Minimax Design of Neural Net Controllers for Highly Uncertain Plants

Anthony V. Sebald and Jennifer Schlenzig

Abstract—This paper discusses the use of evolutionary programming (EP) for computer-aided design and testing of neural controllers applied to problems in which the system to be controlled is highly uncertain. Examples include closed-loop control of drug infusion and integrated control of HVAC/lighting/utility systems in large multi-use buildings. The method is described in detail and applied to a modified Cerebellar Model Arithmetic Computer (CMAC) neural network regulator for systems with unknown time delays. The design and testing problem is viewed as a game, in that the controller is chosen with a minimax criterion; i.e., minimize the loss associated with its use on the worst possible plant. The technique permits analysis of neural strategies against a set of feasible plants. This yields both the best choice of control parameters and identification of that plant which is most difficult for the best controller to handle.

I. INTRODUCTION

SYSTEMS with nonlinear, time-varying and poorly understood plant equations present many difficulties for adaptive control algorithms. In some cases, notably closed-loop control of drug infusion [1], the underlying plant dynamics are essentially unknowable a priori. More difficulty is encountered if such systems are subject to unmeasured or stochastic disturbance inputs (see Fig. 1). Typical adaptive approaches hypothesize a plant model structure and use on-line system identification to determine appropriate plant parameters. These estimates are then used to generate appropriate control values. While useful for many applications, the relative lack of intelligence of these controllers limits their utility in the most stringent applications. Finally, all of these conditions are exacerbated if the plant has an unknown pure delay between the time a control is applied and the time it begins to take effect.

Efforts in the application of neural networks for control generally fall into two classes. In the first approach, all training is done off-line. Motivated by the now well-known fact that networks can approximate virtually any useful nonlinear function [2]–[4], these controllers learn the fixed inverse plant dynamics and are then simply applied to the system being controlled. Here, the performance information available during actual control of the plant is ignored. This approach is suitable for fixed plants (e.g., the cart-pole problem).

In the second approach, neural controllers use on-line learning to improve performance based on the effect of their previous control outputs. This approach is emphasized in this paper, since the class of problems of interest here requires a great deal of adaptability.

The goal of this paper is to describe a computer-aided design method for the second type of controller. In particular, we investigate the use of evolutionary programming (EP) in combination with a minimax optimization approach to provide a priori computer-aided design for these networks.

First, the design problem is stated in detail and then the proposed design algorithm is elucidated. Finally, an example is given.

II. THE DESIGN PROBLEM

There are three phases to designing a suitable controller: 1) The structure and appropriate parameters of the controller are selected. Some parameters must be specified off-line and some are learned on-line. 2) Some information is required regarding the plant that is to be controlled. In more conventional control design problems, one may know that the plant input/output variables satisfy a known differential equation. At the other extreme, very little may be known about the plant save some experimental input/output data. And 3) Some sort of performance index must be specified. Given a particular plant and a particular controller, this index computes a score for the combination in question. The control design problem then essentially consists of finding the best set of control parameters to cope with the plant in question so as to generate good closed-loop performance. This design process cannot typically be accomplished without computer-aided design and testing.

For problems of interest here, both the controller and the plant are uncertain. All neural controllers have parameters that must be specified off-line. These typically include the network architecture (e.g., the number of nodes and layers and types of squashing functions) and parameters of the online learning algorithm (e.g., learning rate). The actual weights and biases and sometimes even the network architecture are then learned on-line [4]. In addition, the plant is not completely specified. Typically, there is a vector of parameters that must
be chosen to specify an individual plant. The set over which these parameters are permitted to range comprises the set of admissible plants.

The controller designer’s true problem is therefore to choose the best controller parameters in such a way that the resulting controller will perform adequately over the entire set of admissible plants. This can be described mathematically as follows.

Given a plant, \( x_{k+1} = f(x_k, \theta, u_k) \), a control strategy, \( u_k = g(x_k, \gamma) \), parameter vectors \( \theta \in \Theta \) and \( \gamma \in \Gamma \), and a performance index, \( J(\theta, \gamma) \), find the best controller (i.e., find the best \( \gamma \in \Gamma \)). As the performance index is a function of both \( \theta \) and \( \gamma \), this is an ill-posed problem unless the dependence on \( \theta \) is eliminated. This is commonly done in one of two ways: Bayes or minimax. In the Bayes context, a prior probability distribution, say \( \tau \), is hypothesized on \( \theta \) and then \( \gamma \) is chosen as

\[
\arg \min_{\gamma \in \Gamma} \{ E_\tau J(\theta, \gamma) \} \tag{1}
\]

where \( E_\tau \) denotes expectation with respect to \( \tau \).

In the minimax context, \( \gamma^* \in \Gamma \) is chosen such that

\[
\max_{\theta \in \Theta} J(\theta, \gamma^*) \leq \max_{\theta \in \Theta} J(\theta, \gamma) \quad \forall \gamma \in \Gamma \tag{2}
\]

In either case, the problem can be characterized as a mathematical game as shown in Fig. 2.

A. The Choice between Minimax and Bayes

There has been considerable debate over the relative merits of adopting a Bayes or minimax approach. Minimax enthusiasts are quick to point out that the prior, \( \tau \), must be chosen with some rationale. Bayes enthusiasts point out that minimax can be overly pessimistic in that in order to protect against the worst case (which may not happen very often), the minimax solution accepts unnecessarily large losses in more benign and possibly more commonly encountered circumstances.

Neither of these arguments is completely sound. There are reasonable methods for estimating the underlying prior \([5]\). Further, the pessimism sometimes associated with minimax can be avoided \([6]\) if the performance index, \( J \), is modified to an incremental form \( \Delta J \):

\[
\Delta J(\theta, \gamma) = J(\theta, \gamma) - J(\theta, \gamma_{\text{OPT}}), \tag{3}
\]

where \( \gamma \) is the current controller candidate and \( \gamma_{\text{OPT}} \) is the optimal controller one would use if the true value of \( \theta \) were known. The minimax solution for \( \Delta J(\theta, \gamma) \) will then minimize the maximum difference between the value of \( J \) for the current controller candidate, \( \gamma \), and the performance that would have been achieved had we known the correct value of \( \theta \) and used the best possible \( \gamma_{\text{OPT}} \) (that which yields the smallest \( J \) when the true \( \theta \) is used). This technique spreads the penalty evenly around the space \( \Theta \).

Furthermore, under fairly benign conditions, the minimax solution sometimes corresponds to the Bayesian solution associated with a particular prior probability distribution, \( \tau \) on \( \Theta \) \([7]\). Since this prior produces the maximum \( J \) when the minimax \( \gamma \) is used, it is called the least favorable prior.

Note that this is not the same as the standard Bayesian concept of a prior distribution on the plants that are likely to be observed. In the standard Bayesian approach, each plant’s probability of presenting itself to the controller (i.e., the probability that plant \( i \) will appear) defines the prior probability distribution over the plant population. In contrast, the minimax least favorable prior is the probability distribution on the plant that would cause the unknown minimax controller the most loss, \( J \).

Consider the difficulty associated with specifying a least favorable prior in the control context. It must be determined by asking what probability distribution should Nature (the player choosing the plant parameters) place on plant parameters in order to make the controller’s job most difficult? Of course, the controller is free to respond in the best possible way to any choice by Nature. For example, the controller probably has extensive on-line identification ability. The choice of least favorable prior must therefore anticipate the controller’s best response to each choice of \( \Theta \) and mix the results so as to create the highest Bayesian loss to the controller. Therefore, the choice of such a prior must explicitly take account of all three components of the problem: 1) the set of all admissible plant parameters, \( \Theta \), 2) the nature of all possible controller characteristics (given both its architecture and all possible parametric variations in \( \Gamma \)), and finally 3) the performance index, \( J \). In effect, while the overall minimax controller might be Bayes with respect to some least favorable prior, direct determination of the prior is often extremely difficult.

In practice, we do not use the Bayes/minimax connection for at least two reasons. First, there is typically no guarantee that the minimax-Bayes connection exists. However, existence of at least a close approximation to the minimax solution is guaranteed because the admissible parametric search spaces can all be chosen to be closed (e.g., the interval \([0, 1]\)). Second, while the form of the least favorable prior is unknown, we can directly perform the nested minimax optimization without regard to priors or Bayesian theory.

Finally, the minimax criterion may be favored over the Bayes criterion when the cost of failure is great, such as is the case in medical control problems. The standard Bayesian controller minimizes the average cost of a controller over a group of patients. This seems inappropriate because controllers may perform very poorly on patients who occur infrequently and therefore have very little weight in the overall average penalty.
Note that the minimax approach identifies the $\gamma^* \in \Gamma$ that solves equation (2), and therefore yields both the best controller and its worst performance when trying to cope with all plants in $\Theta$. If the performance of $\gamma^*$ is satisfactory, then it can be used with more confidence because it has been tested across all anticipated plants. If not, one needs to change the set $\Gamma$ and try again. The algorithm also identifies the worst plant, $\theta^* \in \Theta$, corresponding with $\gamma^*$. The search for a new control strategy is greatly facilitated by knowledge of where (i.e., $\theta^*$) the best available control strategy (i.e., $\gamma^*$) is performing worst.

B. Factors in the Choice of a Numerical Minimax Procedure

Given the adoption of the minimax criterion, the following components are required:

1) A performance index. This is typically a function defined by a combination of rules, simulations and standard mathematical formulae. It is usually neither differentiable nor convex.

2) A set of constraints on both the plant parameters and the controller parameters. These are also sometimes expressed as a combination of bounds and rules or formulae defined on either the parameters themselves or the resulting plant trajectories.

3) A numerical procedure for finding the minimax controller parameter vector $\gamma^*$ in equation (2).

Since the minimax search involves a nested optimization, it must be implemented as efficiently as possible. Without human intervention, it must be able to determine the global extremum of nonconvex and nondifferentiable performance indices with potentially many local extrema. It must be able to handle essentially arbitrary constraints on both the plant and controller parameter spaces $\Theta$ and $\Gamma$.

III. THE OPTIMIZATION ALGORITHM

We have tried a variety of approaches. Gradient-based techniques are simply not feasible because they are prone to entrapment at local extrema. We encountered problems with simulated annealing if the response surface being searched contained large flat regions. The temperature tended to reach "freezing" before the search could escape the plateau. Attempts to slow the cooling schedule resulted in no adequate solution being found in a reasonable amount of time. Alternatively, evolutionary programming (EP) has proven to provide satisfactory performance on these difficult optimization problems.

A. Evolutionary Programming

Simulating natural evolution as a means for generating artificial intelligence was originally offered by Fogel [8]–[10]. Detailed descriptions of the evolutionary programming algorithm and its variations can be found in [11], [12]. For nested minimax optimization, the EP algorithm is:

1) Establish an initial population of $P$ controllers, each initialized at random over the corresponding constraint set.

2) Score each organism in the controller population by performing a complete EP maximization over the plant parameter space $\Theta$ (see below).

3) Use random competition to rank the controller organisms. This is done by conducting a probabilistic round-robin tournament to assign "wins" to individual sets of controller parameters, following [12]. Controllers that minimize the score from step 2) are favored over those that do not.

4) Keep the best performing half of the controllers in the population (in terms of number of wins) to use as parents for the next generation.

5) Mutate the new parent controllers to fill out the population. The new parents are mutated by varying each parameter using a Gaussian random variable with zero mean and preselected variance. In the current experiments, the variance is a fixed function of the range of each parameter, although more sophisticated methods for self-adapting the variance have been proposed in [13], [14].

6) If the desired number of generations has not been completed, go to step 2, otherwise stop.

The EP maximization required by step 2) must be done for each candidate solution in the controller population. This is conducted in exactly the same manner as above, with the following slight modifications in each step:

1) Establish a randomly selected initial population of $M$ plants consistent with the corresponding constraints.

2) Score the plant population by simulating control of the plant in question with the particular controller currently being evaluated from the outer evolutionary optimization loop.

Steps 3)–6) are performed as above except that they are performed on plant parameter sets rather than control parameter sets. The probabilistic round-robin tournament is used here as well. Plants that yield the worst performance in light of the score from the control simulations are statistically retained through competition to become parents of successive iterations of evolution. Each iteration (i.e., generation) in the outer loop then requires $M \times P$ objective function evaluations (i.e., control simulations).

IV. AN EXPERIMENT

A. Background

The robotics control literature has reported work on neural network algorithms for control [15]. These algorithms are attractive because they require very little a priori knowledge of the underlying plant and, after training, their real-time computational requirements are relatively small. Neural networks are also able to accurately approximate an extremely wide class of functions (see [2]–[4] for examples), and therefore make attractive candidates for controllers—provided they can be trained sufficiently quickly.

Difficult control problems require fast adaptation on-line, but learning all of the weights in a complex multi-layer network is a demanding chore. If one could eliminate the
learning of hidden layer weights, presumably faster learning would result. Both the modified cerebellar model arithmetic computer (CMAC) algorithm in [16] and the adaptive critic 

Table I

<table>
<thead>
<tr>
<th>Table I</th>
<th>The Constraint Satisfaction Algorithm for Generating Feasible Patients (Plants)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plant parameters:</td>
<td>$a, b, d, D_0, D_1, D_2$ and $D_3$.</td>
</tr>
<tr>
<td>Simple hard constraints</td>
<td>The first four parameters ($a, b, D_0$ and $d$) in the plant were constrained to the vectors:</td>
</tr>
<tr>
<td></td>
<td>Upper $= [0.9672, -0.25, 5.740, 60]$</td>
</tr>
<tr>
<td></td>
<td>Lower $= [0.9355, -5.00, 2.565, 15]$</td>
</tr>
<tr>
<td>Relative constraints:</td>
<td>The steady-state solution, $D_0/(1 - a)$, was constrained to the interval [80, 120].</td>
</tr>
<tr>
<td></td>
<td>The remaining $D$ parameters were constrained such that the entire solution trajectory remained in the interval [80, 120].</td>
</tr>
<tr>
<td>ALGORITHM:</td>
<td>1) The first four parameters are randomly chosen from the intervals in (5).</td>
</tr>
<tr>
<td></td>
<td>2) Given a candidate $a$ from 1), $D_0$ is randomly chosen in the interval specified by $D_0/(1 - a)$ in [80, 120].</td>
</tr>
<tr>
<td></td>
<td>3) Candidates for the remaining parameters, $D$, are randomly selected from a set of likely upper and lower bounds chosen to reduce the number of infeasible solutions.</td>
</tr>
<tr>
<td></td>
<td>4) The resulting complete parameter set is then simulated to make certain that the entire solution trajectory remains in the interval [80, 120].</td>
</tr>
<tr>
<td></td>
<td>5) If success is not achieved, two cases are considered:</td>
</tr>
<tr>
<td></td>
<td>a) if a predetermined number of trials (e.g. 10) has not occurred, a new set of candidates for $D_1, D_2$ and $D_3$ is chosen and the simulation repeated (i.e., go to step 4).</td>
</tr>
<tr>
<td></td>
<td>b) if no admissible values have been found after the desired number of trials, go to step 1).</td>
</tr>
</tbody>
</table>

As discussed in [18], these algorithms have two important advantages. First, they have no hidden nodes and can learn in real time on the actual plant being controlled. Second, their architecture lends itself very well to exploitation of human experience and permits the design engineer and the neural network algorithm to cooperate nicely. The latter requires further comment. Two major areas of human expertise can be exploited in the design of the underlying grid and the online learning scheme. First, human experience can be very valuable in specifying the outward extent of the grid as well as its density and whether it should have uniformly distributed grid spacing. Second, the concept of online reinforcement learning is greatly facilitated because the human is usually very well aware of how to determine whether or not the controller is performing well. This is in rather stark contrast to the ACE/ASE concept of having the neural controller learn for itself when the system is performing poorly prior to initiating a change in the control. Reinforcement learning basically encourages the controller to do more of the same if it is succeeding and to change (via the weights) if it is not.

B. An Example

To illustrate the method, we describe the design of a modified CMAC neural controller. This example was motivated by work on closed-loop control of blood pressure for patients in cardiac surgery [16], [21]. The patient is a model of the response of the patient's mean arterial pressure (MAP) to an input of the vasodilating drug Sodium Nitroprusside (SNP). The goal of the controller in this example is to regulate the MAP to 70 mmHg for 700 time steps (approximately 23 minutes).

The patient (i.e., plant) model is

$$X[k+1] = \alpha X[k] + \beta [k+d] + D_0 + D_1 k + D_2 k^2 + D_3 k^3 \tag{4}$$

where $X[k]$ is the MAP in mmHg at time step $k$, $d$ is the transport delay and $u$ is the drug infusion rate. The parameters $D_0, D_1, D_2, D_3$ model the effects of external disturbances such as the input of additional drugs. This model was suggested in [22]. The constraints on $(\alpha, \beta, d, D_0, D_1, D_2, D_3)$ cannot be specified simply with upper and lower bounds. Instead, the set of feasible patients is additionally constrained by realistic patient behavior, given no SNP input and no external disturbances. To find a feasible patient the parameters $(\alpha, \beta, d, D_0)$ are respectively constrained to the ranges

$$\text{upper} = [0.9672, -0.25, 5.740, 60]$$
$$\text{lower} = [0.9355, -5.00, 2.565, 15]. \tag{5}$$

These ranges were determined based on prior knowledge of the trajectories obtained in [22]. Here, one uses all available information including plant damping levels, physiological information on time for the drug to flow from the infusion pump to the drug receptor sites in the body (which depends on cardiac output), and so forth.

Furthermore, since typical patients have undisturbed MAP in this range, the non-perturbed, steady-state trajectory, $D_0$, must remain within [80, 120] mmHg for $k$ in [0, 700]. Finally, the remaining parameters of the cubic polynomial, $D_1, D_2$ and $D_3$, are constrained such that the entire solution trajectory remains in the interval [80, 120]. These steps ensure that the controller is not wrongly penalized for poorly controlling an unrealistic patient. Satisfying these constraints is neither unimportant nor simple to achieve. Important details of the procedure are given in Table I. A sample of feasible uncontrolled plant trajectories is shown in Fig. 3.

The control input for each time step is calculated as follows. First, the state vector,

$$X[k], \Delta X[k] = X[k] - X[k-1], \tag{6}$$

of the system (4) is determined. Second, a CMAC encoder converts each point in state space into a vector of generalized inputs and hashes these inputs to provide a set of weights whose sum is the control $u(k)$ in (4). See [19] for details.
on the CMAC computation. Note that because the control is a
drug infusion rate, its value is required to be non-negative.

The CMAC controller is based on a number of parameters
that must be chosen off-line by the designer. The performance
of the controller depends on these parameters, but there is little
quantitative or theoretical information regarding methods for
choosing these parameters. The specific controller parameters are:

1) $N$, the number of weights selected, which specifies the
number of weights summed to determine the control.
2) $M$, the memory size for the hash coder output. This
determines the probability of collisions due to hash
coding of the CMAC encoder output.
3) $\bar{W}$, the default weight change (7).
4) $b_0$, the initial learning rate (8).
5) $D_a$, the delay assumed by the controller, and
6) $\delta$, the number of time steps between learning rate
changes [see equation (8)].

These parameters were constrained, respectively, to lie within
the following ranges:

- Upper = [75, 2400, 0.05, 0.100, 60, 50]
- Lower = [30, 1000, 0.001, 0.006, 15, 10].

These bounds were chosen from physiological insights and
from experience with manually generated control experiments.
For example, it is believed that the true delay in a patient
ranges from 30 seconds to 120 seconds (the time is stepped at
two-second intervals). Bounds on parameters 1), 2) through 4),
and 6) were educated guesses based on the CMAC response to
variations in the parameters in a limited number of experiments
on individual plants.

1) The Weight Learning Scheme: The weights are adjusted
at each time step so the controller can adapt to the actual
system it is currently controlling. Initially the weights ($W_i$)
are zero. Each time a weight is selected, $W_i$ is changed by:

$$\Delta W_i \triangleq W_i[e[k](\rho/N)], \text{ if } W_i \geq 1.0 e - 4$$

$$\Delta W_i \triangleq \bar{W}, \text{ otherwise}$$

(7)

where $W_i$ is the value of the $i$th weight, $e[k] \triangleq T[k] - X[k]$, $\rho$
is the learning rate and $\bar{W}$ is specified off-line. The learning
rate, $\rho$, is adjusted as follows:

$$\text{if } k = 0 : \quad \rho = b_0$$

else if $k = j \cdot \delta$ : \quad IF $\left(\Delta X[k] > 0 \text{ and } X[k] > 70\right)$

OR $\left(\Delta X[k] < 0 \text{ and } X[k] < 70\right)$

THEN $\rho = \rho + ||\Delta X[k]||/N$. (8)

where $j = 1, 2, \ldots$ and $N$ is the number of weights selected
and $\delta$ is the learning adjustment rate. The maximum possible
learning rate, $\rho$, is 1.0. Note that the learning scheme (7) only
changes the weights if the current error, $e[k]$, is nonzero. The
set of indices $\{i\}$ of the weights to be changed is determined
by the delayed state vector:

$$X[k-D_a], \Delta X[k-D_a] \triangleq X[k-D_a]-X[k-D_a-1])$$

(9)

The delayed state is used to account for the fact that the
pure plant delay, $d$, requires blame for error to be allocated to
the weights that determined the controller output $d$ time steps
in the past. Here, $D_a$ is used in place of $d$ because that is the
delay assumed by the controller.

2) The Performance Index: The performance index used for
each EP maximization search is a modified sum of the
squares:

$$J = \left\{ \begin{array}{ll}
1.0 \times 6, & \text{if } (X[k] \leq 50 \text{ for any } k) \\
0.125, & \text{or } (X[k] \geq 120 \text{ for any } k) \\
\sum_{k=1}^{N} [X[k] - T[k]]^2 \text{ otherwise}
\end{array} \right.$$  \hspace{1cm} (10)

where $T[k]$ is the target pressure at time index $k$, and $T[k]$ =
70 for all $0 \leq k \leq 700$ in the current experiment.

Incorporating the use of a very large penalty when the
controlled sequence $X[k]$ was deemed grossly unacceptable
significantly reduced the required computation because the
entire inner loop (search for the worst case patient) could be
terminated after the first such unacceptable performance was
discovered.

In the current experiment, the plant was initialized at 100
mmHg with the goal of finding the set of controller parameters
that is minimax with respect to $J$ in (10). Fifty generations
were executed in each control evaluation (minimization loop)
and 20 generations were executed in each patient evaluation
(maximization loop). The population for both loops was main-
tained at 50 parents. Despite these relatively small numbers of
generations the final design appears to be quite successful.

V. RESULTS

Computation time of the complete, nested, minimax solution
was acceptable, requiring approximately 39 hours on an HP
720 workstation. Figure 4 describes the average performance
of all parents in the population as a function of the number of
generations. Note that stagnation had not occurred and, as a
result of the relatively small number of generations computed,
additional computation may have resulted in an improved
controller.

At completion, the final performance of the best controller in
the population indicates an average deviation from the target of
5 mmHg on the worst case patient. The best controller, given
in Table II, was subsequently tested on 5000 patients chosen by EP maximization over the patient parameter space, \( \Theta \). This is an active search for the most difficult patients for the controller to handle. A histogram on the resulting controller performance is given in Fig. 5. The patient that had the best closed loop performance in this test is depicted in Fig. 6. Note that 0.6\% of the patients “died” during the test simulations. All of the “deaths” in the test occurred because the controller drove the MAP below 50 mmHg. The uncontrolled trajectories of some of the patients that the best controller could not successfully regulate are given in Fig. 7. Given that the minimax search was truncated after only 50 outer generations and 20 inner generations, the histogram in Fig. 5 shows remarkably good performance.

VI. CONCLUSION

1) The Benefits of the Minimax-EP Approach: Minimax testing provides a rational and practical method for determining a host of neural controller parameters that otherwise must be chosen by hand. As the performance effect of various choices of these parameters is not intuitively clear, a technique such as minimax-EP for evaluating these choices can significantly enhance performance.

The minimax design criterion is useful in many difficult control design problems because it tests potential controllers against a wide range of plants and yields both the best minimax controller and information about which plants present the minimax controller with the greatest difficulty. Minimax design also forces the designer to think about and carefully specify the class of plants to be encountered, as well as the true performance measures needed to both penalize poor behavior and ignore unimportant changes in the trajectory. Both of these areas are difficult and deserve more attention than they typically receive in the design process.

Furthermore, the inner loop of minimax optimization must be done without human intervention. It therefore requires an intelligent optimization approach that is not prone to premature entrapment at local extrema. Evolutionary programming has proved to be very useful in this regard, with both neural and fuzzy controller design [16], [21], [23].

Equation (2) indicates that minimax controller design is inherently a constrained optimization. As seen in the example, the constraints are not always simply interval constraints on the parameters. Rather, they are often specified in terms of allowable trajectories on the controlled or uncontrolled plant. The optimization algorithm must permit an extremely wide range of expression for specifying constraints. Similar flexibility is needed in the expression of the performance index in order that it reflect the truly desired performance. EP is very flexible in both respects.

2) Avoiding Unnecessary Computation: The computational burden associated with the minimax criterion can be signifi-
controller with one unacceptable plant would then be preferred in relation to another with five such failures. The result is often an acceleration in the overall search, since even though all controllers might be unacceptable at a given point in the search, some would have better performance and therefore provide better prospects for breeding acceptable offspring.

3) Computationally Imposed Near-Optimality: It is necessary, for computational reasons, to keep the number of inner generations to a reasonably small number. This means that EP may not find the absolute worst plant prior to being terminated. We tested for this possibility by running a post-facto maximization run in which the controller was constrained to the “minimax” solution and the search was made over a large number of plants. The result was that the “minimax” controller gave “unacceptable” performance in 0.6% of the larger set of plants. While there is no proof that worse results could not occur in a different experiment, we are encouraged by the experiment because: a) the standard alternative in the design of medical controllers is much less reliable, as it involves far less exhaustive testing than the minimax method, and b) even the best human controllers (anesthesiologists) have problems with a very small number of patients. Finally, one can clearly trade off computer time for a reduction of the 0.6% “unacceptable” result. One can also study the plants causing problems in order to understand why the set of admissible controllers has trouble with such plants.

4) Expanding the Method to Incorporate a Suite of Test Conditions: The example given in this paper only tested the neural controller for a single experimental paradigm, that of bringing the plant output from an initial value of 100 to a final value of 70. While illustrative of the possibilities, this is by no means adequate to the task of designing and testing a controller that, for example, would then be implanted in a human. Additional work must be done to develop a suite of tests and enhance the cost function in (10) to incorporate a variety of cases. An example would be to define a suite of test paradigms and let $J$ be the average or the worst of them. An alternative would be to randomly choose a sample from the suite for each
evaluation of $J$ with the final score being the expectation of the samples. One must always be careful with averaging because totally unacceptable performance in one case may be ignored if one averages that case with a sufficiently large number of more benign results. Clearly, such enhancements can seriously impact the computational burden.

**APPENDIX**

**ACE/ASE and SLAYR:**

The Logic of Precoded Neural Controllers

**A. Associative Search Element (ASE)**

Barto _et al._ [20] suggested that control could be done with a state encoder plus a single neuron with teaching done by a critic element (see Fig. A1). Only the neuron would learn. The critic would watch the performance of the controlled system until a “failure” occurred, in which case, the neuron would change its weights because of the learning obtained from the fact that a failure had occurred. The definition of failure can be very flexible. Typically, it means failure to meet some control specification. At each step in the procedure, the current system state is located (i.e., decoded) on a coarse grid representing positions associated with non-failure. Failure is identified with leaving the grid, at which point learning occurs, the system is put back in the grid and the process is repeated. Each box in the grid has a weight that determines the control action to be applied in the event the system enters that box.

A crucial element of the algorithm is a “blame allocator” which, upon failure, assesses which weights are likely to have been responsible for the failure. Only those identified weights are changed in the event of a failure. The weight update algorithm is:

$$w_{ij} = w_{ij} + \text{failure} \cdot \text{eligibility}_{ij}$$  \hspace{1cm} (A1)

where

- $w_{ij}$ = the weight associated with box$_{ij}$,
- $\text{failure} = 1$ if a failure has occurred and 0 if not,
- $\text{eligibility}_{ij} = \delta \cdot \text{eligibility}_{ij} + (1 - \delta) \cdot x_{ij} \cdot y$,
- $\delta$ is a parameter to be chosen in the range $(0, 1)$,
- $y$ is the output of the neuron, and
- $x_{ij} = 1$ whenever the system is in box$_{ij}$.

The term eligibility derives from the idea that weights stored in boxes are eligible for blame or praise for current performance if the system has recently visited them. Note that eligibility and the $x$ variable are updated at each time step while the weights are only updated upon failure.

**B. Adaptive Critic Element (ACE)**

Barto _et al._ [20] found that, taken alone, the ASE suffered from the fact that it used punishment alone as a learning tool. Weights were changed in response to failure, but weights were not reinforced for successful strategies. Furthermore, failure meant leaving the grid (i.e., total failure such that the system needed to be restarted). ACE was proposed as a useful supplement. ACE involves adding a second, virtually identical element to the ASE strategy (Fig. A2). The purpose of the second module is to replace the actual critic input with a prediction of the critic input. This provides two capability enhancements, provided the prediction is accurate. First, the prediction of critic input gives continuously available indications of whether or not the system is tending toward failure. ASE alone receives no such information until a failure has occurred. One then updates the control neuron’s weights on the basis of a predicted failure or even a tendency toward failure, thereby providing learning without actual failure. Second, the ACE can provide both reward for good performance and punishment for failure.

**C. Cerulean Model Arithmetic Computer (CMAC)**

Miller [15] proposed a somewhat different scheme for robot control. Based on the work by Albus [19], Miller proposed a neural-like scheme with a very different learning strategy. Its goal was to learn the inverse kinematics of the robot to which it was applied. The result was able to control a robot arm in both a static positioning problem and a tracking problem. In both cases, the objective was to use end effector error information to specify joint angles and velocities without specifically including the inverse kinematics in the controller mathematics. This very creative teaching approach uses the actual robot as teacher. The controller’s real goal (from the learning perspective) is to learn the robot’s inverse kinematics and dynamics.

The CMAC uses a state location grid which then excites (i.e., points to) a set of memory locations whose contents
are summed to generate the control. Learning is done by adjusting, based on performance, the weights stored in the above memory locations. Because of the size of state spaces associated with multiple input/output systems, care must be taken to reduce the memory requirements of the controller. Miller's version of CMAC consists of a state decoder, a hashor, a neuron, and a trajectory planner. The trajectory planner is used to generate a target path to be attained by the controller. The neuron essentially generates a control output by summing its assigned weights. The state decoder is a more complex grid than the one used in ACE/ASE in that it uses an adjacency scheme yielding an $n$ digit location code for each state dimension, such that a unit change in the value of that state element results in a change in only one of the $n$ digits in the location code. This is intended to generalize results in the sense that small movements in state space only change a couple of the digits in the location code. This has two advantages. First, the controller generates smoother output signals. Second, if the system enters a state location not previously visited, the controller is guided by what has been already learned by visiting adjacent locations (since the respective location codes differ by only a few digits). The hashor compresses the memory range being addressed in order to reduce the memory requirements of the algorithm.

**D. SLAYR**

CMAC and ACE/ASE are members of a class of Single Layor Regulators (SLAYR) that have been proposed [16]–[18] as a potential solution to the adaptive control problem in highly complex and uncertain environments, such as those encountered in closed loop control of drug infusion. SLAYR algorithms have the structure given in Fig. A3. The algorithm has two components, a classifier and a layer of neurons. First, sensor information is processed by a classifier. This predecoded grid essentially discretizes the state space of the plant. Sensor information places the current plant somewhere in the grid. Grid location then turns on a subset of weights controlling a neuron or array of neurons. The output of the neuron layer is summed to determine the control value.

As discussed in [18], these algorithms have two important advantages. First, they have no hidden nodes and can learn in real time on the actual plant being controlled without difficult algorithms, such as back propagation, that are typically needed in more standard multilayer network architectures. This fast on-line learning capability makes them very agile and lessens the danger associated with a network trained off-line suddenly encountering unforeseen circumstances. Second, their architecture lends itself very well to exploitation of human experience and permits the design engineer and the neural net to cooperate nicely. The network uses the human's experience but learns those things the human does not know.

**ACKNOWLEDGMENT**

The authors wish to thank guest editor David Fogel and the anonymous reviewers for their detailed comments and suggestions.

**REFERENCES**

[22] A. V. Sebald, G. Schnurrer, M. Parti, N. T. Smith, and M. L. Quinn, "A dynamic empirical model of the human response to sodium nitropruss-


Jennifer Schlenzig is currently a Ph.D. student in Electrical Engineering at the University of California, San Diego. Her research interests include the application of dynamic neural networks to the areas of nonlinear control and image computing.

Anthony Sebald is an Associate Professor of Electrical Engineering at the University of California, San Diego. He received his Ph.D. in Electrical Engineering from the University of Illinois at Urbana. His current research interests include the application of fuzzy and neural techniques to the control of highly uncertain nonlinear plants. Current applications include a) the commissioning and intelligent operation of commercial buildings and b) closed-loop biomedical control problems.