Mutation-Based Genetic Neural Network

Article *in* IEEE Transactions on Neural Networks · June 2005

DOI: 10.1109/TNN.2005.844858 · Source: PubMed

3 authors:

Paulito Palmes
IBM Dublin Research Lab
18 PUBLICATIONS **379** CITATIONS
SEE PROFILE

Taichi Hayasaka
National Institute of Technology, Toyota College
17 PUBLICATIONS **193** CITATIONS
SEE PROFILE

Shiro Usui
EIIRIS Toyohashi Tech.
284 PUBLICATIONS **1,874** CITATIONS
SEE PROFILE

Mutation-Based Genetic Neural Network

Paulito P. Palmes, Taichi Hayasaka, and Shiro Usui, *Fellow, IEEE*

***Abstract*—Evolving gradient-learning artificial neural networks (ANNs) using an evolutionary algorithm (EA) is a popular approach to address the local optima and design problems of ANN. The typical approach is to combine the strength of backpropagation (BP) in weight learning and EA's capability of searching the architecture space. However, the BP's "gradient descent" approach requires a highly computer-intensive operation that relatively restricts the search coverage of EA by compelling it to use a small population size. To address this problem, we utilized mutation-based genetic neural network (MGNN) to replace BP by using the mutation strategy of local adaptation of evolutionary programming (EP) to effect weight learning. The MGNN's mutation enables the network to dynamically evolve its structure and adapt its weights at the same time. Moreover, MGNN's EP-based encoding scheme allows for a flexible and less restricted formulation of the fitness function and makes fitness computation fast and efficient. This makes it feasible to use larger population sizes and allows MGNN to have a relatively wide search coverage of the architecture space. MGNN implements a stopping criterion where overfitness occurrences are monitored through "sliding-windows" to avoid premature learning and overlearning. Statistical analysis of its performance to some well-known classification problems demonstrate its good generalization capability. It also reveals that locally adapting or scheduling the strategy parameters embedded in each individual network may provide a proper balance between the local and global searching capabilities of MGNN.**

***Index Terms*—Artificial neural networks (ANNs), evolutionary algorithm (EA), evolutionary programming (EP), evolutionary strategies (ESs), genetic algorithm (GA), hybrid algorithm (HA).**

I. Introduction

THE USE of evolutionary algorithms (EAs) [1]–[4] to aid in artificial neural networks (ANNs) learning [5]–[7] has been a popular approach to address the shortcomings of backpropagation (BP) [8]. The typical approach uses a population of gradient-learning ANN undergoing weight adaptation through BP training and structure evolution through EA [9]–[31]. At one extreme end are approaches that rely solely on EA for both of ANNs structure evolution and weight adaptation [8], [32], [33]. Based on these observations and in relation to the proposed algorithm, these current approaches can be classified into two major types: "noninvasive" which refers to the former approaches where EA selection is used but fitness evaluation requires BP or other gradient training; "invasive" which refers to the latter approaches where the system uses EA for ANN's weight and structure evolution. The proposed algorithm falls under the "invasive" approach.

Since BP has been widely studied [34] and many algorithms have been developed to improve its performance [8]–[10], [12], [35]–[40], the "noninvasive" evolution is the most popular. In this approach, the explicit separation of operations between weight adaptation by BP and structure evolution by EA often requires the development of a dual representation scheme. Because of this duality requirement [41], it is natural for this approach to adopt a GA-type evolution.

The "noninvasive" approach does not change aggressively the typical learning mechanisms of the individual network. The EA is only used to serve as a background process during evolution. Its successful performance still heavily relies on the proper initialization of BP parameters and the proper choice of BP implementation [8]. Individual network still undergoes gradient error minimization which is prone to the "local optima" problem [42], [43].

On the other hand, the "invasive" approach relies solely on an EA for the ANN evolution. Since weight adaptation and structure evolution are carried out directly using EA's perturbation function, efficient implementation of this approach avoids the mapping problem [8] by representing individuals at the species level [41]. By using direct representation and avoiding BP fitness evaluation, important EA operations are fast and makes feasible the use of a bigger population size for a more robust search coverage. One area that needs attention for this approach to be successful is the development of appropriate encoding scheme that supports strong causality and evolvability, allows fast and efficient fitness evaluation, and provides facilities for simultaneous structure evolution and weight adaptation. Another important area that needs to be addressed is the development of an appropriate stopping criterion to avoid premature learning and overlearning. This is the line of research undertaken by this paper.

One major issue of the "noninvasive" approach is the reliance to dual representations in representing BP-GA-aware ANNs. Finding a proper interpretation function to map ANN's genotype to phenotype is a big challenge due to the possible occurrence of *deceptive* mapping [3], [8]. Some of the major consequences of these deceptions include permutation problem or many-to-one mapping problem [44], one-to-many mapping problem [45], evolvability and causality problems [8].

Fig. 1 illustrates an example of a competing convention problem or many-to-one mapping problem. Genotypes are represented by a string of binary digits where weights are encoded in chunks of 4 b. Changes in the hidden layer due to crossover can produce two different genotypes with topologically similar ANN structure. Consequently, fitness evaluation in both structures produces the same output. This makes the

Manuscript received May 27, 2003; revised June 11, 2004.

P. P. Palmes and S. Usui are with the Laboratory for Neuroinformatics, RIKEN Brain Science Institute, Wako City, Saitama 351-0198, Japan (e-mail: ppalmes@brain.riken.jp; usuishiro@riken.jp).

T. Hayasaka is with the Department of Information and Computer Engineering, Toyota National College of Technology, Toyota, Aichi 471-8525, Japan (e-mail: hayasaka@toyota-ct.ac.jp).

Digital Object Identifier 10.1109/TNN.2005.844858

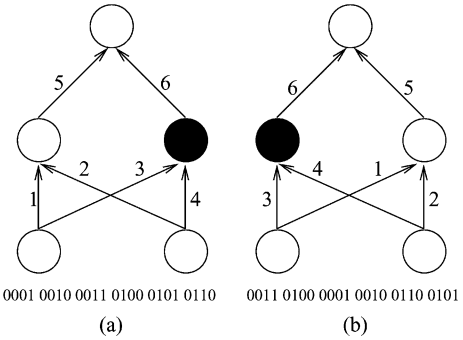

Fig. 1. Deceptive mapping.

evolution process inefficient and reduces population diversity which may lead to premature convergence.

In general, issues in causality [46], evolvability, deceptiveness [3], epistasis variance [47], ruggedness [48] are consequences of this genotype-phenotype mapping. The intuitive notion of these issues is based on the principle that evolution and adaptations are possible if the improvement can be done in cumulative way or stepwise fashion ("building-block hypothesis"). It is important that the mapping will not result into redundant phenotypes and inefficient fitness function evaluation. Also, the mapping must make sure that small changes in the genotype must have corresponding small changes to its phenotype, or else, evolution becomes inefficient and hard to control (evolvability and causality principles).

In the "noninvasive" approach, the high sensitivity of BP to the initial setup of its parameters causes a noisy evaluation of its fitness function which makes the entire process unreliable [45]. Another problem is that BP is an iterative approach with no well-defined criterion to stop training. Since individuals undergo BP training for every generation to determine their fitness, the evolution compounds the inefficiency of gradient training. Due to this inefficiency, typical implementation prefers small population size to have faster convergence at the expense of having poor coverage of the search space. Hence, it will often miss the global solution and converges to local solution.

On the other hand, typical fitness evaluation in the "invasive" approach utilizes a one-way feedforward computation which is many times faster than BP. Consequently, typical implementation prefers large population size because it has no significant impact in the overall CPU-time execution compared with that of the BP implementation. Since "invasive" approach scales better than its counterpart, it has better potential to find the global solution.

II. Related Work

Until now, relatively few researches have been done in "invasive" evolution to assess its effectiveness especially in the real-world domain. The work of McDonnell and Waagen [49]–[51] used the "invasive" approach to evolve the weight and connectivity matrices of ANN. However, they have only examined the effectiveness of their approach to a limited set of artificial problems such as binary mapping problem and 3-b parity problem. Fogel [32] used "invasive" evolution of three-layered feedforward ANN in order to develop good strategies for the tic-tac-toe problem. Common to these approaches is the use of *Gaussian* perturbation in the mutation operation to bring about changes in structure or weights of ANN

$$w = w + \mathbf{N}(0, \alpha\epsilon(\eta)) \quad \forall w \in \eta \tag{1}$$

where $\mathbf{N}(0, \alpha\epsilon(\eta))$ is the Gaussian perturbation with mean 0 and standard deviation $\alpha\epsilon(\eta)$, w is a weight, and $\epsilon(\eta)$ is an error function [e.g., mean-squared error (MSE)] which is scaled by the user-defined constant α. Selection of weights for mutation is random with probability based on certain criterion such as the variance of incident nodes [50] or using equal probability [32].

Among the "invasive" approaches, GNARL algorithm [8] shares the most number of features employed by mutation-based genetic neural network (MGNN) such as: structural and weight learning by mutation; sparse connections instead of uniform and symmetric topologies; hidden nodes varies from 0 to a user-specified limit; and real-valued weights. They differ, however, in the implementation of strategies for selection, encoding, mutation, fitness function formulation, and stopping criteria.

GNARL's selection strategy chooses the top 50% to become parents of the next generation based on a user-specified fitness function and discards the rest of the population. Reproduction is carried out using two types of mutation: parametric mutation and structural mutation. Parametric mutation is carried out by perturbing ANN's weights using Gaussian noise with the severity of mutation controlled by the annealing temperature T [52]

$$T(\eta) = 1 - \frac{\text{fitness}(\eta)}{\text{fitness}_{\text{best}}} \tag{2}$$

$$\hat{T}(\eta) = \mathbf{U}(0,1)T(\eta) \tag{3}$$

$$w = w + \mathbf{N}(0, \alpha\hat{T}(\eta)) \quad \forall w \in \eta \tag{4}$$

where $\mathbf{U}$ and $\mathbf{N}$ are *uniform* and *Gaussian* random variables, respectively. The preceding equations imply that individuals with inferior fitness values are mutated severely to improve its fitness while those with fitness values close to the best are mutated slightly. The middle equation lessens the frequency of occurrence of large mutations to avoid drastic changes that may have harmful effect on the offspring's ability to perform better than its parent.

Structural mutation involves addition or deletion of nodes or links. Node and links to be mutated are selected uniformly with the number of units to be modified determined by

$$\Delta_{\min} + \lfloor \mathbf{U}[0,1]\hat{T}(\eta)(\Delta_{\max} - \Delta_{\min}) \rfloor \tag{5}$$

where $(\Delta_{\max} - \Delta_{\min})$ is a user-defined interval.

GNARL examines three measures of fitness in its implementation, namely: sum of squared errors; sum of absolute errors; and sum of exponential absolute errors

$$\sum_i (y_i - \text{Out}(\eta, x_i))^2 \tag{6}$$

$$\sum_i |y_i - \text{Out}(\eta, x_i)| \tag{7}$$

$$\sum_i e^{y_i - \text{Out}(\eta, x_i)}. \tag{8}$$

Similar to MGNN, GNARL algorithm does not require any restriction in the fitness function formulation. In a supervised learning where a target vector is required, a measure of fitness similar to the previous equations can be formulated to support this requirement. Since both do not use any gradient information, changes in the formulation of their fitness function have no significant bearing on the mechanics and implementation of both algorithms.

One prominent approach called EPNet by Yao and Liu [45] is an example of a transition from a purely "noninvasive" strategy to "invasive" strategy. They recognized the shortcomings of the standard BP in ANN evolution and addressed it by incorporating the annealing strategy [52] to avoid local optima trap and minimize the noise in fitness evaluation. Also, due to the mapping problems of dual representation, they adopted the mutation and representation strategy of EP. However, EPNet still suffered from the scalability problem and minimized only the effect of noise in the fitness function evaluation due to their continued reliance to BP during mutation. In their follow-up study [53], they combined the solutions of EPNET population (ensembles) and produced better solutions compared to any of its isolated networks.

Several papers in evolutionary neural networks have demonstrated that utilizing population information provided better accuracy in classification instead of using the best network found by the evolution process. CNNE [54] is a recent approach which uses a constructive algorithm to evolve cooperative neural network ensembles. Instead of the typical way of evolving the best network configuration to solve a particular problem, CNNE relies on the contribution of individuals in the population. Hence, it uses incremental learning based on negative correlation to maintain diversity among the different networks in the population. This ensures that the redundancy in exploring the same region of the solution space is minimized which enables different networks to learn different aspects of the training data resulting to a superior solution of the ensemble. This model, however, uses "noninvasive" evolution which relies on the proper implementation of the BP algorithm. CNNE shares many similarities with the EPNET in terms of its constructive approach and may suffer from the "structural hill-climbing" problem [8] although this is minimized due to the utilization of ensembles.

On the other hand, instead of combining solutions of a neural network ensemble, COVNET [55] treats each neural network as a combination of subnetworks coevolving in cooperation with other subnetworks in other subpopulations. COVNET is an example of an "invasive" evolutionary model which uses direct encoding and does not utilize the crossover operation. Also, COVNET has less restriction in its structure and weights evolution and uses one-way feedforward computation for fitness evaluation. COVNET's evolutionary techniques such as selection, addition/deletion of nodes are similar to EPNET while the use of *Gaussian* perturbation during mutation is similar to MGNN, GNARL, and other EP models. COVNET has been shown to perform well in classification problems.

The good performance of COVNET, however, can be attributed to the simulated annealing it employs during mutation. Simulated annealing is a time-consuming process and its performance is greatly influenced by the choice of the appropriate temperature schedule. It will be interesting to determine how significant is the contribution of the coevolutionary model without the presence of simulated annealing but this issue is not addressed in the paper. The improvement it gains in terms of the error rate may not be enough to justify the computational requirement to carry-out this type of evolution. Besides, simulated annealing itself, if properly implemented, guarantees a global optimum solution without the need of coevolution.

The last two approaches are becoming popular to improve the performance of evolutionary models. Both are based on the principles that the information stored by the population is superior than the information learned by the best network. Both approaches, however, adds more complexity to the standard model. Due to the added complexity, they have to deal with several issues such as: maintaining interdependence among subnetworks, maintaining diversity, and credit assignment.

III. MGNN ENCODING

The application of the Darwinian principles in algorithm development can be traced back in the year 1960s. Earliest implementations can be classified into three major approaches, namely: genetic algorithm (GA) [1], [3]; evolutionary strategies (ESs) [56], [57]; and evolutionary programming (EP) [2], [58].

One popular EA approach to solve optimization problems is the GA. Holland's Schema Theorem formalized GA and provided the theoretical underpinning of the mechanisms behind it. One attractive feature of GA evolution is its support for the generic implementation of its major operations such as crossover, mutation, selection, and replacement. This is achieved by using a dual representation where evolutionary operations are done at the genotype level while fitness evaluation is carried out at the phenotype level.

On the other hand, earliest implementations of EP and ES relied on Gaussian mutation alone as the main source of perturbation. Instead of binary representation, both approaches were optimized to handle problems involving real-valued parameters. Notably, EP emphasizes the preservation of the behavioral links between parent and child. To carry out this objective, EP uses direct representation by representing individuals at the species level. Moreover, it avoids using crossover operation to avoid the possibility of destroying the behavioral links from the parent to the child. The recent trend among practitioners [40], [58]–[61] is to combine these three different approaches to address more complicated problems in AI and other fields.

To carry out the EA operations in ANN, it is important to have a proper encoding scheme [40]. Too complicated encoding scheme provides high flexibility in problem representation but may reduce EA's efficiency due to complicated operations. On the other hand, too simple representation may suffer from slow or premature convergence. This requires a careful selection of an encoding scheme such that EA operations are not compromised but still provides enough flexibility to support dynamic weight adaptation and evolution of structures.

Fig. 2 shows a typical example of a MGNN structure [Fig. 2(b)] and its equivalent ANN structure [Fig. 2(a)]. Individuals are represented by vectors and matrices of real numbers. Connection weights from the input layer to the hidden layer

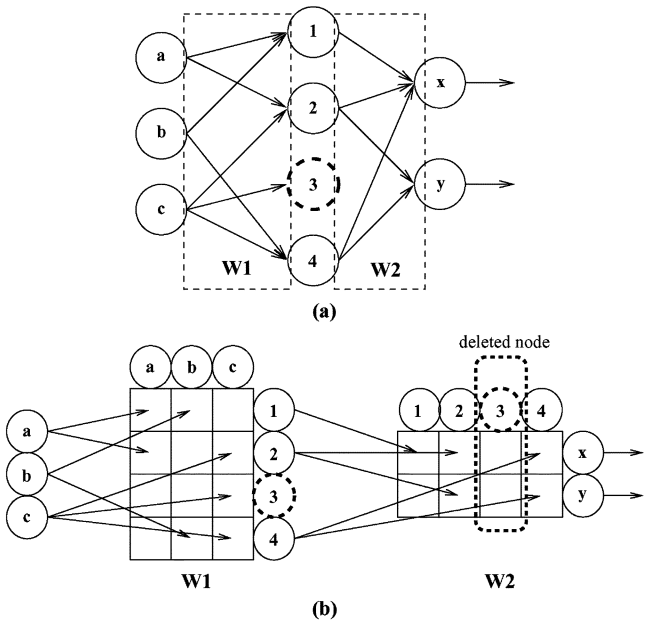

Fig. 2. MGNN encoding. (a) ANN. (b) MGNN.

are encoded in the weight matrix $W1$ while the weight matrix $W2$ contains the connection weights from the hidden-layer to the output-layer. These vectors and matrices are subjected to random perturbations during mutation to improve the ANN fitness.

One nice feature of this representation is its implicit support to structure evolution and weight adaptation. To illustrate, any column in $W2$ with values all set to zero represents a deleted or unutilized node in the hidden layer. For example, column 3 in $W2$ [Fig. 2(b)] indicates that there is no connection between node 3 [Fig. 2(a)] in the hidden layer to the output layer of ANN. This simple representation of ANN enables MGNN's mutation of $W1$ and $W2$ to dynamically change the structure and weights of the network.

Dynamic structure changes and local adaptation of weights through mutation are implemented using (9)–(11)

$$\delta = \rho(\sigma) \tag{9}$$

$$m' = m + \rho(\delta) \tag{10}$$

$$w'_{ij} = w_{ij} + \rho(m') \tag{11}$$

where

$$\begin{aligned} \sigma &\rightarrow \text{SSP (step size parameter)} \\ \delta &\rightarrow \text{mutation strength intensity} \\ \rho &\rightarrow \text{perturbation function} \\ m &\rightarrow \text{adapted strategy parameter.} \end{aligned}$$

The maximum strength of mutation is controlled by the step size parameter (SSP) σ while the level (δ) of mutation strength is dynamically computed during evolution. Evolution is carried out by starting a population of networks with zero weights. Hence, the only way for any network to evolve is by mutation. Assuming that the distribution of ρ is symmetric, deletion of connections/nodes is implicitly done when the result of operation in (11) is equal or close to zero.

Gaussian perturbation with mean $\mu = 0$ and standard deviation σ is a symmetric distribution where most of the perturbations are concentrated in the neighborhood of zero. MGNN uses this perturbation together with a small mutation probability to support strong causality and evolvability.

IV. MGNN Fitness Function

To achieve a proper balance in ANN's network complexity and generalization capability, MGNN's fitness function (Q_{fit}) (15) considers three important criteria to be minimized, namely: classification accuracy (Q_{acc}); training error (Q_{nmse}); and network complexity (Q_{comp})

$$Q_{\text{fit}} = \alpha * Q_{\text{acc}} + \beta * Q_{\text{nmse}} + \gamma * Q_{\text{comp}} \tag{12}$$

$$Q_{\text{acc}} = 100 * \left(1 - \frac{\text{correct}}{\text{total}}\right) \tag{13}$$

$$Q_{\text{nmse}} = \frac{100}{NP} * \sum_{j=1}^{P} \sum_{i=1}^{N} (T_i - O_i)^2 \tag{14}$$

$$Q_{\text{comp}} = \frac{c}{c_{\text{tot}}} \tag{15}$$

where: Q_{acc} is the percentage error in classification; Q_{nmse} is the percentage of normalized mean-squared error (NMSE); Q_{comp} is the complexity measure in terms of the ratio between the active connections c and the total number of possible connections c_{tot}; N; and P are the total number of input and training patterns, respectively; T and O are the target and network output, respectively. Since MGNN is based on a three-layered feedforward architecture, the value of c_{tot} is based on the size of its input (*in*), output (*out*), and the user-defined maximum number of hidden units (hid): $c_{\text{tot}} = \text{in} \times \text{hid} + \text{hid} \times \text{out}$.

The user-defined constants α, β, and γ are set to small values ranging between 0 and 1. They are used to control the strength of influence of their respective factors in the overall fitness measure. For example, implementations favoring accuracy over training error and complexity may set $\alpha = 1$, $\beta = 0.70$, and $\gamma = 0.30$. This particular assignment implies that for networks that have similar accuracies, ties are resolved by favoring individuals with smaller training error, and followed by those with minimal complexity. Simulations done in this paper use these assignments based on the results of preliminary experiments.

V. MGNN Algorithm

Any individual net_i [Fig. 2(a)] in MGNN is represented based on a general model described in the following:

$$\text{net}_i = \{W1, W2, \theta_{w1}, \theta_{w2}, \rho(\text{pr}, m, \sigma)\}$$

where pr is the probability of mutation and the other variables follow similar definitions from the previous discussion.

Algorithm 1 summarizes MGNN training algorithm. Initially (at iteration $t = 0$), MGNN generates a population $P(t)$ of μ ANNs with all the vectors and matrices initialized to zero

$$P(t) = \left\{\text{net}_1^t, \ldots, \text{net}_\mu^t\right\}.$$

Next, each net_i undergoes transformation through the mutation operator ρ (9)–(11) operating on the two threshold vectors

$(\theta_{w1}, \theta_{w2})$ and two connection weight matrices ($W1$ and $W2$). Then, MGNN evaluates each net_i performance using the fitness function (12). A new population $P(t+1)$ is formed by selecting individuals with better fitness from $P(t)$. From this new population, the processes of mutation, fitness evaluation, and selection are repeated until the stopping criterion is satisfied or the maximum number of generations is reached.

The output value O_i in the second term (Q_{nmse}) of the fitness function follows the typical feed-forward computation that uses sigmoidal activation function (f_i) and threshold values (θ_i) of ANN

$$O_i = f_i\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right). \tag{16}$$

MGNN selection policy is elitist wherein it retains the two best parents and replaces the rests with the new generation of offsprings. Elitism during training avoids the possibility of loosing the best traits in a particular generation. MGNN variants use two types of selection policies, namely: roulette-wheel selection and rank-based selection [3].

The stopping criterion uses the validation data to measure overfitness occurrences which help in the decision to stop training. The stopping criterion also monitors and keeps track the network with the best validation performance. The specific implementation to attain this objective is described by Algorithm 4 and discussed in the next section.

MGNN implements two types of mutation, namely: *stochastic* mutation (GA-inspired) and *scheduled* stochastic mutation (EP-inspired). Algorithm 2 describes the former while Algorithm 3 describes the latter. In the stochastic implementation, each weight in matrices $W1$, $W2$ and vectors θ_{w1}, θ_{w2} has the same 0.01 probability of perturbation. On the other hand, the *scheduled* implementation assigns higher probabilities to those with low fitness and lower probabilities to those with high fitness within the range of 0.01–0.05 using their fitness rank score. This is a similar idea to the annealing implementation of GNARL except that the assignment of probability in MGNN is purely deterministic. It works under the principle that individuals located away from the best solution need drastic changes to improve their fitness than those located near the optimal solution [8], [52].

Data: $Parents = \{net_1, \ldots, net_{popsize}\}$; $nextParents$; $windowSize$; $fittestNet$; $bestNet$
Result: $bestNet$
begin
1 $Parents.initialize()$;
2 $Parents.mutate()$;
3 **for** $generation \leftarrow 0$ ***to*** $maxGeneration$ **do**
4 $\forall net \in Parents$, $net.computeFitness()$;
5 **if** $fmod(generation, windowSize) == 0$ **then**
/*time to check if the current $fittestNet$ can replace the current $bestNet$ and check if overfitness is apparent (see Algo 4) */
6 **if** $stop(fittestNet, bestNet)$ **then** break ;
7 $nextParents \leftarrow Select(Parents)$;
8 $swap(Parents, nextParents)$;
9 $Parents.mutate()$;
10 **return** *bestNet*;
end

Algorithm 1. Training algorithm.

Data: $Parents = \{net_1, \ldots, net_{popsize}\}$; SSP;
Result: mutated $Parents$
begin
1 **for** *each network* $net \in Parents, net \neq fittestNet$ **do**
/*compute mutation strength to be adapted */
2 $net.mutationSSP \leftarrow gaussProb(0, SSP)$;
3 $net.mutation$ += $gaussProb(0, net.mutationSSP)$;
/*mutate weight matrices */
4 **for** *each weight matrix* $W \in net$ **do**
5 **for** $i \leftarrow 1$ ***to*** $rowSize$ *of* $net.W$ **do**
6 **for** $j \leftarrow 1$ ***to*** $columnSize$ *of* $net.W$ **do**
7 **if** $uniformRandom(0,1) \leq net.mutationProb$ **then**
8 $net.W[i][j]$ += $gaussProb(0, net.mutation)$;
/*mutate threshold vectors */
9 **for** *each threshold vector* $T \in net$ **do**
10 **for** $i \leftarrow 1$ ***to*** *size of* $net.T$ **do**
11 **if** $uniformRandom(0,1) \leq net.mutationProb$ **then**
12 $net.T[i]$ += $gaussProb(0, net.mutation)$;
end

Algorithm 2. Stochastic mutation policy.

VI. Stopping Criterion

The stopping criterion (Algorithm 4) is based on the observation that a good training performance does not necessarily imply a good validation performance. Allowing the system to continue more training than what is necessary often results into a poor generalization performance. At the early training cycle, however, validation performance is erratic and basing the decision to stop training at the onset of overfitness may lead to premature stopping.

One way to address this problem is by using interval sampling or "sliding-window." At the end of every ten-generation training interval (*window*), the stopping algorithm measures the presence of overfitness in the fittest network (*fittestNet*) using [62]

$$\text{over fitting} = 100 * \left(\frac{\text{fittest}Val + 1}{\text{best}Val_j + 1} - 1\right). \tag{17}$$

Also, MGNN measures its validation performance (*fittestVal*) and compares it to the validation performance of the succeeding fittest networks using

$$\text{best}Val_j = \text{Min}(\forall(\text{best}Val_i)) \quad i = 1, 2, \ldots, j. \tag{18}$$

Data: $Parents = \{net_1, \ldots, net_{popsize}\}$;
Result: mutated $Parents$
begin
  /*mutation probability ranges from 0.01 to 0.05 */
1 $start \leftarrow 0.01$;
2 $end \leftarrow 0.05$;
3 $step \leftarrow (end - start)/populationSize$;
  /*sort individuals ascendingly based on fitness */
4 $Parents.sort()$;
  /*assign mutation probability ($mutationProb$) to each network based on sort order */
5 **for** *each network* $net \in Parents, net \neq fittestNet$ **do**
6 $net.mutationProb \leftarrow start$;
   $start \leftarrow start + step$;
  /*now call the real mutation algo (see Algo 2) */
7 $Parents.mutate()$;
end

Algorithm 3. Scheduled mutation policy.

Data: current $fittestNet$; $bestNet$; $bestVal$; $maxOverFitCount$; $overFitCount$
Result: new $bestNet$ and $bestVal$. Returns "TRUE" if overfitness is apparent or "FALSE", otherwise.
begin
  /*compute fittestNet validation fitness. */
1 $fittestVal \leftarrow fittestNet.validate()$;
  /*measure overfitness */
2 $overfitting \leftarrow 100 * \left(\frac{fittestVal + 1}{bestVal + 1} - 1\right)$;
3 **if** $overfitting \geq 0$ **then**
    /*there is overfitness or no change in fitness. update counter */
4 $overFitCount$++ ;
5 **if** $overFitCount \geq maxOverFitCount$ **then**
      /*overfitness is apparent. stop training */
6 **return** *true* ;
   else
    /*found a better solution. update $bestNet$ and $bestNetValFit$ */
7 $bestNet \leftarrow fittestNet$;
8 $bestVal \leftarrow fittestVal$;
    /*consecutive overfitness is blocked. reset counter. */
9 $overFitCount \leftarrow 0$;
    /*continue training to get better solution */
10 **return** *false* ;
end

Algorithm 4. Stopping criterion.

If there is overfitness in 100 consecutive windows (1000 generations), the stopping criterion signals a flag to stop training. Otherwise, the counter for overfitness is reinitialized to zero, *bestNet* is updated to the network with the *bestVal*, and the evolution process continues. This implies that the real solution is the *bestNet* found 1000 generations before stopping. These extra generations are needed to ensure that the training stops only if there is apparent overfitness.

To illustrate, Fig. 3 shows the trend in the validation error (Q_{acc}) and the training error (Q_{fit}) of the fittest network per

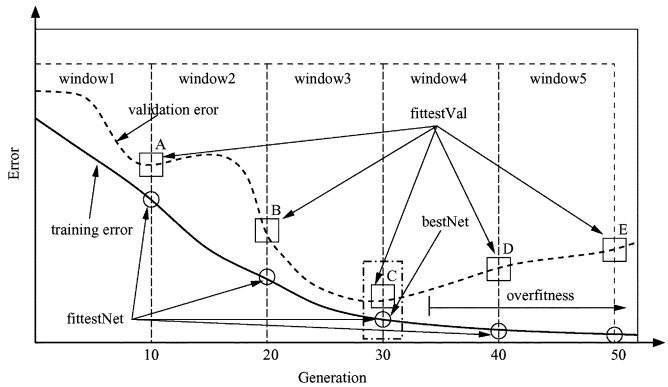

Fig. 3. Sliding window.

TABLE I
MGNN VARIANTS AND FEATURES

Features	MGNN Variants		
	roul	rank	ep
selection policy*	fitness	rank	rank
mutation policy*	stochastic	stochastic	scheduled
mutation probability(pr)*	0.01	0.01	(0.01,0.05)
mutation SSP(σ)*	SSP5, SSP100, SSP200		
mutation type(ρ)	gaussian		
replacement policy	elitist (retains 2 parents)		
classification encoding	1-of-m classes using 1,0 output		
classification method	accept only if $max(\lvert T_i - O_i \rvert) \leq 0.3$		
sliding window interval	every 10 generations		
max no. of windows	100 (1000 generations)		
no. of input/output	problem-specific		
max hidden units	10		
population size	100		
max generation	5000		
no. of trials (replications)	50		
training fitness measure	Q_{fit} (eqn. 12)		
Q_{fit} constants	$\alpha = 1, \beta = 0.70, \gamma = 0.30$		
validation/testing measure	Q_{acc} (eqn. 13)		

generation. Due to the elitist selection/replacement policy during training, the trend in training error is nonincreasing. On the other hand, validation performance fluctuates because generalization is difficult especially at the early stages of learning. The boxes indicate the locations of the validation performances of the fittest networks (indicated by circles) in each window interval. The stopping criterion compares the trend of these boxes in detecting overfitness while at the same time stores the network with the lowest validation error (*bestNet*).

From box A to box B to box C, the validation error is decreasing which implies that there is no overfitness. However, the trends from C to D and from C to E indicate that overfitness happens twice with respect to C (current *bestNet*). If this trend continues for 100 consecutive windows, the stopping criterion flags a signal to stop and returns the network with the best validation performance which is found in box C. If MGNN encounters another solution better than C before reaching the maximum number of windows, the stopping criterion updates *bestNet* and reinitializes the frequency counter to zero. This

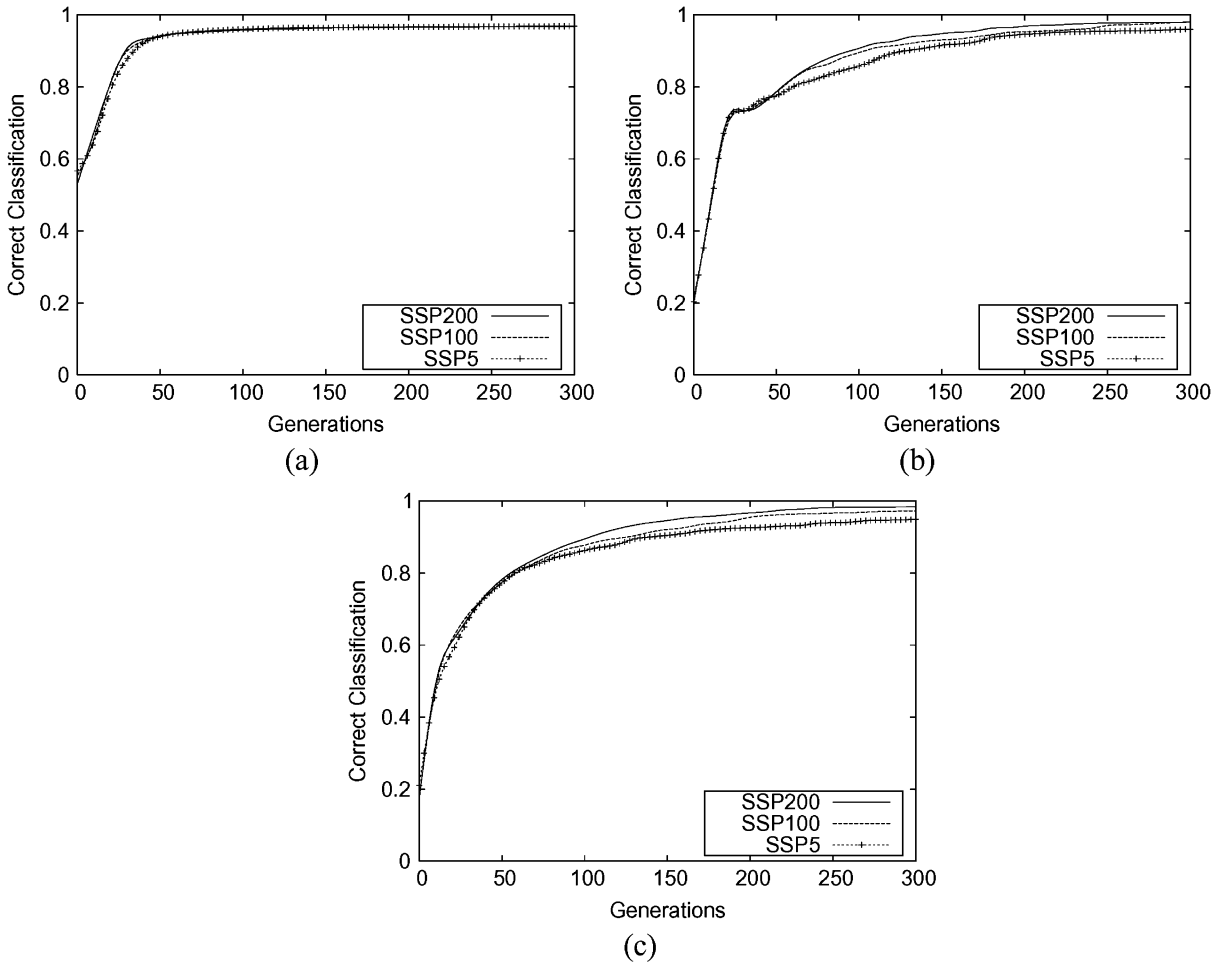

Fig. 4. Training performance of the different SSPs in MGNN-ep grouped by classification problem. (a) Cancer1. (b) Iris. (c) Wine.

cycle between training and stopping criterion evaluation continues until reaching either the maximum window or the maximum generation. The last *bestNet* before the training stops in Algorithm 4 is the optimal solution found by MGNN.

VII. Experiments and Results

Preliminary experiments showed that mutation probability of 0.01 significantly outperformed mutation probability of 0.05. In addition, these experiments indicated the superior performance of using *Gaussian* perturbation over *uniform* perturbation. Consequently, all simulations in this paper used *Gaussian* mutation with the mutation probability fixed at 0.01, except for the MGNN-ep that utilized "scheduled" mutation probability.

To evaluate the performance of MGNN, several experiments were conducted using some well-known classification problems from the database found in the UCI repository [63]: *iris* classification data, *wine* recognition data, and Wisconsin *breast-cancer* classification data. MGNN used the following factors for multivariate analysis:

- mutation step size (SSP5 versus SSP100 versus SSP200);
- selection and mutation strategy (ep versus rank versus roul).

The effects of these factors to the performance of MGNN were measured using the following dependent variables:

- percentage of wrong classification (ClassError);
- number of connection weights (connections);
- number of generations (generations).

The datasets were partitioned based on the suggestions of Prechelt [62]. While other implementations use a combination of training and validation data during the training phase, MGNN variants strictly use separate datasets for each phase and do not allow intersection. The data were divided into 50% training, 25% validation, and 25% testing using SRS (simple random sampling) except for the cancer problem. The datasets in the cancer problem were copied from the sampling in the paper of Prechelt. This allows statistically valid comparisons of MGNN with other algorithms that use the same datasets. Unfortunately, these standard sets are limited in scope and do not include the wine and iris problems.

This study used multivariate analysis of variance (MANOVA) and Duncan's multiple range test (DMRT) for the post hoc tests. Table I lists the important implementation-specific features of the three variants. Their names are based from the main feature of each approach. MGNN-roul uses roulette-wheel selection while MGNN-rank uses rank-based selection [3]. On the other hand, MGNN-ep differs from MGNN-rank by using EP-based *scheduled* mutation policy. In the object-oriented paradigm, MGNN-roul serves as the base class with MGNN-rank inheriting all of the former's features except the selection policy. For

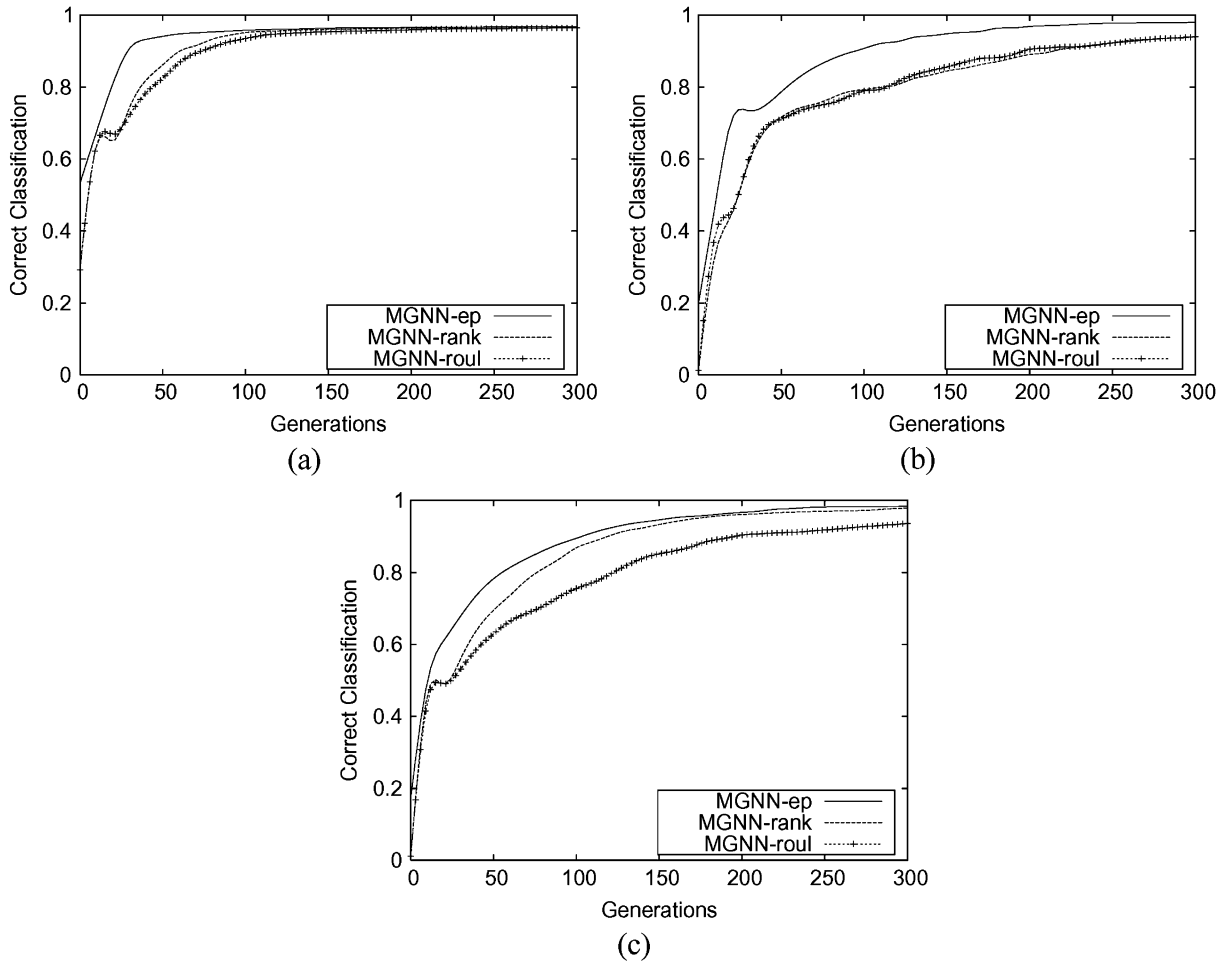

Fig. 5. Training performances of the different MGNN variants using SSP200 grouped by classification problem. (a) Cancer1. (b) Iris. (c) Wine.

MGNN-ep, it inherits all the features of MGNN-rank except the mutation policy.

The sizes for the input and output units are problem-specific while the maximum number of hidden units is a user-defined parameter. The classification method uses 1-of-m encoding for m classes using output values of either 1 or 0. One generation in MGNN is equivalent to a single presentation of the entire training data to each member of the population. For example, using a population size of 100 implies that one generation is roughly equivalent to 100 epochs without error BP. All variants use a strong restriction in the classification policy by considering correct classification only if the maximum absolute error is less than or equal to 0.3 (i.e., $\max(|T_i - O_i|) \leq 0.3$). Majority of these features can be tweaked to improve MGNN performance. In this paper, features with asterisks $(*)$ were setup for statistical analysis to better understand their influences.

A. MGNN Training Performance

Fig. 4 demonstrates the superior performance of SSP200 over SSP100 and SSP5 in MGNN-ep. This trend is fairly consistent to all the three problems. The same trend can be observed with the other two variants in the preliminary experiment but not shown here. Since the performances of the three variants are optimal in the cancer problem [Fig. 5(a)], the discrepancy in performance among SSPs [Fig. 4(a)] is not as apparent as the other two problems [Fig. 4(b) and (c)].

Fig. 5 shows the training performances of MGNN variants based on the percentage of correct classification $(1 - Q_{\mathrm{acc}})$. These results were averaged over 50 replications using a population size of 100. These tests used SSP200 since it provided the optimal performance compared to the other two SSPs (Fig. 4). It is apparent that MGNN-ep followed by MGNN-rank classifies better than MGNN-roul during the training phase.

Fig. 6 shows the optimal performances of the different variants using SSP200. In just less than 150 generations, MGNN-ep was able to correctly classify more than 90% of the training data in all problems. In particular, it correctly classified more than 90% of the cancer data in less than 50 generations. Both MGNN-rank and MGNN-roul took less than 300 generations to classify 90% of data in all problems.

B. MGNN Testing Performance

One of the most important criteria to determine the effectiveness of ANN learning is its generalization capability when confronted with a completely new set of data. To aid in the analysis of the generalization performance, DMRT tables are presented to highlight significant differences in the performance of the different variants. Mean values of factors in different subsets

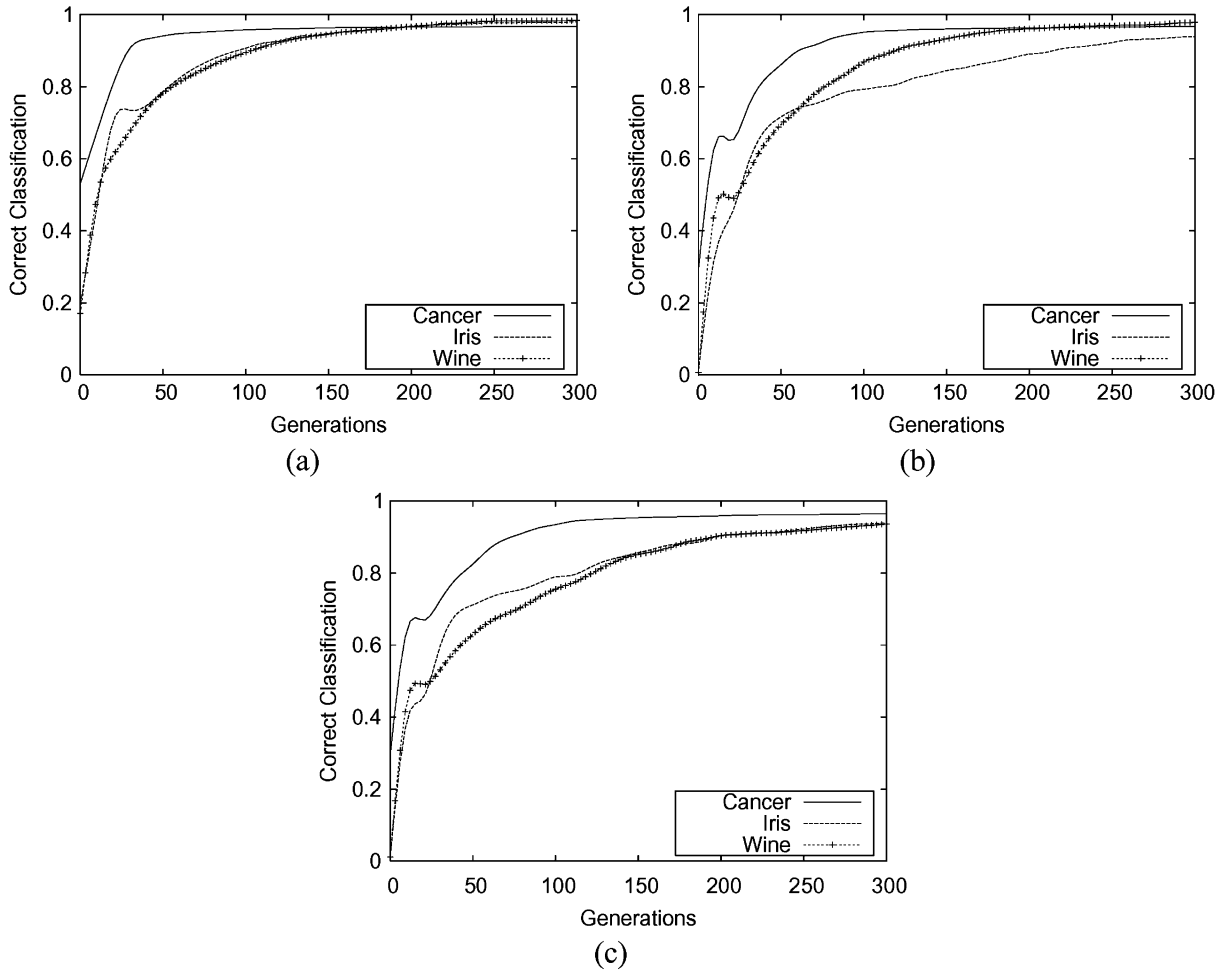

Fig. 6. Training performances of the different MGNN variants using SSP200 grouped by strategy. (a) MGNN-ep. (b) MGNN-rank. (c) MGNN-roul.

TABLE II
TESTING PERFORMANCE IN POOLED SAMPLING (a) DMRT OF DIFFERENT STRATEGIES USING POOLED SSPS. (b) DMRT OF DIFFERENT SSPS USING POOLED STRATEGIES

Strategy	TestError(%)		Connections			Generations	
	1	2	1	2	3	1	2
Cancer1: ep	3.28			80.87		243.20	
rank	3.33		68.46			301.20	
roul	3.05			76.40		282.93	
Iris: ep	6.17			56.38		453.93	
rank	7.28	7.28	47.06			477.60	
roul		8.43		55.13		422.07	
Wine: ep	5.12				132.83		694.93
rank	6.23		111.53				689.90
roul		8.46		120.36		445.00	

(a)

SSP	TestError(%)		Connections		Generations	
	1	2	1	2	1	2
Cancer1: SSP200	3.20		71.15		213.40	
SSP100	3.14			76.75		312.87
SSP5	3.33			77.83		301.07
Iris: SSP200	6.50		49.93		388.73	
SSP100	7.17	7.17	50.58		419.53	
SSP5		8.22		58.07		545.30
Wine: SSP200	6.18		118.15		529.67	
SSP100	6.14		119.44		607.93	607.93
SSP5	7.49			127.13		692.13

(b)

(columns) indicate that they have significantly different performances at $\alpha = 0.05$ level of significance.

Table II(a) shows the generalization performances of the different variants. These were computed based on their average performances over all SSPs (pooled SSP) using a total of 150 trials (50 trials $\times$ 3 SSP types). The table indicates that the generalization performances of the three approaches are not significantly different in the cancer problem. For the iris and wine problems, MGNN-ep and MGNN-rank performed better than MGNN-roul. Although MGNN-ep obtained lower classification errors in the iris and wine problems, these values are not significantly different from that of MGNN-rank.

In addition, the table shows that MGNN-rank significantly used less number of connections compared to the other two variants. The table also indicates that the scheduled mutation strategy improved the generalization performance of MGNN-ep over MGNN-rank at the cost of significantly using more connections. However, this added complexity did not negatively af-

TABLE III
TESTING PERFORMANCE OF DIFFERENT STRATEGIES USING SSP200

Strat-SSP		Accuracy(%)		Connections			Generations	
		1	2	1	2	3	1	2
Cancer1:	ep	3.14			77.12		167.00	
	rank	3.22		63.86			249.60	
	roul	3.23			72.46		223.60	
Iris:	ep	4.68			53.52		378.60	
	rank		7.46	42.50			436.60	
	roul		7.35		53.76		351.00	
Wine:	ep	4.68				129.70		619.20
	rank	4.91		106.52				599.20
	roul		8.96		118.24		370.60	

fect the generalization performance of MGNN-ep. Except for the wine problem, there is no significant difference in the average number of generations among all variants. It is interesting to note that although MGNN-roul significantly used the least number of generations, it significantly exhibited the worst generalization performance (only 8.46%). It also performed worst during the training phase (Fig. 5).

Table II(b) shows the effect of the different SSPs to the MGNN's generalization performance. The computation of SSP performance was based on the pooled trials (150 trials) of the three MGNN variants. The table indicates that there is no significant difference in the generalization performances of the three SSPs. What is apparent, however, is the significant superiority in convergence speed (number of generations) of SSP200 and SSP100 over SSP5. In the case of SSP100, it performed significantly as good as SSP200 in the iris and wine problems. This trend is also similar in their influence to MGNN's complexity. SSP200 consistently produced the least number of connections while SSP100 significantly performed as good as SSP200 in the iris and wine problems.

Table III shows the optimal performance of MGNN variants using SSP200. All variants required an average of less than 250 generations to achieve about 3% error in the testing data of the cancer problem.

For the iris problem, MGNN-ep had significantly better classification (4.68%) than MGNN-rank (7.46%) and MGNN-roul (7.35%). All variants required an average of less than 450 generations and an average of less than 55 connections.

For the wine problem, MGNN-ep and MGNN-rank had significantly better generalization than MGNN-roul. The wine problem required all variants to have relatively bigger number of connections and more generations compared to the other two problems. With the exception of MGNN-roul, MGNN-ep and MGNN-rank had relatively small classification errors in spite of the added complexity and generations.

C. Performance Comparison

Performance comparison in this section used the three groupings suggested by Prechelt for the cancer problem, namely: cancer1, cancer2, and cancer3. The well-behave nature of the cancer problem makes it an ideal choice for benchmarking. MGNN-ep using SSP200 was the variant chosen for comparison with the three types of BP architectures used by Prechelt [62]. These optimal BP architectures were generated by trial-and-error.

TABLE IV
MGNN VERSUS HAND-CODED BP ARCHITECTURE

Problem		MGNN Variants		
		$TestError(\%)$	$Overfitness$	$Corr$
cancer1:	MGNN-ep	3.14 ± 0.01	≤ 0	0.15
	L-BP	2.93 ± 0.18	0.55 ± 0.59	–
	P-BP	1.47 ± 0.60	4.48 ± 4.87	0.81
	N-BP	1.38 ± 0.49	3.10 ± 2.54	0.64
cancer2:	MGNN-ep	6.41 ± 0.01	≤ 0	0.23
	L-BP	5.00 ± 0.61	5.36 ± 10.21	–
	P-BP	4.52 ± 0.70	5.76 ± 06.70	0.51
	N-BP	4.77 ± 0.94	3.82 ± 01.90	0.14
cancer3:	MGNN-ep	4.61 ± 0.01	≤ 0	0.23
	L-BP	5.17 ± 0.00	0.35 ± 0.64	–
	P-BP	3.37 ± 0.71	3.37 ± 1.32	0.28
	N-BP	3.70 ± 0.52	3.33 ± 1.64	0.59

Table IV shows the performance of MGNN and the three types of BP, namely: L-BP (Linear BP), P-BP (Pivot BP), and N-BP (No shortcut BP). The second column indicates the classification error during testing while the last column indicates the correlation between validation error and testing error. Correlation data in L-BP were not computed in the paper of Prechelt. The learning algorithms of all BP variants used RPROP [64] algorithm with early stopping. L-BP had no hidden nodes and used the identity activation function. The two other approaches used multilayer ANN with at most 2-hidden layers. The details of the implementation can be found in [62].

While MGNN-ep had higher generalization errors, it consistently obtained the lowest standard deviation ($\sigma = 0.01$) in all the three problem instances. Moreover, no particular approach had the best performance for all the three problems. The best performer in cancer1 was N-BP, P-BP for cancer2, and P-BP for cancer3. Both, however, had relatively high standard deviations. Although MGNN-ep performed better than L-BP in cancer3, it performed worst in cancer1 and cancer2.

To have a better understanding of the relationship between correlation and overfitness, Table V shows the trend of the correlation from SSP5 to SSP200 in the three problems. There were

TABLE V
CORRELATION OF TRAINING, VALIDATION, AND TESTING PERFORMANCES IN MGNN

Strat	SSP5			SSP100			SSP200		
	TrVal	TrTst	ValTst	TrVal	TrTst	ValTst	TrVal	TrTst	ValTst
Cancer1: ep	0.03	0.06	0.17	0.10	0.30	-0.11	0.06	0.29	0.17
rank	0.25	0.46	0.27	0.28	0.06	0.13	0.02	0.04	0.02
roul	0.02	0.32	0.33	-0.16	-0.43	0.25	-0.12	0.53	-0.14
Iris: ep	0.92	0.88	0.90	0.48	0.09	0.37	0.22	0.21	0.33
rank	0.10	0.60	0.40	0.13	0.45	0.55	0.78	0.88	0.76
roul	0.90	0.90	0.90	0.53	0.78	0.70	-0.12	0.43	0.41
Wine: ep	0.88	0.92	0.82	0.42	0.42	0.17	0.50	0.50	0.36
rank	0.79	0.87	0.73	0.67	0.78	0.68	0.38	0.36	0.01
roul	0.87	0.96	0.85	0.62	0.78	0.53	0.74	0.79	0.74

three correlations tested, namely: TrVal (training versus validation); TrTst (training versus testing); ValTst (validation versus testing). While SSP5 in the wine problem had the worst training and generalization performances, it relatively obtained high correlations. On the other hand, the cancer1 problem where all variants and SSPs performed well, relatively received low correlations. Furthermore, majority of the cases indicate a decreasing trend of correlation from SSP5 to SSP100 to SSP200.

D. Discussion

Fig. 5 indicates that during training, the rank-based selection policy was a better option than the fitness-based selection policy. Also, the *scheduled* mutation policy was a better option than the ordinary stochastic mutation policy. MGNN-ep having both of these features consistently outperformed the two other approaches in the three classification problems. This consistency is further depicted in Fig. 6.

The trend (Table IV) in overfitness and correlation of BP variants with good solutions seems to indicate that a relatively high overfitness value has a corresponding high correlation value. In addition, it suggests that a small overfitness in the validation data can be useful in the generalization. Tolerating overfitness, however, may have a bad consequence if applied to other noisy problems. This trend will be further investigated in the future.

Table II(a) and Fig. 5 suggest that MGNN variants with good training performances have also good generalization performances. One way to achieve this is to use large SSP [see Fig. 4 and Table II(b)]. Large SSP speeds up convergence and minimizes network complexity. On the other hand, using a small SSP provides similar generalization performance, albeit much slower and requires significantly more connections. This slow convergence, however, may be useful in the later part of evolution to refine the search. This suggests that adapting or annealing SSP size may help improve MGNN's performace. Developing an *annealed* or *scheduled* SSP will be further investigated in the future.

Table V suggests a direct relationship between SSP size and the resulting correlation of its performance. Small SSP causes good correlation in training, validation, and testing. However, it produces networks with poor training and generalization performances. On the other hand, large SSP produces the opposite trend, i.e., poor correlations but superior performance in training and generalization. One possible explanation of this trend is the influence of SSP size in the local and global searching of MGNN. Using small SSP allows the network to converge to common local minima in the training, validation, and testing data. Consequently, this causes high correlations. On the other hand, using large SSP enables the network to escape these local minima but needs smaller SSP at the later part of the evolution to localize its search and improve the correlations. This speculation, which will be investigated in the future, requires a more detailed study using more classes of problems.

VIII. SUMMARY AND CONCLUSION

Simulation results indicate the following:

- rank-based selection is a better strategy than fitness-based selection during training and generalization;
- large SSP is superior than small SSP in terms of generalization performance, convergence speed, and complexity; and
- scheduled mutation probability ranging from 0.01 to 0.05 is superior than static mutation probability of 0.01.

The dynamic structure evolution and weight adaptation of MGNN allow it to develop the appropriate architecture in concurrence with the constraints of the problem. Its consistency was demonstrated in its classification performances in the three problems. This study suggests that for stochastic evolution of ANN structure and weights, the level of complexity or the number of generations is not a good parameter to indicate overfitness. Furthermore, it suggests that overfitness may happen even if the network uses less connections. Also, more number of generations do not necessarily result to overtraining. If the appropriate architecture and weights are chosen, more training improves and refines the classification ability of the ANN.

Studies in BP learning have indicated that networks using more connections suffer from poor generalization due to overfitness. As a consequence, many constructive, pruning, and evolutionary approaches place strong constraints in the structural evolution of the network to avoid too much complexity. By restricting structural evolution, however, it is prone to local optima due to "structural hill-climbing" problem [8]. Also, the system becomes too dependent in the weight learning algorithms which forces learning even to networks with poor architectures making the entire process inefficient. Moreover, most of the current structural evolutions are governed by rules

based on certain criteria. These static rules, however, are usually tuned only to certain problems and become inefficient if applied to another set of problems [65].

MGNN implementation, on the other hand, uses stochastic processes to adjust its weights and structures at the same time. Since there are no restrictions to govern its structure evolution and weight adaptation, it does not suffer from the "structural hill-climbing" problem observed in constructive and pruning approaches of ANN. It is true that using well-defined rules that restrict network growth will necessarily make the network compact. But this technique does not offer the same guarantee that the network has the appropriate architecture because of the strong constraints that the architecture evolution must satisfy.

Using several instances of the same problem in the benchmarking revealed that the generalization of MGNN and the BP variants varied to a considerable degree. One algorithm may be superior to the other in one problem instance but may have inferior performance in the other problem instance. This has important implications in benchmarking. Many researchers usually compare results from algorithms that use different sets of problem instances and make conclusions when the basis of comparison is not valid.

Moreover, generalization performance is also influenced by the amount of overfitness tolerated during training. In algorithms that use validation, it is important that the validation data has the proper "goodness-of-fit" to the generalization data, or else, the stopping criterion will stop prematurely or too late. In general, proper partitioning of the dataset is tantamount to the success of ANN learning since the prediction and generalization potential of the network is greatly influenced by the sampling procedure.

Some papers have suggested that multistrategy approach may be better than single-strategy approach especially for error surfaces that have complicated terrains [59], [66]–[68]. One possible area to be examined for future work is the effect of integrating several strategies. This can be carried out by allowing the system to switch from one strategy to another based on a certain function, schedule, or adaptation.

Results in SSP's influence to MGNN's performance indicated that small SSP had a tendency to prematurely converge. However, in cases where the state of the network is near the optimal solution, these small random mutations can be effective to localize the search process.

One strategy that can be implemented in the future is to use large SSP and large population size at the initial stage of evolution to avoid local optima problem. Using large SSP induces more diversity and makes the probability of finding similar individuals small. At the later part of the cycle, SSP and population size will be reduced based on a certain function or adaptation to localize the search.

"Noninvasive" evolution has been widely studied and many techniques have been developed to improve its search capability. MGNN is an early attempt to tackle ANN learning the "invasive" way. The results of our study suggest the feasibility of "invasive" evolution. Studies in "invasive" evolution open new insights and opportunities to better understand the various ways of exploiting the dynamic behavior, interactions, parallelism, and self-adaptation of the ANN population.

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.

[2] L. Fogel, A. Owens, and M. Walsh, Eds., *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[3] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[4] J. Koza, *Genetic Programming*. Cambridge, MA: MIT Press, 1992.

[5] S. Grossberg, Ed., *Neural Networks and Natural Intelligence*. Cambridge, MA: MIT Press, 1988.

[6] D. Rumelhart and J. McClelland, Eds., *Parallel Distributed Processing: Explorations in Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986.

[7] J. M. Zurada, Ed., *Introduction to Neural Systems*. St. Paul, MN: West, 1992.

[8] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–64, Jan. 1994.

[9] N. Nikolaev and H. Iba, "Learning polynomial feedforward neural networks by genetic programming and backpropagation," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 337–350, Mar. 2003.

[10] F. Leung, H. Lam, S. Ling, and P. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 79–88, Jan. 2003.

[11] J. T. Alander. (1995) An indexed bibliography of genetic algorithms and neural networks. Univ. Baasa, Dept. Information Technol. and Production Economics. [Online]. Available: ftp://ftp.uwasa.fi/cs_report94-1/gaNNbib.ps.Z

[12] R. K. Belew, J. McInerney, and N. N. Schraudolph, "Evolving networks: Using the genetic algorithm with connectionist learning," in *Artificial Life II*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds. Reading, MA: Addison-Wesley, 1992, pp. 511–547.

[13] P. J. B. Hancock, "Coding strategies for genetic algorithms and neural nets," Ph.D. dissertation, Dept. Comput. Sci. Math., Univ. Stirling, 1992.

[14] S. A. Harp and T. Samad, "Optimizing neural networks with genetic algorithms," in *Proc. Amer. Power Conf.*, vol. 54, 1992, pp. 1138–1143.

[15] M. Potter, "A genetic cascade-correlation learning algorithm," in *Proce. COGANN'92 Int. Workshop Combinations of Genetic Algorithms and Neural Networks*, 1992, pp. 123–133.

[16] P. Robbins, A. Soper, and K. Rennolls, "Use of genetic algorithms for optimal topology determination in back propagation neural networks," in *Proc. Int. Conf. Artificial Neural Networks and Genetic Algorithms*, 1993, pp. 726–730.

[17] 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 *Proc. Conf. Combinations of Genetic Algorithms and Neural Networks*, 1992, pp. 1–37.

[18] W. Schiffmann, M. Joost, and R. Werner, "Application of genetic algorithms to the construction of topologies for multilayer perceptrons," in *Proc. Int. Conf. Artificial Neural Networks and Genetic Algorithms*, 1993, pp. 675–682.

[19] D. Whitley and C. Bogart, "The evolution of connectivity: Pruning neural networks using genetic algorithms," in *Proc. Int. Joint Conf. Neural Networks*, 1990, pp. 134–137.

[20] P. Wilke, "Simulation of neural network and genetic algorithms in a distributed computing environment using neurograph," in *Proc. World Congr. Neural Networks*, 1993, pp. I269–I272.

[21] V. Manniezzo, "Searching among search spaces: Hastening the genetic evolution of feed-forward neural networks," in *Proc. Int. Joint Conf. Neural Networks and Genetic Algorithms*, 1993, pp. 635–642.

[22] P. Kohn, "Combining genetic algorithms and neural networks," M.S. thesis, Dept. Comput. Sci., Univ. Tenessee, Knoxville, TN, 1994.

[23] E. Keedwell, A. Narayanan, and D. Savic, "Using genetic algorithms to extract rules from trained neural networks," in *Proc. Genetic Evolutionary Computation Conf.*, vol. 1, W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, Eds., Jul. 13–17, 1999, p. 793.

[24] J. Bishop, M. Bushnell, A. Usher, and S. Westland, "Genetic optimization of neural network architectures for color recipe prediction," in *Proc. Int. Conf. Artificial Neural Networks Genetic Algorithms*, 1993, pp. 719–725.

[25] P. de la Brassinne, "Genetic algorithms and learning of neural networks," *Bulletin Scientifique de l'Association des Ingenieurs Electriciens sortis de l'Institut Electrotechnique Montefiore*, vol. 106, no. 1, pp. 41–58, 1993.

[26] H. Braun and J. Weisbrod, "Evolving neural feedforward networks," in *Proc. Int. Conf. Artificial Neural Networks Genetic Algorithms*, 1993, pp. 25–32.
[27] V. Petridis, S. Kazarlis, and A. Papaikonomou, "A genetic algorithm for training recurrent neural networks," in *Proc. Int. Joint Conf. Neural Networks*, Nagoya, Japan, 1993, pp. 2706–2709.
[28] W. Philipsen and L. Cluitmans, "Using a genetic algorithm to tune potts neural networks," in *Artificial Neural Networks and Genetic Algorithms*, R. Albrecht, C. Reeves, and N. Steele, Eds. New York: Springer-Verlag, 1993, pp. 650–657.
[29] D. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. 11th Int. Joint Conf. Artificial Intelligence*, 1989, pp. 762–767.
[30] R. Mitchell, J. Bishop, and W. Low, "Using a genetic algorithm to find the rules of a neural network," in *Artificial Neural Networks and Genetic Algorithms*, R. Albrecht, C. Reeves, and N. Steele, Eds. New York: Springer-Verlag, 1993, pp. 664–669.
[31] M. Kouchi, H. Inayoshi, and T. Hoshino, "Optimization of neural-net structure by genetic algorithm with diploidy and geographical isolation model," *J. Jap. Soc. Artif. Intell.*, vol. 7, no. 3, pp. 509–517, May 1992.
[32] D. B. Fogel, "Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe," in *Proc. Amer. Power Conf.*, 1993, pp. 875–879.
[33] D. B. Fogel and L. J. Fogel, "Method and apparatus for training a neural network using evolutionary programming," Patent 5 214 746, May 25, 1993.
[34] D. Chakraborty and N. Pal, "A novel scheme for multilayered perceptrons to realize proper generalization and incremental learning," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 1–14, Jan. 2003.
[35] S.-C. Ng, C.-C. Cheun, and S.-H. Leung, "Magnified gradient function with deterministic weight modification in adaptive learning," *IEEE Trans. Neural Netw.*, vol. 15, no. 6, pp. 1411–1423, Nov. 2004.
[36] J. Schmidhuber, "Accelerated learning in back-propagation nets," in *Connectionism in Perspective*. Amsterdam, The Netherlands: Elsevier, 1989, pp. 439–445.
[37] P. A. Castillo, V. Rivas, J. J. Merelo, A. Prieto, and G. Romero, "G-prop II: Global optimization of multilayer perceptrons using GAs," in *Proc. Congr. Evolutionary Computation*, vol. 3, P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, Eds., 1999, pp. 2022–2028.
[38] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems, Denver 1989*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, vol. 2, pp. 524–532.
[39] R. Setiono and H. Liu, "Improving backpropagation learning with feature selection," *Applied Intelligence*, vol. 6, no. 2, pp. 129–139, 1996.
[40] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[41] D. B. Fogel, "Phenotypes, genotypes, and operators in evolutionary computation," in *Proc. 1995 IEEE Int. Conf. Evolutionary Computation*, Piscataway, NJ, 1995, pp. 193–198.
[42] S. Lawrence, C. L. Giles, and A. C. Tsoi, "Lessons in neural network training: Overfitting may be harder than expected," in *Proc. 14th Nat. Conf. Artificial Intelligence (AAAI'97)*, Menlo Park, CA, 1997, pp. 540–545.
[43] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 1, pp. 76–86, Jan. 1992.
[44] P. J. B. Hancock, "Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification," in *Proc. COGANN Workshop*, Baltimore, MD, 1992, pp. 108–122.
[45] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.
[46] I. Rechenberg, *Evolutionsstrategie*. Berlin, Germany: Friedrich Frommann Springer-Verlag, 1994.
[47] Y. Davidor, "Foundations of genetic algorithms," in *Epistasis Variance: A Viewpoint on GA-Hardness*. San Marteo, CA: Morgan Kaufmann, 1991, pp. 23–35.
[48] S. A. Kaufmann, *Lectures in the Science of Complexity*. Reading, MA: Addison-Wesley, 1989, vol. 1, pp. 567–618.
[49] J. R. McDonnell and D. Waagen, "Neural network structure design by evolutionary programming," in *Proc. 2nd Annu. Conf. Evolutionary Programming*, D. B. Fogel and W. Atmar, Eds., La Jolla, CA, 1993, pp. 79–89.
[50] ——, "Evolving neural network connectivity," in *Proc. Amer. Power Conf.*, 1993, pp. 863–868.
[51] ——, "Evolving neural network architecture," *Proc. SPIE-Int. Soc. Opt. Eng.*, vol. 1766, pp. 690–701, 1992.
[52] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.
[53] X. Yao and Y. Liu, "Making use of populations information in evolutionary artificial networks," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 28, no. 3, pp. 417–425, Jun. 1998.
[54] M. Islam, X. Yao, and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 820–834, Jul. 2003.
[55] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez, "Covnet: A cooperative coevolutionary model for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 575–596, May 2003.
[56] H.-P. Schwefel, *Evolution and Optimum Seeking*. New York: Wiley, 1995.
[57] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart, Germany: Frommann-Holzboog Verlag, 1973.
[58] D. Fogel, *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995.
[59] H. Schwefel, "Deep insight from simple models of evolution," *BioSystems*, vol. 64, pp. 189–198, 2002.
[60] C. Igel and M. Kreutz, "Operator adaptation in evolutionary computation and its application to structure optimization of neural networks," *Neurocomput.*, vol. 55, no. 1–2, pp. 347–361, Sep. 2003.
[61] I. D. Falco, A. D. Ciopa, and E. Tarantino, "Mutation-based genetic algorithm: Performance evaluation," *Appl. Soft Comput.*, vol. 1, pp. 285–299, 2002.
[62] L. Prechelt, "Proben1—a set of neural network benchmark problems and benchmarking," Fakultat fur Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep., Sep. 1994.
[63] P. M. Murphy and D. W. Aha, "UCI repository of machine learning databases," Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, 1994.
[64] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. IEEE Conf. Neural Networks*, San Francisco, CA, Apr. 1993, pp. 586–591.
[65] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Feb. 1997.
[66] G. Ochoa, I. Harvey, and H. Buxton, "On recombination and optimal mutation rates," in *Proc. Genetic and Evolutionary Computation Conf.*, vol. 1, Orlando, FL, 1999, pp. 488–495.
[67] P. J. Angeline, "Adaptive and self-adaptive evolutionary computations," in *Computational Intelligence: A Dynamic Systems Perspective*, M. Palaniswami and Y. Attikiouzel, Eds. Piscataway, NJ: IEEE Press, 1995, pp. 152–163.
[68] R. Hinterding, Z. Michalewicz, and A. E. Eiben. (1997) Adaptation in evolutionary computation: A survey. *Proc. IEEE Conf. Evolutionary Computation, World Congr. Computational Intelligence*. [Online], pp. 65–69

Paulito P. Palmes received the M.S. degree in computer science from the Ateneo de Manila University, Philippines, and the B.S. degree in applied mathematics (*cum laude*) from the University of the Philippines, Visayas.

In 1991, he joined the faculty of the Department of Physical Sciences and Mathematics, University of the Philippines. He is currently with the Laboratory for Neuroinformatics, RIKEN Brain Science Institute, Saitama, Japan while working for his Doctor of Engineering degree in information and computer sciences. His research interests center on the development of intelligent agents using machine learning and evolutionary computation.

Mr. Palmes is a Member of the Association of Computing Machinery (ACM).

Taichi Hayasaka received the Doctor of Engineering degree from Toyohashi University of Technology, Toyohashi, Japan, in 1999.

He became a Research Associate in the Department of Information and Computer Sciences, Toyohashi University of Technology, in 1999. He is currently a Lecturer in the Department of Information and Computer Engineering, Toyota National College of Technology, Toyota, Japan. His scientific interests include statistical model selection, machine learning, and human object recognition.

Shiro Usui (S'71–M'74–SM'89–F'94) received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkely in 1974.

In 1979, he became a Professor and Head of the Biological and Physiological Engineering Laboratory, Department of Information and Computer Sciences, Toyohashi University of Technology, Toyohashi, Japan. His research interests are focused on a multitude of topics related to neuroscience, neurocomputing, and neuroinformatics. In 1999, he initiated an interdisciplinary national project: Neuroinformatics Research in Vision, which is supported by the Target Oriented Research and Development for Brain Science in Japan. Since 2003, he has been the Head of Neuroinformatics Laboratory at RIKEN Brain Science Institute. He has been an Associate Editor of *Neural Networks*, *Neurocomputing*, and several biomedical engineering journals.

Dr. Usui is the President of the Japanese Neural Network Society (JNNS), a Fellow of the Institute of Electrical, Information, and Communication Engineers (IEICE), and a Member of the Society for Neuroscience. In 1996, he received the Inose Award and the Best Paper Award from the IEICE of Japan.