks by genetic algorithm," in *Proceedings of the 1992 International Conference on Artificial Neural Networks*, I. Aleksander and J. Taylor, Eds., Amsterdam, Netherlands, 1992, vol. 2, pp. 991–994, Elsevier Science.

[45] B. LeBaron, "An evolutionary bootstrap approach to neural network pruning and generalization," unpublished working paper, 1997.

[46] D. Whitley and C. Bogart, "The evolution of connectivity: Pruning neural networks using genetic algorithms," Tech. Rep. CS-89-113, Colorado State University, Department of Computer Science, Fort Collins, 1989.

[47] S. J. Marshall and R. F. Harrison, "Optimization and training of feedforward neural networks by genetic algorithms," in *Proceedings on the Second International Conference on Artifical Neural Networks and Genetic Algorithms*. 1991, pp. 39–43, Springer Verlag.

[48] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, vol. 4, no. 4, pp. 461–476, 1990.

[49] J. W. Boers and H. Kuiper, "Biological metaphors and the design of modular artificial neural networks," M.S. thesis, Leiden University, The Netherlands, 1992.

[50] F. Gruau, "Genetic synthesis of boolean neural networks with a cell rewritting developmental process," in *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, D. Whitley and J. D. Schaffer, Eds., Los Alamitos, CA, 1992, pp. 55–74, IEEE Computer Society Press.

[51] S. Nolfi, J. L. Elman, and D. Parisi, "Learning and evolution in neural networks," *Adaptive Behavior*, vol. 3, no. 1, pp. 5–28, 1994.

[52] X. Yao, Ed., *Proceedings of the First IEEE Symposium on Combinations of Evolutionary Algorithms and Neural Networks*, IEEE Press, 2000.

[53] L. Prechelt, "A quantitative study of experimental evaluations of neural network learning algorithms: current research practice," *Neural Networks*, vol. 9, no. 3, pp. 457–462, 1996.

[54] C.L. Blake and C.J. Merz, "UCI repository of machine learning databases," 1998.

[55] E. Cantú-Paz and C. Kamath, "Using evolutionary algorithms to induce oblique decision trees," in *Proceedings of the Genetic and Evolutionary Computation Conference 2000*, D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, L. Parmee, and H.-G. Beyer, Eds., San Francisco, CA, 2000, pp. 1053–1060, Morgan Kaufmann Publishers.

[56] D. Opitz and R. Maclin, "Popular ensemble methods: an empirical study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.

[57] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural Computation*, vol. 10, no. 7, pp. 1895–1924, 1998.

[58] E. Alpaydin, "Combined 5×2cv F test for comparing supervised classification algorithms," *Neural Computation*, vol. 11, pp. 1885–1892, 1999.

[59] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations," *Evolutionary Computation*, vol. 7, no. 3, pp. 231–253, 1999.

[60] K. Deb, A. Anand, and D. Joshi, "A computationally efficient evolutionary algorithm for real-parameter optimization," *Evolutionary Computation*, vol. 10, no. 4, pp. 371–395, 2002.

[61] D. E. Goldberg, "The hurdle, and the sweet spot: Genetic algorithms as a computational model of innovation," *Proceedings of International Workshop on Soft Computing in Industry 1999*, p. 223, 1999.

[62] J. Reunanen, "Overfitting in making comparisons between variable selection methods," *Journal of Machine Learning Research*, vol. 3, pp. 1371–1382, 2003.

[63] C. Ambroise and G. J. McLachlan, "Selection bias in gene extraction on the basis of microarray gene-expression data," *Proceedings of the National Academy of Sciences*, vol. 99, no. 10, pp. 6562–6566, 2002.

[64] E. Cantú-Paz, "Pruning neural networks with distribution estimation algorithms," in *Genetic and Evolutionary Computation Conference – GECCO-2003*, E. Cantú-Paz et al., Ed., Berlin, 2003, pp. 790–800, Springer-Verlag.

[65] R. Männer and B. Manderick, Eds., *Parallel Problem Solving from Nature, 2*, Amsterdam, 1992. Elsevier Science.

[66] J. D. Schaffer, Ed., *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufmann.