Multi-Objective Evolutionary Optimization of Exemplar-Based Classifiers: A PNN Test Case

Talitha Rubio, Tiantian Zhang, Michael Georgiopoulos, Assem Kaylani

Abstract—**In this paper the major principles to effectively design a parameter-less, multi-objective evolutionary algorithm that optimizes a population of probabilistic neural network (PNN) classifier models are articulated; PNN is an example of an exemplar-based classifier. These design principles are extracted from experiences, discussed in this paper, which guided the creation of the parameter-less multi-objective evolutionary algorithm, named MO-EPNN (multi-objective evolutionary probabilistic neural network). Furthermore, these design principles are also corroborated by similar principles used for an earlier design of a parameter-less, multi-objective genetic algorithm used to optimize a population of ART (adaptive resonance theory) models, named MO-GART (multi-objective genetically optimized ART); the ART classifier model is another example of an exemplar-based classifier model. MO-EPNN's performance is compared to other popular classifier models, such as SVM (Support Vector Machines) and CART (Classification and Regression Trees), as well as to an alternate competitive method to genetically optimize the PNN. These comparisons indicate that MO-EPNN's performance (generalization on unseen data and size) compares favorably to the aforementioned classifier models and to the alternate genetically optimized PNN approach. MO-EPPN's good performance, and MO-GART's earlier reported good performance, both of whose design relies on the same principles, gives credence to these design principles, delineated in this paper.**

Keywords: Exemplar Based Classifiers, Neural Networks, Multi-Objective Optimization, Multi-Objective Evolutionary Algorithms, Probabilistic Neural Network

I. Introduction

EXEMPLAR-based classifiers are pattern recognizers that encode their accumulated evidence with the use of exemplars. The exemplars in EBC's (exemplar based classifiers) are formulated via clustering of training patterns associated with the same class label. In essence, these classifiers use exemplars to summarize training data belonging to the same class and then utilize a similarity or proximity measure to classify a previously unseen test pattern. The aforementioned summarization of input patterns corresponds to a form of local learning, since the information regarding a cluster of patterns is represented by a single exemplar rather than being distributed. Furthermore, this local learning attribute of an EBC extends to the prediction of a class label of an unseen input pattern, where the prediction of the label of such a pattern relies only on the information that a few exemplars (in the vicinity of this input pattern) convey. Examples of EBC's, in the domain of neural network classifiers, include ART classifiers (Fuzzy ARTMAP (FAM) [1], Ellipsoidal ARTMAP (EAM) [2], Gaussian ARTMAP (GAM) [3]), Radial Basis Function Neural Networks (RBFNNs) [4], and PNNs [5]. Other examples of EBC's are the k-nearest neighbor [6], and the Parzen Window Classifier [7].

A perennial problem in designing an EBC is how to choose the number, location, and quite often radius of influence of the exemplars from the available training data. For instance, in a recent paper [8], Kaylani, et al., used a multi-objective evolutionary optimization approach to determine the number, location, and radius of influence of the exemplars pertinent to three popular ART classifiers, FAM [1], EAM [2], and GAM [3]; in their approach they focused on minimizing the generalization error and the size of the resulting ART classifier. The authors in [8] called their design MO-GART (multi-objective genetically optimized ART), explained how MO-GART was designed and demonstrated that its performance is very competitive compared to other classifier models, such as CART [9] and SVMs [10]. This assessment relied on experimentations with eleven different classification problems.

Using genetic algorithms to optimize performance has not been limited to classifiers, such as ART. Ishibuchi and his colleagues have produced a good number of papers, dated back in 1994 [11], where genetic algorithms were used for the first time to design fuzzy classifiers. In one of their most recent publications [12], Ishibuchi and his colleagues examine the interpretability-accuracy tradeoff in fuzzy rule-based classifiers using a multi-objective fuzzy genetics-based machine learning (GBML) algorithm. There is a very rich literature on multi-objective optimization of fuzzy classifiers, pioneered by Professor Ishibuchi and his colleagues, but a literature that has many more contributors as the *Evolutionary Multi-Objective Optimization of Fuzzy Rule-Based Systems Bibliography* page by M. Cococcioni demonstrates. Furthermore, Gonzalez, et al., [13] have designed a multi-objective radial basis function neural network (RBF-NN) using a number of specialized genetic operators. In their paper they compared the performance of certain combinations of these operators in solving function

Manuscript received February 10, 2011.

T. Rubio is with the Department of EECS at the University of Central Florida. T. Rubio is supported by NSF grants 0717680, 0647120, 064718.

T. Zhang is with the Department of EECS at the University of Central Florida. T. Zhang I supported by NSF grants 0717680, 0647120, 0837307.

M. Georgiopoulos is with the Department of EECS at the University of Central Florida (Phone: 407-823-5338, Fax: 407-823-5835, E-Mail: michaelg@ucf.edu). M. Georgiopoulos is supported by NSF grants 0917680, 0837332, 0525429 and 0963146.

Assem Kaylani is with InCube, FZCO, Dubai, UAE.

approximation problems. In their comparisons the measures of merit was the accuracy of the resulting RBF-NN and its complexity which was directly correlated with the number of hidden nodes used by the RBF-NN.

In this paper we demonstrate that the principles that were instrumental for the design of MO-GART can also be applied for the design of multi-objective, evolutionary algorithms that optimize other EBC models, such as the PNN [5]. Evolutionary optimization of the PNN has appeared in the literature before, such as the work by Mao, et al. [14]. In [14] the authors, first use a genetic algorithm to define the smoothing parameters of the exemplars (pattern layer neurons) used in the PNN design; then a forward regression orthogonal algorithm is used to determine suitable pattern layer neurons. In [14] the authors, report a small PNN structure with good accuracy on a simulated problem, the IRIS problem, and a face recognition problem.

The contributions of this paper are various: First we design a multi-objective (generalization error and size are the two objectives to be minimized) evolutionary algorithm to optimize a population of PNNs; we name the resulting algorithm, *MO-EPNN* (multi-objective evolutionary PNN). Secondly, we compare the performance of MO-EPNN with the performance of other popular classifier models (CART [9], SVM [10], Mao's PNN [14]) on eight different classification problems and show that MO-EPNN's performance is competitive. Thirdly, by identifying the similar principles used for the design of MO-GART and MO-EPNN we postulate principles that are useful for the design of other EBC models.

The organization of the paper is as follows: In Section II we provide background information about the PNN classifier, and evolutionary multi-objective optimization approaches to solve multi-objective optimization problems. In Section III, we introduce the MO-EPNN and its important components. In Section IV we describe the datasets (classification problems) used, and the experiments conducted that assess MO-EPNN's performance; in the same section we compare MO-EPNN's performance with the performance of other popular classifiers and we make pertinent observations. In Section V, we postulate the principles needed for the effective design of a multi-objective evolutionary algorithm that optimizes an EBC, relying on the experiences with the MO-EPNN (this paper) and MO-GART [8]. In Section VI, we provide a summary of the paper and point out, once more, the contributions of the work.

II. BACKGROUND INFORMATION

A. Brief Review of the PNN

Consider a classification problem where the pattern data, designated by the vector $\mathbf{x}$ belong to different classes, designed by the letters $A, B, C,...,$ etc. The Bayesian classifier is the classifier that minimizes the probability of misclassifying the labels of unseen data. The Bayesian classifier chooses as the predicted label of an unseen pattern $\mathbf{x}$ the label l that maximizes the following a-posteriori probability.

$$P(l \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid l) \cdot P(l)}{p(\mathbf{x})} \tag{1}$$

In order to calculate the above probabilities for every label $l = A, B, C,...,$ one needs to compute the class conditional probabilities $p(\mathbf{x} \mid l)$ for every l, and the a-priori probabilities $P(l)$. The a-priori probabilities $P(l)$ can be estimated from the available training data. The class conditional probabilities $p(\mathbf{x} \mid l)$ can also be estimated using the training data, by using an approximation for the probability density function formula, suggested by Parzen [7], and depicted below, for the class label $l = A$.

$$p(\mathbf{x} \mid A) = \frac{1}{(2\pi)^{D/2}} \cdot \frac{1}{N_{tr}(A)} \cdot \sum_{t \in S_{tr}(A)} \prod_{d=1}^{D} \frac{1}{\boldsymbol{\sigma}_t^A(d)} \exp - \left(\frac{(\mathbf{x}(d) - \mathbf{x}_t^A(d))^2}{2(\boldsymbol{\sigma}_t^A(d))^2} \right) \tag{2}$$

where in the above equation D is the dimensionality of the pattern vector $\mathbf{x}$, $N_{tr}(A)$ is the number of points in the training set that are of label A, $S_{tr}(A)$ is the set of indices that correspond to the training points that are of label A, $\mathbf{x}_t^A$ is the training pattern of index t, that is of label A, and $\mathbf{x}_t^A(d)$ is the d-th component of this pattern, while $\boldsymbol{\sigma}_t^A$ is the corresponding spread parameter vector (standard deviation vector) associated with training pattern $\mathbf{x}_t^A$, and $\boldsymbol{\sigma}_t^A(d)$ is the d-th component of this spread parameter vector. The right hand side (RHS) of (2) is actually an approximation of $p(\mathbf{x} \mid A)$ but for simplicity we still use the $p(\mathbf{x} \mid A)$ notation.

The PNN is a neural network implementation that accepts an input pattern vector $\mathbf{x}$ and produces as an output the label l that maximizes the product

$$p(\mathbf{x} \mid l) \cdot P(l) \tag{3}$$

where $p(\mathbf{x} \mid l)$ is approximated by the RHS of (2), and $P(l)$ is approximated by the ratio $\frac{N_{tr}(l)}{N_{tr}}$, where N_{tr} are the number of points in the training set.

B. Brief Overview of Multi-Objective Optimization

Many real world problems involve simultaneous optimization of conflicting objectives. This is the basic challenge of multi-objective optimization research (see [15] and [16] for an overview of multi-objective evolutionary

algorithms (MOEA)). With conflicting multiple objectives, there is no single optimal solution, but rather, there are a set of good solutions with varying degrees of merit. Evolutionary algorithms (EAs) are suitable for solving multi-objective optimization problems because EAs are population based search algorithms, and as such they can find, in a single run, multiple good solutions on the surface defined by the multiple objectives that are to be optimized.

Formally the multi-objective optimization problem can be stated as follows: Optimize (without loss of generality, we can think of optimization as equivalent with minimization) the vector function $f(\mathbf{x})$ of L objectives by finding a solution $\mathbf{x}^*$ from the feasible domain of solutions, F, such that the resulting vector function

$$f(\mathbf{x}^*) = [f_1(\mathbf{x}^*), f_2(\mathbf{x}^*),..., f_L(\mathbf{x}^*)]^T \quad (4)$$

has component values that are acceptable to the user. The set of functions $f_1(\mathbf{x}), f_2(\mathbf{x}),...$ are usually of conflicting nature; in other words it is rare to find a solution $\mathbf{x}^*$ that minimizes all the components of the function f. Therefore, several solutions may exist that optimize one or more objectives, but not all of them, resulting in a collection of tradeoff solutions. The minimum set of such a collection is called the Pareto optimal set. A solution is called Pareto-optimal if there is no other solution that would decrease one objective without causing a simultaneous increase in at least one other objective. The Pareto-optimal set is also referred to as the set of non-dominated solutions. Formally a solution $\mathbf{x}^*$ is said to be non-dominated if there is no other solution $\mathbf{x} \in F$ such that

$$\forall i \quad f_i(\mathbf{x}) \le f_i(\mathbf{x}^*) \text{ and } \exists i : f_i(\mathbf{x}) < f_i(\mathbf{x}^*) \quad (5)$$

The main focus in MOEA research is to minimize the distance of the generated solutions to the true Pareto set and to maximize the diversity of the discovered Pareto set. A good Pareto set may be obtained by appropriate guiding of the search process through careful design of the selection operator and the fitness assignment strategies. Special care is also taken to prevent non-dominated solutions from being eliminated in the evolutionary process. Multi-objective optimization using Evolutionary Algorithms follows the same general procedure (as single objective optimization), and it is listed below.

Generate Initial Population ( )
Repeat
  Fitness Evaluation ( )
  Selection ( )
  Reproduction ( )
until stopping criterion is satisfied
return Pareto-Optimal solutions

Fig. 1. Pseudo-code of a Multi-Objective Evolutionary Algorithm (MOEA)

In this paper an MOEA is used to generate the Pareto-optimal solutions of a population of PNNs. The two objectives that are to be optimized (minimized) are the PNN's generalization error and the PNN's size (defined as the number of training patterns used in its design).

III. Design of the MO-EPNN

We assume from this point on that when a classification problem is provided to us a training set, a validation set and a test set are furnished. We use the training set to create the initial population of the PNNs that are utilized by the MOEA, we use the validation set to reduce the size and error of this initial population of the PNNs during their evolution, and we finally assess the performance of the final approximate Pareto-optimal set of solutions, using the test set.

A. Encoding of the PNNs in the MO-EPNN

The first task needed to be accomplished is to encode the PNNs as chromosomes to facilitate their evolution from one generation to the next. We therefore assume that every PNN is represented by collection of categories. Each category consists of the pattern vector, its associated spread parameter vector, and its corresponding label. For instance, the category $\mathbf{x}_t, \boldsymbol{\sigma}_t, A$ is the category associated with t-th training pattern $\mathbf{x}_t$, whose label is A, and its spread parameter vector is $\boldsymbol{\sigma}_t$. To simplify the notation we represent the gene corresponding to this PNN category as

$$\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A \quad (6)$$

Consequently the part of the chromosome that encapsulates all the training input patterns of label A, is given below.

$$\mathbf{x}_1^A, \boldsymbol{\sigma}_1^A; \mathbf{x}_2^A, \boldsymbol{\sigma}_2^A; ...; \mathbf{x}_{N_t^A}^A, \boldsymbol{\sigma}_{N_t^A}^A \quad (7)$$

where N_t^A is the number of training patterns of class A, used by the PNN. A complete PNN chromosome consists of the training patterns and their associated spread parameter vectors from all the class labels, such as $A, B, C, ...$

B. Initial Population of the MO-EPNN

The MO-EPNN is initialized by using a population of PNNs, whose training patterns and associated spread parameters are chosen randomly. Assume that the population size of PNNs used in the MO-EPNN's evolution is Pop_{size}. Then, the i-th, $1 \le i \le Pop_{size}$, PNN in this initial population, designated, PNN_i^0, chooses its training patterns from the training set as follows: If $S_{tr}(A)$ is the index set of training patterns of label A, then each one of these indices is chosen to be used as a training pattern by PNN_i^0 with

probability p_i, where p_i is a number uniformly distributed in the interval $[0, 1]$. Training patterns of labels $B, C, \ldots$ are chosen in a similar fashion. After the training patterns of every label are chosen by PNN_i^0 their corresponding spread parameter vectors $\boldsymbol{\sigma}$ are chosen. We use eight schemes to choose the parameter vectors $\boldsymbol{\sigma}$, all of which are described below; note that Scheme *B* encompasses 5 sub-schemes. Each one of these schemes is explained below. First, we define the discrete set

$$SS = \{0.05, 0.075, 0.1, 0.15, 0.2\} \tag{8}$$

Scheme A: Consider that a training pattern $\mathbf{x}_t^A$ has been chosen to be one of the categories of PNN_i^0. A $\sigma_{\max}$ value for this category is chosen to be a number uniformly distributed over the discrete set *SS*. Then, each spread parameter $\boldsymbol{\sigma}_t^A(d)$ of the training pattern $\mathbf{x}_t^A$ is chosen independently to be a value uniformly distributed in the interval $(0, \sigma_{\max})$.

Scheme B: Consider that a training pattern $\mathbf{x}_t^A$ has been chosen to be one of the categories of PNN_i^0. A $\sigma_{\max}$ value for this category is chosen to be one (deterministically) of the numbers from the set *SS*. Then, each spread parameter $\boldsymbol{\sigma}_t^A(d)$ of the training pattern $\mathbf{x}_t^A$ is chosen independently to be a value uniformly distributed in the interval $(0, \sigma_{\max})$. This scheme gives rise to 5 distinct sub-schemes, referred to as *Scheme_B_0.05*, *Scheme_B_0.075*, *Scheme_B_0.1*, *Scheme_B_0.15* and *Scheme_B_0.2*, depending on the specific value that $\sigma_{\max}$ is chosen (deterministically) to be.

Scheme C: Consider that a training pattern $\mathbf{x}_t^A$ has been chosen to be one of the categories of PNN_i^0. A $\sigma_{\max}$ value for this category is chosen to be a number uniformly distributed over the set *SS*. Then, each spread parameter $\boldsymbol{\sigma}_t^A(d)$ of the training pattern $\mathbf{x}_t^A$ is chosen to be equal to this $\sigma_{\max}$ value.

Scheme D: For every class and every dimension find the minimum and the maximum values of the training patterns, i.e., you find $\mathbf{x}_{t,\min}^A(d)$ and $\mathbf{x}_{t,\max}^A(d)$. Define the range of spread values for dimension d of class A data, as shown below:

$$\boldsymbol{\sigma}_{range}^A(d) = \frac{\mathbf{x}_{t,\max}^A - \mathbf{x}_{t,\min}^A}{2} \tag{9}$$

Then, find the split interval of spread values for dimension d of class A data, as shown below:

$$SI^A(d) = \frac{\boldsymbol{\sigma}_{range}^A(d)}{S} \tag{10}$$

Now, define S spread values for dimension d of class A data, as shown below:

$$\boldsymbol{\sigma}_s^A(d) = s \cdot SI^A(d) \quad 1 \le s \le S \tag{11}$$

The spread parameter $\boldsymbol{\sigma}_t^A(d)$ of the training pattern $\mathbf{x}_t^A$ is chosen randomly from the discrete set $\{\boldsymbol{\sigma}_s^A(d)\}_{s=1}^S$. In Scheme *D*, we chose $S = 5$, which allows the spread vector across every dimension of patterns of a specific class to assume 5 distinct values.

Each one of the above schemes to initialize the spread parameters is not as arbitrary as it seems. For instance, the assumption made for Schemes *A, B* and *C* is that the input patterns are normalized across every dimension, so that their values lie in the interval $[0,1]$. Hence, choosing the spread parameters in Schemes *A-C* by utilizing the discrete set of values in *SS* (see (8)) guarantees that the spread parameter values of every PNN category is a portion of the maximum range of values over which the input pattern values lie, a logical choice for PNN classifiers. Scheme *D* is attempting to do something more elaborate by considering the fact that although the range of pattern values could be the entire $[0,1]$ interval, it is plausible that the data of a particular class have values that reside over a smaller range that could differ from dimension to dimension.

C. Selection in MO-EPNN; Fitness Function

As it can be seen in Figure 1, where the MOEA pseudo code is presented an important component of the MOEA is the selection process and the associated fitness function that allows us to select a member of the PNN population over another member.

As mentioned above the MO-EPPN algorithm starts by generating an initial population, $P(0)$ of PNNs, each one of them trained with a subset of the available training data and with randomly chosen spread parameter values.

Also, MO-EPNN initializes an empty secondary population, $AR(0)$, that will be used to store non-dominated solutions found during the evolution. In each generation, each solution in the population is evaluated according to each objective function. That is, the error rate of each PNN is evaluated by running it against the validation set. The second objective, complexity, is represented by the number of categories present in each network (the categories in the PNN correspond to the training patterns used by the PNN in its design, i.e., (7)). Once networks in population P are evaluated, the archive AR is updated by adding to it the solutions in P that are non-dominated by solutions in AR. Also, solutions in AR that are now dominated by solutions just added from P are removed from the archive AR. This mechanism ensures elitism. The algorithm runs for a maximum number of

generations, denoted as $G_{\max}$; $G_{\max}$ is not fixed a-priori but it is determined by how the MO-EPNN fares with each dataset that it is applied to (more details about the stopping criterion are provided in Section III.E).

The selection process creates a temporary population P', where the parent chromosomes used to create the next generation are selected. The chromosomes in the archive AR and population P are assigned fitness values based on a dominance relationship. In this scheme each individual is assigned a strength value that is equal to the number of solutions it dominates. After that, a raw fitness, $R(x)$, is assigned for each individual to be the sum of the strengths of all its dominators in both AR and P. The raw fitness is then adjusted as follows. For each individual, x, the Euclidean distance, in objective space, to the k-th nearest neighbor is found and denoted as $dis_k(x)$. The objective space is the space defined by the "Error" and "Size" of a member of the population (PNN). The "Error" is the PNN error on the validation set, and the "Size" is the number of training patterns used by the PNN. The value of k is chosen to be the square root of the sum of the size of the archive and population. The fitness of each individual is then calculated using the following equation:

$$Fit(x) = R(x) + \frac{1}{dis_k(x) + 2} \tag{12}$$

The parents are then chosen using a deterministic binary tournament selection with replacement, as follows: For each parent, randomly select two chromosomes from the combined set of AR and P, and choose, the chromosome with the smallest fitness value. Boundary solutions, which are networks with smallest error rate and smallest size, are ensured to be copied in the set P' of parents. The more detailed MO-EPNN code is now provided in the following figure.

$P(0) \leftarrow$ Generate initial population of PNNs ( )
$AR(0) \leftarrow$ Initialize initial archive ( )
for $g \leftarrow 1$ **to** $Gen_{\max}$ **do**
  Evaluation ( )
  Update-Archive $(P(g), AR(g))$;
  If stopping criteria met then exit **for**;
  $P'(g) \leftarrow$ Selection $(P(g), AR(g))$;
  $P(g) \leftarrow$ Reproduction $P'(g)$;
end
return $AR(g)$

Fig. 2. Pseudo-Code of the MO-EPNN Algorithm for one initial population corresponding to a specific initialization scheme

It is worth noting that the MO-EPNN algorithm that we described above is the SPEA2 algorithm developed by Zitzler [17]. The only difference is that we do not set a limit on the size of the archive created by MO-EPNN as the original SPEA2 algorithm sets.

D. Reproduction Operators in MO-EPNN; Prune Operator

As the MO-EPNN pseudo-code in Figure 2 demonstrates after the temporary population of parents P' is produced reproduction operators are applied to the P' population to produce the new and improved population of off-springs. In a typical evolutionary optimization process the typical reproduction operators are cross-over and mutation. For our problem of interest, where our focus is the tandem reduction of the PNN's generalization error and size we introduce a unique reproduction operator called the *Prune* operator. This operator was introduced for the first time in the literature in the successful design of MO-GART [8]: MO-GART outperformed popular classification models, such as SVMs and CART.

The effect that the Prune operator in MO-EPNN has on the P' population is to simply prune categories from each member of the population, as follows: The category of every member of the population has an associated probability of being pruned that depends on the category's confidence factor, CF. A category of a member in the MO-EPPN population is a training pattern used in the design of this member's classification model. The confidence factor of a category in a PNN is calculated at every generation and its value ranges in the interval $[0, 1]$; a high value of a category's confidence factor is an indication that this category is very valuable for the good performance of the PNN (i.e., its low generalization error on the validation set) and it should have a low probability of being pruned, while a low value of a category's confidence factor is an indication that this category is not very valuable for the good performance of the PNN and it should have a high probability of being pruned.

Let us now define the CF of category of a PNN in the MO-EPPN population. We assume, without loss of generality, that this PNN's category is defined, as (6) has suggested earlier, by the vectors $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$. In order to define the confidence factor of this PNN's category we first introduce four relevant parameters, $\mathrm{Re}\, w_t^A(\mathbf{x}_v)$, $CSel_t^A(\mathbf{x}_v)$, $Pen_t^A(\mathbf{x}_v)$, $ESel_t^A(\mathbf{x}_v)$, referred to as *Reward*, *Correct Selectivity*, *Penalty*, and *Erroneous Selectivity*, respectively.

The reader who wants to omit the details of the definitions of the $\mathrm{Re}\, w_t^A(\mathbf{x}_v)$, $CSel_t^A(\mathbf{x}_v)$, $Pen_t^A(\mathbf{x}_v)$, $ESel_t^A(\mathbf{x}_v)$ parameters can go directly to (19) where the CF of the category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ is defined in terms of its goodness factor and its badness factor. The goodness factor of a category is expressed as a weighted sum of the normalized reward and correct selectivity of a category (see (15) for more details). The badness factor of a category is expressed as a weighted sum of the normalized penalty and erroneous selectivity of a

category (see (18) for more details). Reward and Correct Selectivity values of a category are expected to be high for categories that are immersed in dense clusters of points that are of the same label as the category. Penalty and Erroneous Selectivity values of a category are expected to be high for categories immersed in dense clusters of points that are of different label than the label of the category. Hence, the calculated category confidence factor seems to favor categories that are immersed in clusters that are dense of points of the same label and non-dense of points of a different label.

The *Reward* parameter, $\mathrm{Re}\, w_t^A$, is calculated for all the class-A validation points $\mathbf{x}_v$ from the validation set S_v that are correctly classified by the PNN to which the category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ belongs. In particular, the Reward parameter, $\mathrm{Re}\, w_t^A(\mathbf{x}_v)$, corresponding to a class-A validation point $\mathbf{x}_v$, correctly classified by the PNN to which this category belongs, is a normalized measure of how much this category contributes to $p(\mathbf{x}_v \mid A)$ compared to the other categories of class-A of the same PNN. Note that $p(\mathbf{x}_v \mid A)$ is one of the factors responsible for the correct classification of a class-A point from the validation set. The parameter $\mathrm{Re}\, w_t^A(\mathbf{x}_v)$ is defined by the following equation:

$$\mathrm{Re}\, w_t^A(\mathbf{x}_v) = \frac{\prod_{d=1}^{D} \frac{1}{\sigma_t^A(d)} \exp\left(-\frac{(\mathbf{x}_v(d)-\mathbf{x}_t^A(d))^2}{2(\sigma_t^A(d))^2}\right)}{\max_{t' \in S_{tr}(A)} \prod_{d=1}^{D} \frac{1}{\sigma_{t'}^A(d)} \exp\left(-\frac{(\mathbf{x}_v(d)-\mathbf{x}_{t'}^A(d))^2}{2(\sigma_{t'}^A(d))^2}\right)} \quad (13)$$

The *Correct Selectivity* parameter, $CSel_t^A$, is calculated for all the class-A validation points $\mathbf{x}_v$ from the validation set S_v. In particular, the Correct Selectivity parameter, $CSel_t^A(\mathbf{x}_v)$, corresponding to a class-A validation point $\mathbf{x}_v$, is a normalized measure of how much category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ contributes to the value of $p(\mathbf{x}_v \mid A)$, compared to the other categories of class-A of the same PNN. Note that $p(\mathbf{x}_v \mid A)$ is one of the factors responsible for the correct classification of a class-A point from the validation set. The parameter $CSel_t^A(\mathbf{x}_v)$ is defined by the following equation:

$$CSel_t^A(\mathbf{x}_v) = \frac{\prod_{d=1}^{D} \frac{1}{\sigma_t^A(d)} \exp\left(-\frac{(\mathbf{x}_v(d)-\mathbf{x}_t^A(d))^2}{2(\sigma_t^A(d))^2}\right)}{\max_{t' \in S_{tr}(A)} \prod_{d=1}^{D} \frac{1}{\sigma_{t'}^A(d)} \exp\left(-\frac{(\mathbf{x}_v(d)-\mathbf{x}_{t'}^A(d))^2}{2(\sigma_{t'}^A(d))^2}\right)} \quad (14)$$

In (13) and (14), $S_{tr}(A)$ are the points in the training set S_{tr} that are class-A points.

Now that the Reward and Correct Selectivity parameters are defined for a category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ of a PNN, and a specific class-A validation point, it is worth defining a cumulative parameter for this category, referred to as the *Goodness Factor*. This parameter takes into consideration this category's Reward and Correct Selectivity parameter values for every class-A validation point. The Goodness Factor, of category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ is defined below.

$$GFac_t^A = 0.5 \frac{\sum_{v \in S_v(A,A)} \mathrm{Re}\, w_t^A(\mathbf{x}_v)}{\max_{t' \in S_{tr}(A)} \sum_{v \in S_v(A,A)} \mathrm{Re}\, w_{t'}^A(\mathbf{x}_v^A)} + 0.5 \frac{\sum_{v \in S_v(A)} CSel_t^A(\mathbf{x}_v)}{\max_{t' \in S_{tr}(A)} \sum_{v \in S_v(A)} CSel_{t'}^A(\mathbf{x}_v)} \quad (15)$$

In the above equation $S_v(A,A)$ is the set with indices from the validation set S_v that correspond to validation points of class label A that are correctly classified by the PNN in which the designated category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ belongs. Furthermore, $S_v(A)$ are the set of indices from the validation set S_v that correspond to class-A points.

The *Penalty* parameter, Pen_t^A, is calculated for all the class-$\overline{A}$ validation points $\mathbf{x}_v$ from the validation set S_v that are incorrectly classified by the PNN, to which the category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ belongs, as class-A points. In particular, the Penalty parameter, $Pen_t^A(\mathbf{x}_v)$, corresponding to a class-$\overline{A}$ validation point $\mathbf{x}_v$, incorrectly classified by the PNN, to which this category belongs, as a class-A point, is a normalized measure of how much this category contributes to the value of $p(\mathbf{x}_v \mid A)$, compared to the other class-A categories of the same PNN. Note that $p(\mathbf{x}_v \mid A)$ is one of the factors responsible for the incorrect classification of a class-$\overline{A}$ point from the validation set. The parameter $Pen_t^A(\mathbf{x}_v)$ is defined by the following equation:

$$Pen_t^A(\mathbf{x}_v) = \frac{\prod_{d=1}^{D} \frac{1}{\sigma_t^A(d)} \exp\left(-\frac{(\mathbf{x}_v(d)-\mathbf{x}_t^A(d))^2}{2(\sigma_t^A(d))^2}\right)}{\max_{t' \in S_{tr}(A)} \prod_{d=1}^{D} \frac{1}{\sigma_{t'}^A(d)} \exp\left(-\frac{(\mathbf{x}_v(d)-\mathbf{x}_{t'}^A(d))^2}{2(\sigma_{t'}^A(d))^2}\right)} \quad (16)$$

The *Erroneous Selectivity* parameter, $ESel_t^A$, is calculated for all the class-$\overline{A}$ validation points $\mathbf{x}_v$ from the validation set S_v. In particular, the Erroneous Selectivity parameter, $ESel_t^A(\mathbf{x}_v)$, corresponding to a class-$\overline{A}$ validation point $\mathbf{x}_v$, is a normalized measure of how much category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ contributes to the value of $p(\mathbf{x}_v \mid A)$, compared to the other class-A categories of the same PNN. Note that $p(\mathbf{x}_v \mid A)$ is one of the factors responsible for the incorrect classification of a class-$\overline{A}$ point from the validation set. The parameter $ESel_t^A(\mathbf{x}_v)$ is defined by the following equation:

$$ESel_t^A(\mathbf{x}_v) = \frac{\prod_{d=1}^{D} \frac{1}{\sigma_t^A(d)} \exp\left(-\frac{(\mathbf{x}_v(d) - \mathbf{x}_t^A(d))^2}{2(\sigma_t^A(d))^2}\right)}{\max_{t' \in S_{tr}(A)} \prod_{d=1}^{D} \frac{1}{\sigma_{t'}^A(d)} \exp\left(-\frac{(\mathbf{x}_v(d) - \mathbf{x}_{t'}^A(d))^2}{2(\sigma_{t'}^A(d))^2}\right)} \quad (17)$$

In (15) and (17) $S_{tr}(A)$ are the points in the training set S_{tr} that are class-A points.

Now that the Penalty and Erroneous Selectivity parameters are defined for a category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ of a PNN, and a specific class-$\overline{A}$ validation point, it is worth defining a cumulative parameter for this category, referred to as the *Badness Factor*. This parameter takes into consideration this category's Penalty and Erroneous Selectivity parameter values for every class-$\overline{A}$ validation point. The Badness Factor, of category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ is defined below.

$$BFac_t^A = 0.5 \frac{\sum_{v \in S_v(\overline{A}, A)} Pen_t^A(\mathbf{x}_v)}{\max_{t' \in S_{tr}(A)} \sum_{v \in S_v(\overline{A}, A)} Pen_{t'}^A(\mathbf{x}_v)} + 0.5 \frac{\sum_{v \in S_v(\overline{A})} ESel_t^A(\mathbf{x}_v)}{\max_{t' \in S_{tr}(A)} \sum_{v \in S_v(\overline{A})} ESel_{t'}^A(\mathbf{x}_v)} \quad (18)$$

In the above equation $S_v(\overline{A}, A)$ is the set with indices from the validation set S_v that correspond to validation points of class label $\overline{A}$ that are incorrectly classified by the PNN in which the designated category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ belongs, as class A points. Furthermore, $S_v(\overline{A})$ are the set of indices from the validation set S_v that correspond to class-$\overline{A}$ points.

Now we are in a position to define the confidence factor of a category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ in terms of the Goodness and Badness factors of a category. In particular, the confidence factor of a category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ is defined as follows:

$$CF_t^A = GF_t^A \cdot (1 - BF_t^A) \quad (19)$$

We can use this confidence factor to define the probability according to which a category in a PNN is pruned during the evolution. In particular, the probability of pruning the category $\mathbf{x}_t^A, \boldsymbol{\sigma}_t^A$ during the evolution of the PNNs is defined below.

$$PRune_t^A = 1 - \frac{Rank(CF_t^A)}{MaxRank_{t' \in PNN}(CF_{t'}^A)} \quad (20)$$

In the above equation the highest confidence value gets the highest rank and the lowest confidence value gets the lowest rank. Note that the above probability changes from generation to generation of the evolutionary process and depends on how important this category is for the good performance of the PNN in which it belongs. Hence, the user does not need, with the above approach, to arbitrarily define a probability of pruning a PNN category.

E. MO-EPNN Stopping Criterion

It is important to define a good stopping criterion for the MO-EPPN's evolution. This stopping criterion should be such that stopping MO-EPNN's evolution according to this criterion would not compromise the quality of the solutions. We chose a stopping criterion that relies on the measure of coverage, firstly introduced in Zietzler, et al., [18], where in comparing two families of solutions A, B we calculate the coverage of B by A using a metric denoted as $C(A, B)$. This metric is defined as follows:

$$C(A, B) = \frac{|\{b \in B : \exists\, a \in A \; a \geq b\}|}{|B|} \quad (21)$$

where in the above equation $|S|$ means the size of set S (i.e., number of elements in the set S) and $a \geq b$ implies that solution a dominates solution b. If we assume for a moment that A is the archive at generation g and B is the archive at generation $g+1$, then a $C(A, B)$ value of 1 means that the archive in the current generation is covered by the archive in the previous generation. Our stopping criterion is to terminate the PNN evolution, if and only if $C(A, B) = 1$ for 10 consecutive generations.

F. The MO-EPNN Algorithm

We used $N_{seed} = 10$ initial seeds. For each initial seed we generated eight populations of PNNs corresponding to each of schemes *A*, *C* and *D* and each of the five *B* schemes. Each such population had $Pop_{size} = 30$ members. The eight populations corresponding to the same initial seed are different because the schemes of choosing the spread parameters are different. The diversity of PNN solutions provided by this initial collection of PNNs is obvious from the following numbers that provide the mean and standard deviation of the Error on the validation set and the size of the initial population of PNNs: CinS (MeanError 7.08%, StdError 6.06%, MeanSize 504.95, StdSize 286.33), G4c_25 (MeanError 28.62%, StdError 3.91%, MeanSize 253.99, StdSize 143.17), G6c_15 (MeanError 19.65%, StdError 4.85%, MeanSize 257.04, StdSize 144.46), IRIS (MeanError 6.06%, StdError 2.77%, MeanSize 252.93, StdSize 143.17), SEG (MeanError 17.27%, StdError 8.38%, MeanSize 406.70, StdSize 229.03), WAVE (MeanError 32.24%, StdError 6.44%, MeanSize 505.45, StdSize 286.37), ABA (MeanError 44.98%, StdError 2.99%, MeanSize 253.51, StdSize 143.19), PAGE (MeanError 14.16%, StdError 7.51%, MeanSize 258.02, StdSize 145.17).

Each one of the populations corresponding to a particular seed followed the pseudo-code of Figure 2 in order to generate an archive of solutions at the completion of the evolution (final archive). The solutions of these eight final archives were combined to generate a *Better-Archive*; the *Better-Archive* contains the solutions of the final archives that are non-dominated. The *Better-Archives* of every seed (10 of them) are combined to generate a *Best-Archive*; the *Best-Archive* contains all the solutions of the *Better-Archives* that are non-dominated.

IV. Experiments and Results

A. Datasets

We have experimented with 8 datasets (see Table I), of which 3 are simulated databases and 5 are real databases. The simulated databases include 2 Gaussian databases: G4c-25 and G6c-15. These are, 2-dimensional databases with 4-classes and 6-classes, and 15% and 25% overlap, between the classes. The overlap in these Gaussian datasets implies that if the optimal (Bayes) classifier were to be used the classification error attained (optimal error) would be equal to the overlap percentage. The database denoted by CinS is the benchmark one circle in a square problem, 2-dimensional, two class classification problem. The probability of finding a data point within a circle or inside the square of the circle is equal to 1/2. The rest of the databases were obtained from the UCI repository (see [19]) and they include: Modified Iris (IRIS), Image Segmentation (SEG), Waveform (WAV), Abalone (ABA), and Page Blocks (PAGE). More details about these databases can be found there.

Each dataset is divided into a training set, a validation test, and a test set. The training set is used for the training of PNNs under consideration. The validation set is used to estimate the classification error during the evolutionary process, as explained in the previous section. Finally, the test set is used to assess the performance of the optimized networks created. The characteristics of the datasets used in our experiments are provided in Table I below.

Table I
Characteristics of Datasets used in MO-EPNN Experiments

Data Set	Train #	Val #	Test #	Atr #	Class #
CinS	1000	5000	5000	2	2
G4c_25	500	5000	5000	2	4
G6c_15	504	5004	5004	2	6
IRIS	500	4800	4800	2	2
SEG	800	810	700	19	7
WAVE	1000	2000	2000	21	3
ABA	500	2000	1677	7	3
PAGE	507	2176	2790	10	5

B. MO-EPNN Experiments; Comparison with Other Classifiers

We compared the performance of MO-EPNN (best) and MO-EPNN (small) with the performance of three other classifiers, CART [9], SVM [10], and the genetic PNN in Mao, et al., [14].

The MO-EPPN (best) is the PNN in the *Best-Archive* that produced the lowest error on the validation set. The MO-EPNN (small) is the PNN in the *Best-Archive* whose error rate is within 1% of the error rate of MO-EPNN (best) and has as many categories as the number of classes of the classification problem at hand; if such a PNN does not exist MO-EPNN (small) is the one with error rate within 1% of the MO-EPNN (best) error rate with the smallest number of categories. If there is no PNN with error rate within 1% of the error rate of MO-EPNN (best) then the MO-EPNN (small) is non-existent. The MO-EPPN (best) and MO-EPPN (small) performance (for the eight datasets) is shown in Table II; the PCC in Table II is percentage of correct classification on the test set. The reported size for the MO-EPNN (best) and MO-EPNN (small) of Table II corresponds to the number of categories of these PNNs.

To produce the SVM results of Table II we trained and validated the SVM classifier for a number of C and γ parameters. Note that C is the regularization parameter of the SVM classifier and γ is the width of the kernel used (in our case we used the popular RBF kernel). The results shown in Table II correspond to the SVM test performance of the SVM model that provided the highest PCC on the validation set; the SVM size reported in Table II corresponds to the number of support vectors of the SVM classifier. To produce the CART results we used the CART algorithm with the Gini split criterion and the 1-SE rule (see [9] for more details). The results shown in Table II correspond to the test performance of the CART model that the 1-SE rule produces; the size of this model corresponds to the number of intermediate nodes (not leaves) of the CART model. To produce Mao's results in Table II we implemented Mao's algorithm, discussed in [14].

Table II
Percentage of Correct Classification (PCC) and Size of the Classifier

Algorithm\Dataset	CinS PCC	CinS Size	G4c_25 PCC	G4c_25 Size
SVM	99.67	88	75.24	277
CART	97.57	28	73.5	**4**
Mao's	90.14	10	73.44	**4**
MO-EPNN (best)	97.90	84	74.72	5
MO-EPNN (small)	96.80	10	74.86	**4**

Algorithm\Dataset	G6c_15 PCC	G6c_15 Size	IRIS PCC	IRIS Size
SVM	84.99	504	95.04	79
CART	80.42	**6**	94.02	4
Mao's	85.35	**6**	93.63	**2**
MO-EPNN (best)	84.89	45	95.23	54
MO-EPNN (small)	84.83	**6**	94.52	**2**

Algorithm\Dataset	SEG PCC	SEG Size	WAVE PCC	WAVE Size
SVM	97.29	230	87.45	574
CART	93.43	17	75.2	14
Mao's	88.00	101	81.45	9
MO-EPNN (best)	89.57	64	84.40	11
MO-EPNN (small)	89.29	42	85.10	5

Algorithm\Dataset	ABA PCC	ABA Size	PAGE PCC	PAGE Size
SVM	61.66	337	95.3	150
CART	61.18	17	93.84	7
Mao's	57.96	11	85.66	171
MO-EPNN (best)	60.94	238	95.94	8
MO-EPNN (small)	60.17	5	96.24	7

The results are created by SVM, CART, Mao's algorithm, and MO-EPPN (best and small) for the CinS, G4c_25, G6c_15, IRIS, SEG, WAV, ABA, and PAGE datasets. The PCC shown is the performance of each classifier on the test set. The gray highlighted table entries correspond to the best PCC and smallest size attained by any of these classifiers on the respective dataset. If an entry corresponding to size is bold-faced this size is equal to the smallest possible size of a classifier that can be attained (equal to the number of classes of the classification problem)

C. Discussion of Results

A careful look at the results presented in Table II allows us to make a few useful observations regarding MO-EPNN's performance. **Observation 1:** MO-EPNN's (best or small) PCC is competitive compared to SVM since, quite often (in five out of the eight datasets), it produces a PCC nearly as good (within 1%), and occasionally slightly better, as the SVM PCC. Furthermore, MO-EPNN has size that is much smaller than SVM (quite often orders of magnitude smaller). **Observation 2:** MO-EPNN's performance is competitive compared to CART. In six datasets MO-EPNN's PCC is better than CART's, while in the remaining two datasets CART's PCC is better than MO-EPPN's. In two of the datasets (G6c_15, WAVE) MO-EPNN's PCC is significantly better than CART's, while CART's performance is significantly better than MO-EPNN's PCC in one dataset (SEG). In almost all the datasets (except SEG) MO-EPPN (small)'s size is either smaller or equal to the CART's size. **Observation 3:** MO-EPPN's PCC is competitive compared to Mao's PCC. In all the datasets except one (G6c_15) MO-EPPN's PCC is better than Mao's, and in some of them significantly better (e.g., CinS and PAGE). In all the datasets MO-EPNN (small)'s size is smaller than or equal to Mao's size and in a few occasions (SEG, PAGE) a large factor times smaller.

The above observations demonstrate that MO-EPPN compares well with other classifier models, such as SVM and CART, and Mao's algorithm ([14]).

V. DISCUSSION

Our intent in this paper was not only to design an effective, parameter-less, multi-objective evolutionary algorithm to optimize a population of PNN classifiers (MO-EPPN) but also to identify the principles associated with such a design. We single out these design principles below: **Design Principle 1:** An appropriate multi-objective evolutionary optimization approach is needed to select the PNNs that are reproduced from one generation to the next. In this paper we chose SPEA2 [17], but other evolutionary multi-objective optimization approaches could work as well; SPEA2 worked well for the evolution of ART classifiers (MO-GART). **Design Principle 2:** An initial population of diverse solutions for the classification model needs to be created, so that its subsequent evolution through selection and reproduction operators will lead to good final solutions. In order to produce this diverse initial population the user needs to know what provides the diversity of solutions for the classification model under consideration (ART, PNN, others), a relatively reasonable expectation for the user. **Design Principle 3:** Appropriate evolutionary operators need to be designed to evolve the initial population of solutions, such as the Prune operator proposed for the PNN in this paper. Most importantly, the Prune operator needs to assign appropriate credit to the genes of the chromosome (categories in a PNN classifier model or ART classifier model) so that good genes are retained while bad genes are pruned. The Prune operator was the most important operator in the evolution of the PNN in this paper and also of ART in [8]. Other operators might be needed, besides the Prune operator, such as an appropriate cross-over or mutation operator. Cross-over and mutation operators did not produce better MO-EPNN results, but cross-over and mutation were useful for the evolution of ART networks. **Design Principle 4:** The evolutionary operators need to be adaptive, meaning that they should not require the user to specify parameters that are hard to choose without costly experimentation. For example, the Prune operator in the evolution of the PNNs used the adaptive confidence factor of a category (calculated automatically) that determined the Prune probability. A similar confidence factor was defined in the evolution of ART networks in [8]. **Design Principle 5:** The multi-objective evolutionary algorithm needs to use an adaptive stopping criterion that determines the termination of the evolutionary process. In MO-EPNN this criterion was based on the measure of coverage, introduced by Zietzler [18]. A similar stopping criterion was used for MO-GART. **Design Principle 6:** Experimentation with the multi-objective evolutionary algorithm will determine what affects the

variability of the answers in the final archive. In order to design a parameter-less multi-objective evolutionary algorithm to optimize EBC models (such as ART, PNN, others) one has to eliminate this variability to the maximum possible extent. In the case of the evolution of the PNNs this was accomplished by considering multiple initialization schemes and multiple random initial seeds. In the case of the evolution of ARTs this was accomplished by considering multiple random initial seeds.

VI. SUMMARY AND CONCLUSIONS

We have designed an effective, parameter-less, multi-objective evolutionary algorithm that optimizes a population of PNN models. This algorithm was named MO-EPNN. The statement that MO-EPNN is parameter-less is justified because the user of this algorithm does not have to specify any algorithmic parameters, since all the pertinent parameters (such as Pop_{size}, N_{seed}, SS and the different initialization schemes) have already been predefined and work well for all datasets that we experimented with. MO-EPNN not only produces a family of PNN solutions for the classification problem at hand but it also produces solutions that compare favorably with other popular classification models, such as SVM and CART and other evolutionary PNN models, such as the one developed by Mao [14].

The contributions of the paper are many. In the design of the MO-EPNN, (a) we chose the multi-objective evolutionary algorithm that created future PNN populations from earlier PNN populations, (b) we chose novel reproduction operators (prune operator) that alter the members of the populations, (c) we defined an adaptive probability of pruning PNN categories for the prune operator, and (d) we identified what affects the variability of the final PNN populations produced by our MOEA. All these choices led to a successfully designed and competitive MO-EPNN. Furthermore, the similarity of the principles that led into the successful design of MO-EPNN and MO-GART point into a future research direction where these design principles can be illustrated as being universal principles for the optimization of any exemplar based classifier.

REFERENCES

[1] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multi-dimensional maps," *IEEE Transactions on Neural Networks*, Vol. 3, No. 3, pp. 698-713, 1992.

[2] G. Anagnostopoulos, "Novel approaches in adaptive resonance theory for machine learning," Ph.D. dissertation, University of Central Florida, Orlando, Jan, 2001.

[3] J. R. Williamson, "Gaussian ARTMAP: A neural network for fast incremental learning of noisy multi-dimensional maps," *Neural Networks*, Vol. 9, No. 5, pp. 881-897, 1996.

[4] J. Moody, C.J. Darken, "Fast learning in networks of locally tuned processing units," *Neural Computation*, Vol. 1, No. 2, pp. 281-294, 1989.

[5] D. F. Specht, "Probabilistic Neural Networks," *Neural Networks*, Vol. 3, No. 1, pp. 109-118, 1990.

[6] T. Cover, and P. Hart, "Nearest neighbor pattern classification," *Proceedings IEEE Transactions on Information Theory*, Vol. 13, No. 1, pp. 21-27, Jan, 1967.

[7] E. Parzen, "On estimation of probability density function and mode," *Annals of Mathematical Statistics*, Vol. 33, No. 3, pp. 1065-1076, 1962.

[8] A. Kaylani. M. Georgiopoulos, M. Mollaghasemi, G. C. Anagnostopoulos, "An adaptive multi-objective approach to evolving ART architectures," *IEEE Transactions on Neural Networks,* Vol. 21, No. 4, pp. 529-550, 2010.

[9] L. Breiman, J. Friedman, R. Olshen, C. Stone, *Classification and Regression Trees*, Wadsworth, 1984.

[10] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.

[11] H. Ishibuchi, K. Nozaki, N. Yamamoto, H. Tanaka, "Construction of fuzzy classification systems with rectangular fuzzy rules using genetic algorithms," *Fuzzy Sets and Systems*, Vol. 65, No. 2/3, pp. 237-253, 1994.

[12] H. Ishibuchi and Y. Nojima, "Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning," *International Journal of Approximate Reasoning*, vol. 44, No. 1, pp. 4–31, 2007.

[13] J. Gonzalez, I. Rojas, J. Ortega, H. Pomares, F. J. Fernandez, A. F. Diaz, "Multi-objective evolutionary, optimization of the size, shape, and position parameters of radial basis function networks for function approximation," IEEE Transactions on Neural Networks, Vol. 14, No. 6, pp. 1478-1495, November 2003.

[14] K. Z. Mao. K. C. Tan, and W. Ser, "Probabilistic neural network structure determination for pattern classification," IEEE Transactions on Neural Networks, Vol. 11, No. 4, pp. 1009-1016, 2000.

[15] C. A. Coello, "An updated survey of GA-based multiobjective optimization techniques", *ACM Computing Surveys*, Vol. 32, No. 2, pp. 109-143, 2000.

[16] C. A. Coello, "Evolutionary multi-objective optimization: A historical view of the field," *IEEE Computational Intelligence Magazine*, Vol. 1, No. 1, pp. 28-36, February 2006.

[17] Eckart Zitzler, Marco Laumanns, and Lothar Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," *Technical Report 103*, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

[18] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation*, Vol. 8, No. 2, pp. 173-195, 2000

[19] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: http://www.ics.uci.edu/»mlearn/MLRepository.html