# A HIGHLY FUNCTIONAL DECISION PARADIGM BASED ON NONLINEAR ADAPTIVE GENETIC ALGORITHM

Contract No. DAAH04-95-C-0045

Period of Performance: 06/01/95 to 05/31/96

***Final Report***

**Sponsor:**

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

**Technical Monitor:**

John Seluchins

**Contractor:**

Physical Optics Corporation
20600 Gramercy Place, Suite 103
Torrance, CA 90501



**Principal Investigator:**

Andrew Kostrzewski, Ph.D.
(310) 530-1416

**Key Personnel**

Jeongdal Kim, Ph.D.



**July 30, 1996**

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | July 30, 1996 | Final, 06/01/95 to 05/31/96 |

**4. TITLE AND SUBTITLE**
A Highly Functional Decision Paradigm Based on Nonlinear Adaptive Genetic Algorithm

**5. FUNDING NUMBERS**
DAAH04-95-C-0045

**6. AUTHOR(S)**
Jeongdal Kim

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Physical Optics Corporation
2545 West 237th Street, Suite B
Torrance, California 90505

**8. PERFORMING ORGANIZATION REPORT NUMBER**
3331

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
ARO 34084.1-EL-SBR

**11. SUPPLEMENTARY NOTES**
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
Approved for public release; distribution unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

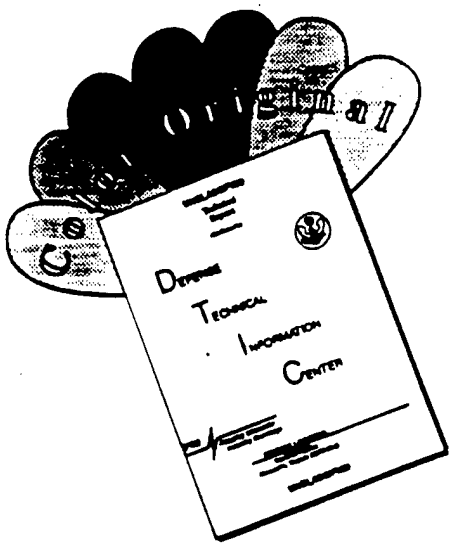
In the first year of this Phase II research program, POC refined the GA Optimizer by rewriting it in DLL format and optimizing the fuzzy logic-based GA control rules. We also evaluated existing neural network training methods using the GA, and examined areas in which our algorithm can improve their performance. POC has also initiated a new line of optimization development tools, the route optimizer. Initial development efforts demonstrate the proof of concept, and show that the algorithm can be applied to many optimization problems; these include making financial predictions, medical predictions, medical diagnoses, and market classifications, as well as modeling manufacturing processes and the anticipated resulting product quality, classifying biological organisms estimating job costs, detecting fraud, and many others. Finally, POC has begun developing the protocol of a development tool combining fuzzified genetic algorithm (GA) and a neural network (NN). This tool will find the optimal structure for the NN by training based on various combinations of the input data, and will optimize it by using NN performance as a fitness value.

In future research, POC will continue its efforts to develop the protocol of the GA/NN system.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Optimization, Fuzzified Genetic Algorithm, Neural Network | 25 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |

# DISCLAIMER NOTICE




THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

# 1.0 INTRODUCTION

Current genetic algorithm (GA) techniques are much faster than state-of-the-art non-GA convergence techniques (see Table 1-1), but faster still is POC's Fuzzy Logic Adaptive Genetic algorithm (FLAGA), which also has a user-friendly computer interface. POC's FLAGA is projected to have a convergence rate **an order of magnitude** faster than currently available GAs. This advantage becomes critical for large systems, which can have more than 10,000 degrees of freedom. Degrees of freedom define the **dimensionality** of a system, which consists of the space-time domain (x,y,z;t) and any number of other constraints that define a specific problem.

Table 1-1 Convergence Time Versus Size of the Problem. Comparison of Non-GA and GA Sorting Algorithms

| Searching Algorithm | GA | Size of the Problem, N | Basic Formula | Convergence Time | Convergence Time for t = 10 μs and N = 10,000 |
|---|---|---|---|---|---|
| Unordered Sequential | No | 10,000 | $O(N)$ | $t \cdot N$ (6) | 100 s |
| Ordered Sequential (1) | No | 10,000 | $O(\log_2 N)$ | $t \cdot \log_2 N$ | 13.29 s |
| GA (State-of-the-Art) (2) | Yes | 10,000 | $O(\log_m N)$ (4) | $t \cdot \log_m N$ | 4 s |
| GA (FLAGA) (3) | Yes | 10,000 | $O\left(\frac{\log_m N}{k}\right)$ (5) | $\frac{\tau \cdot \log_m N}{k}$ | 0.4 s |

Note: (1) Well-known binary search algorithm
(2) Standard GA with fixed evolutionary operation rates
(3) GA with fuzzy set control layer
(4) m is the size of the population in each generation (here m = 10).
(5) k is usually between 8 and10, depending on the complexity of the fitness function.
(6) t is unit search time.

In the first year of this program, POC investigated various ways to take advantage of FLAGA's high-dimensional search capability. With its parallel and global searching power, genetic algorithm can perform a powerful combinatorial search in problems where the search space is prohibitively large. As its first application, POC built a general purpose scheduler. Even though this problem looks simple enough for a human operator to solve, the combinatorial complexity can be daunting as the number of inputs increases. The second area considered is to apply genetic algorithm in another evolutionary computing method, neural networks. For example, in classification or financial prediction problems that use a neural network to train the data, a large number of variables can affect the output results, and the number of possible combinations among them is enormous. In that case, if the developer or the user must try all the possible combinations of input variables and train the data and check the results based on them, it will require several days or even weeks just to find out the set of input variables that affects the desired result. Needless to say, as the number of inputs grows, the number of possible combinations grows exponentially. If instead we use genetic algorithm to find the best-performing combination of input variables and use that combination for later NN training, we can avoid countless neural network realizations which would otherwise eventually be abandoned. Even if the problems otherwise appropriate for genetic

algorithm and for NNs are different (see Section 4.0), the use of genetic algorithm to facilitate the selection of data sets and NN architectures is a potentially fruitful area of investigation.

In this report, we first describe a GA scheduler; then the Neural network application of genetic algorithm will be outlined. The development of the software for the latter is currently under way. After the description of GA schedule and NN application, we summarize the progress of the initial period of this program.

## 2.0 BACKGROUND OF APPLICATION

As telecommunications mushroom, the amount of information that any given user handles increases accordingly. One of the most serious challenges now facing information workers is data overload. The information technology explosion has loaded data users with incredible volumes of data that mask, rather than expose, the useful information required to make timely, intelligent decisions. This obviously further complicates control processing units that must be remotely located for both security and cost reasons. This bottleneck adds further levels of complexity to large military system data management problems for Air Force, Navy, Army, and Marine Corps needs. **An example from Desert Storm is illustrative. It involves the transportation of troops/equipment over long distances with large numbers of degrees of freedom** such as: departure/arrival time schedules (for ships, connecting flights, troops, and military equipment), as well as many other highly diversified criteria. Here, for simplicity (but **without** loss of generality), we consider only the problem of transporting troops with a large number of connecting points. We will demonstrate that even in such a relatively simple case the number of **statistical problem realizations** is extremely large.

Given a sufficiently large number of degrees of freedom (>10,000), no existing optimization method is capable of solving the resulting problems. To address this issue, POC is working on a user-friendly parametric GA interface that uses **internal parametric control** to significantly increase the convergence rate: down to less than one second per 10,000 degrees of freedom.

## 3.0 TROOP AIR TRANSPORTATION DOUBLE-SORTING GLOBAL OPTIMIZATION PROBLEM

Consider a large-volume (high-dimensionality) military data management problem: *Troop Transportation Double-Sorting Global Optimization.* For problems such as this, the time needed to reach an optimal solution increases **logarithmically** with respect to the size of the problem (or, the number of degrees of freedom). In the FLAGA case, however, the logarithmic base **is reduced** step by step as shown in Figure 3-1. Therefore, in our case, convergence time is much less dependent on the size of the problem. This is because, in the case of POC's fuzzy-logic controlled GA, the **crossover, mutation**, and **reproduction** functions have **adaptable rates**, which are controlled by the **chromosome pool** first-order differential, a new **internal** parameter of the GA-system. POC's GA system is a **parametric one**, i.e., the *crossover*, *mutation*, and *reproduction* rates are controlled **internally** by the convergence process.

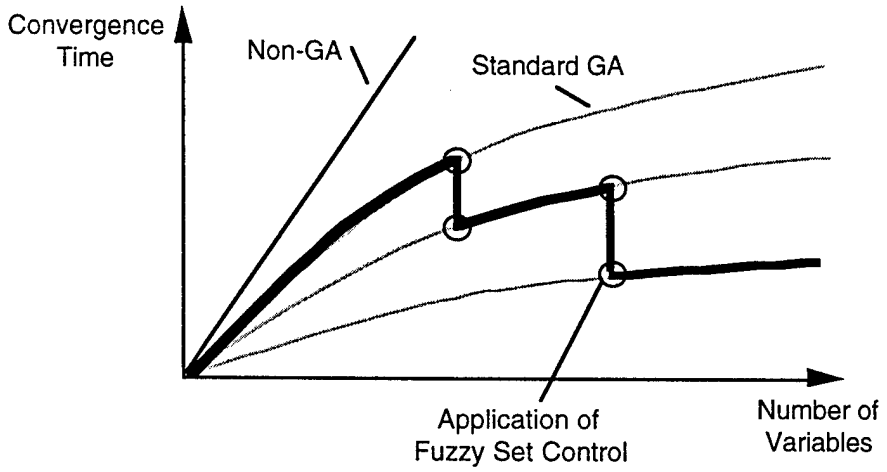



Figure 3-1
Comparison between standard optimization methods and POC's FLAGA method.

## 3.1 Description of the Problem

The *Troop Air Transportation (TAT) Double-Sorting Global Optimization* problem belongs to the general class of **scheduling** problems. Consider an illustration of the TAT problem, as shown in Figure 3-2.

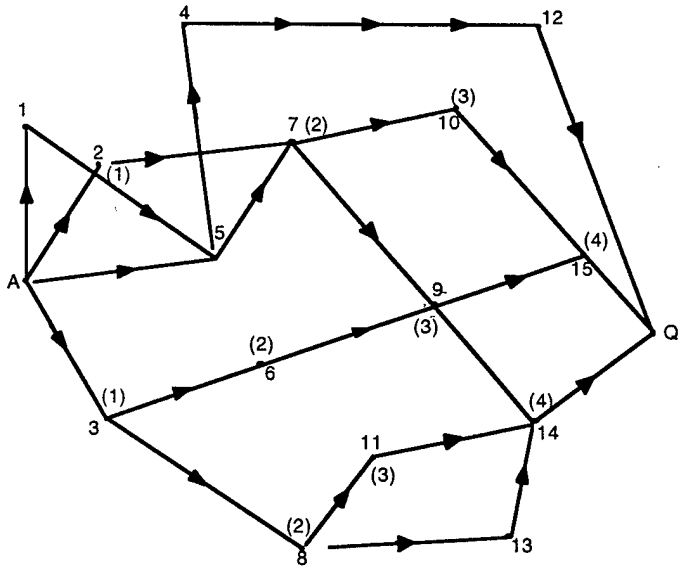



Figure 3-2
Illustration of the TAT through 17 airports, from Port A to Port Q.
Each port (except A,Q) is numbered by a digit 1 through 15. The numbers in brackets determine the specific flight, such as: A, (1), (2), (3), (4), Q, where four connection ports are always assumed. This is only a spatial coordinate map; i.e., the schedule is not shown.

At the start, the routing paths can be chosen almost arbitrarily with, let us say, a fixed number of connection points (such as A, 2, 7, 10, 15, Q, etc.). Each flight path is organized as a binary stream, or **gene**. For K number of planes, a single statistical realization is represented by K flight paths, so a **chromosome** is K-dimensional.

## 3.2 ANALYSIS OF THE PROBLEM

### 3.2.1 Assumptions

In order to demonstrate the feasibility of applying the FLAGA software for global optimization of troop transportation, we need first to generate the problem quantization, or *chromosome pool*, and to define the proper *cost (fitness)* function. We must make a number of simplifying assumptions, not for the purpose of problem simplification (since much more complicated problems can be solved by using the FLAGA), but rather for the purpose of explanation.

These are the simplifying assumptions:

(a) We transport only troops.
(b) We transport only by air (using airplanes).
(c) Although the military units are transported through a variety of airports (or *ports*), all units are transported from Port A to Port Q as shown in Figure 3-2. For simplicity, a constant number of four connection ports is always assumed.
(d) The military units transported are small enough that they are not partitioned during flight (e.g., platoons).

In spite of these simplified assumptions, a great deal of *flexibility* is still present in the program, in the form of such variables as numbers of units and troops, numbers of ports, planes, and their locations, and varying plane speeds. Also, very flexible *schedules* based on the "window" concept are assumed.

### 3.2.2 FLAGA Software Engine

The *Fuzzy Logic Adaptive Genetic algorithm (FLAGA)* can be applied to a broad variety of *global optimization* problems (so-called *NP-complete* problems [2]), assuming that the specific application problem will be transformed into FLAGA language. POC's FLAGA is faster than classical GA, and much faster than non-GA sorting methods [2].

### 3.2.3 Translation into FLAGA Language

Two steps are required in order to translate the problem into FLAGA language.

a) Quantization and Chromosome Space (Pool)
The space must be quantified and transformed into multi-dimensional *binary strings*. Each string represents a *gene*, and multi-dimensional strings represent *chromosomes*.

b) Cost Function

The *cost* (or fitness) function must be defined, including the *constraints* for global problem optimization. This function should be uniquely attached to a *chromosome space (pool).*

### 3.2.4 Notation, Summation Symbols

| | | |
|---|---|---|
| (i,j) | = | Military Units |
| (k, l) | = | Planes |
| (n,m) | = | Ports |
| q | = | Ports in sequence for specific flight (q=A,1,2,...,Q) |

### 3.2.5 Constants

| | | |
|---|---|---|
| K | = | Total number of planes |
| I | = | Total number of units |
| N | = | Total number of ports |
| M | = | Number of ports per flight |
| MRT | = | Maximum allowable flight time for a unit (including connections) |

### 3.2.6 Variables

| | | |
|---|---|---|
| $L_n$ | = | Location of port *n* |
| $L_i$ | = | Locations of units *i* |
| $L_{ik}$ | = | Location of unit *i* in plane *k* |
| $C_k$ | = | Capacity of plane *k* (i.e., maximum allowable number of troops) |
| $ED_{nm}$ | = | Euclidean distance between port locations *n* and *m* |
| $R_{ik}$ | = | Total flight time of unit *i* in plane *k* |
| $L_k$ | = | Location of plane *k* |
| $L_{kn}$ | = | Location of plane in port *n* |
| v'v" | = | Lowest/highest plane speeds (v"> v') |
| $\Delta t_{kn}$ | = | Uncertainty of plane *k* staying at connection port *n* |

### 3.2.7 Schedule and GA Windows

Each unit has a given schedule; that is, it needs to start from Port A within the *time window*:

$$(\tau_{iA'}, \tau_{iA"}); i = 1, 2,..., I \ , \qquad (A1)$$

where $\tau_{iA'}$ is the *earliest* time of departure and $T_{iA"}$ is the *latest* time of departure of units in *entrance* Port A. Also, the units should arrive at *exit* Port Q within the following time window:

$$(\tau_{iQ'}, \tau_{iQ"}); i = 1, 2,..., I \ . \qquad (A2)$$

Obviously, the given schedule should include the fact that time window (A2) is much wider than time window (A1), for two reasons:

a) Plane speed has an *uncertainty*, within the tolerance:

$$(v', v'') \tag{A3}$$

b) At any connection port, the plane stays some uncertain time $\Delta tkn$.

Therefore, the relation between (A2) and (A1) is:

$$\left(t_{iQ'}, t_{iQ''}\right) = \sum_{q=\lambda}^{Q} \left(t_{i',q-1},\ t_{i'',q-1}\right) + \left(\Delta t_{qi},\ \Delta t_{qi}\right) + \frac{ED_{q,q-1}}{v'},\ \frac{ED_{q,q-1}}{v''};\ v' > v'' \ , \tag{A4}$$

where the notation of relation under sum, is, in fact:

$$(\text{ARRIVAL, DEPARTURE}) \tag{A5}$$

It should be noted that *t*-times are *real-world* times, in that *t*-times define the constraints implemented into this data management system.

### 3.2.8 System Constraints

System constraints are included in the FLAGA system in the form of *cost function penalty terms*. The first constraint comes from the externally given time schedule, including Eq. (A4). This means that *the time of departure from entrance Port A should be within the departure schedule for all units*:

$$\tau_{Ai}' \leq t_{Ai} \leq \tau_{Ai}'' \tag{A6}$$

as well as the arrival time at exit Port Q, which should be within the arrival schedule for all units:

$$\tau_{Qi}' \leq t_{Qi} \leq \tau_{Qi}'' \ . \tag{A7}$$

The remaining constraints are:

$$\Sigma L_{ik} < C_k \ , \tag{A8}$$

where $L_{ik}$ is the number of troops for unit $i$ in plane $k$, and

$$\text{MAX } R_{ik} < \text{MRT} \tag{A9}$$

and,

$$\Sigma L_{kn} < Cn \ , \tag{A10}$$

where $L_{kn}$ is the number of planes in port *n at the same time*.

Therefore, constraint (A8) represents the capacity of a plane, while constraint (A10) represents the capacity of the port. Constraint (A9) derives from the fact that the flight (travel) time of the unit should not exceed some limit.

### 3.2.9 Double-Sorting Problem

The *Troop Transportation Cost Function* (TTCF or TRCF), defined below, operates at the second level of the double-sorting problem. The second sorting step, discussed below, assumes that all of the flight paths have been defined and preliminary troop adjustments have been made. At the first sorting step, not discussed here, we preliminarily optimize unit relocation, while plane path mapping is fixed. The double-sorting architecture is necessary because it is impossible to optimize both flight paths and allocation of units to planes at the same time.

### 3.2.10 Chromosome Pool for Troop Transportation Global Optimization Problem

The chromosome space (pool) is constructed separately from the travel path for each plane. The procedure is as follows:

(1) q-segmented numbering is introduced; e.g., the example travel path shown in Figure 1 is transferred into the corresponding q-sequence:

$$\begin{array}{lllll} (A, & 2, & 7, & 10, & \textbf{Regular Notation} \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow \quad \updownarrow & \\ (A, & 1, & 2, & 3, & \textbf{q-Sequence} \end{array} \qquad \text{(A11)}$$

(2) All travel paths' regular notations are organized in sequence according to arithmetic value; i.e., for two paths, as shown in Figure 3-2:

**(2, 7, 10, 15) and (3, 6, 9, 15)**

We construct two integers:

**271015 and 36915**

and, obviously, the larger integer is situated after the first one.

(3) A sequential number is uniquely attached to each integer. This route is shown on the basis of a simple example. Assume only four connector ports, and **three** ports used for each travel path. If these ports are numbered 1, 2, 3, and 4, then the following travel paths are possible:

| | | | |
|---|---|---|---|
| 123 | 124 | 234 | 134 |
| 132 | 142 | 243 | 143 |
| 213 | 214 | 324 | 341 |
| 231 | 241 | 342 | 314 |
| 312 | 412 | 412 | 413 |
| 321 | 421 | 421 | 431 |

(A12)

They are organized in sequence, such as:

| 123 | 124 | 132 | 134 | 142 | 143 | 213 |
|---|---|---|---|---|---|---|
| ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ |
| (1) | (2) | (3) | (4) | (5) | (6) | (7) |

(A13)

This is a 24-number sequence.

(4) The organized sequence is presented in the form of binary stream; e.g.,

$$22 = 0 \leftrightarrow 2^5 + 1 \leftrightarrow 2^4 + 0 \leftrightarrow 2^3 + 1 \leftrightarrow 2^2 + 1 \leftrightarrow 2^1 + 0 \leftrightarrow 2^0 = 0 + 16 + 0 + 4 + 2 + 0 = (010110) \quad \text{(A14)}$$

(5) Each flight path is organized as a binary stream, or *gene*. Therefore, for a single realization, which is represented by K flight paths (some of them identical), equivalent to K airplanes, we obtain K genes; so a *chromosome* is K-dimensional.

The maximum number of possible flight paths is:

$$\frac{(1/2)N!}{(N-P)!} \quad \text{(A15)}$$

where N and P are integers, n! = n(n-1) (n-2) ..., N is the number of ports, and P is the number of *connection* ports to be used for each flight. The (1/2) comes from the fact that we have regular flight. Since every flight path can be realized in a number of ways, as in Eq. (A15), and the number of possible paths is equal to the number of planes K, the total number of realizations (or number of degrees of freedom) is:

$$D = D(N, K; P) = \left[\frac{(1/2)N!}{(N-P)!}\right]^K . \quad \text{(A16)}$$

For example:

N=50, P=4, K=200; then expansion (A16) becomes:

$$D=(50,200;4)=\left[\frac{(1/2)N!}{46!}\right]^{200}=(0.5\ x\ 47\ x\ 48\ x\ 49\ x\ 50)^{200}=\left(2.76.10^{6}\right)^{200}=10^{1288} \quad \text{(A17)}$$

i.e., we obtain the extremely large number, $10^{1288}$, which no conventional sorting method can handle.

### 3.2.1.1 Cost Function for the Troop Transportation Double-Sorting Global Optimization Problem

The *Troop Transportation Cost Function* (TTCF, or $T^2CF$) is defined as follows:

$$\begin{aligned} T^2CF = {} & Wt\sum_{K}\sum_{q}\frac{EDnmk}{v''}+Wd_1\sum_{i}\left(MAX,[0,TA'_i-tA_i]\right) \\ & +W_{d''}\sum_{i}\left(MAX[0,tA_i-TA_{i''}]\right)+WA'\sum_{i}\left(MAX[0,TQ_{i'}-t_{Q_i}]\right) \\ & +W_{A''}\sum_{i}\left(MAX[0,tQ_i-TQ_{i''}]\right)+W_{A'}\left(MAX\left[0,\sum L_{ik}-C_k\right]\right) \\ & +W_R\left(MAX[0,R_{ik}-MRT]\right)+Wp\left(MAX\left[0,\sum L_{kn}-Cn\right]\right) \end{aligned} \quad \text{(A18)}$$

where the critical weighting factors are:

| | |
|---|---|
| Wt | Travel weighting factor |
| $W_{d'}$, $W_{d''}$ | Departure schedule penalty weighting factors |
| $W_{A'}$, $W_{A''}$ | Arrival schedule penalty weighting factors |
| $W_a$ | Penalty weighting factor for capacity of the plane |
| Wp | Penalty weighting factor for capacity of the port |
| $W_R$ | Penalty weighting factor for maximum allowable travel time of troops. |

These penalty factors should be adjusted according to the importance of a given constraint. For example, if a "not-too-late" arrival time is more critical than a "not-too-early" arrival time, then

$$W_{A''} > W_{A'} \ . \quad \text{(A19)}$$

The following term is a quadratic bracket:

$$MAX[0, H] \quad \text{(A20)}$$

which is either 0 if a given constraint is satisfied or H if this constraint is violated.

The chromosome space is now tested against the $T^2CF$ cost function value, and the small $T^2CF$ values are promoted via the FLAGA route.

The next section outlines a preliminary version of the transportation scheduler POC built.

## 3.3 GA Route Scheduler

To demonstrate the feasibility of using genetic algorithm for the transportation optimization problem, POC developed a scheduler that gives the shortest route among a set of cities. In this demonstration program, the Los Angeles area was mapped, and a number of cities in the area were made available for selection. The initial state of the program is given in Figure 3-3.

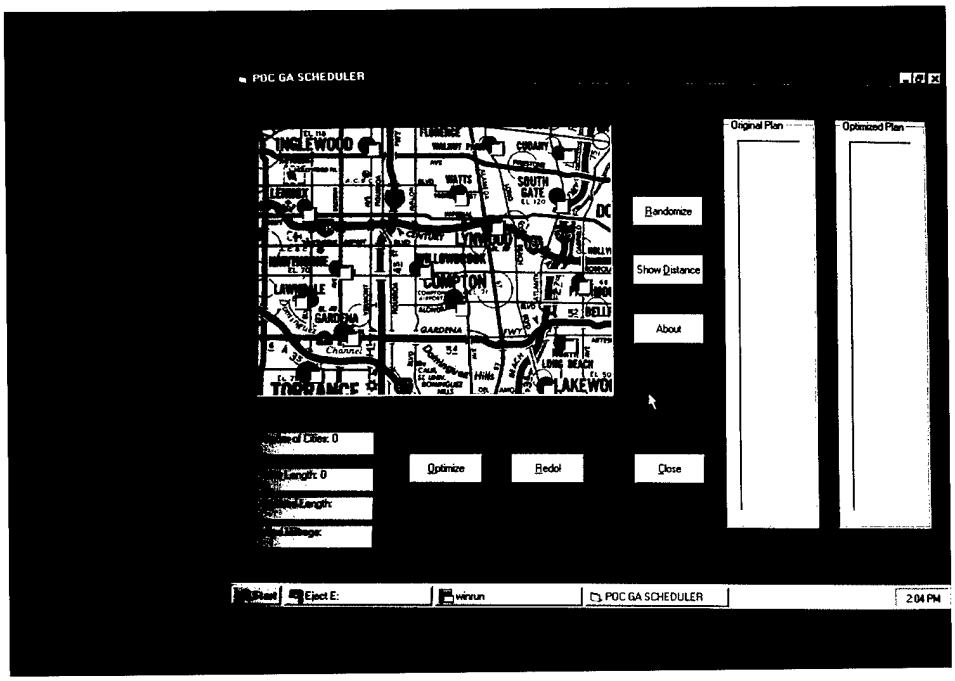



Figure 3-3
Initial state of route optimization problem.

In this program, a map is displayed with a number of checkboxes for selection. Each checkbox is associated with the city next to it, with a total of 16 cities in this demonstration program. The user can select the city he/she needs to include by checking the checkbox. There are also textboxes, for information purposes: the number of cities checked is displayed in the topmost textbox; the accumulated route length is shown in the second box; the optimized route length after optimization is given in the third box; and the mileage saving is displayed in the last box. At the right side of the screen, the original and optimized itineraries are displayed. Figure 3-4 shows eight cities selected.

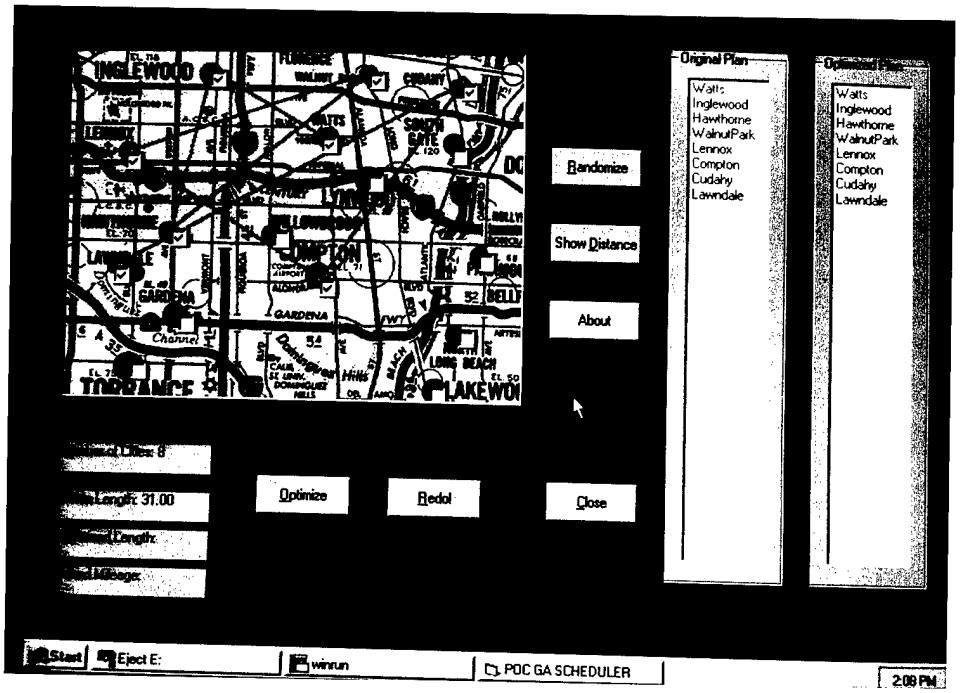
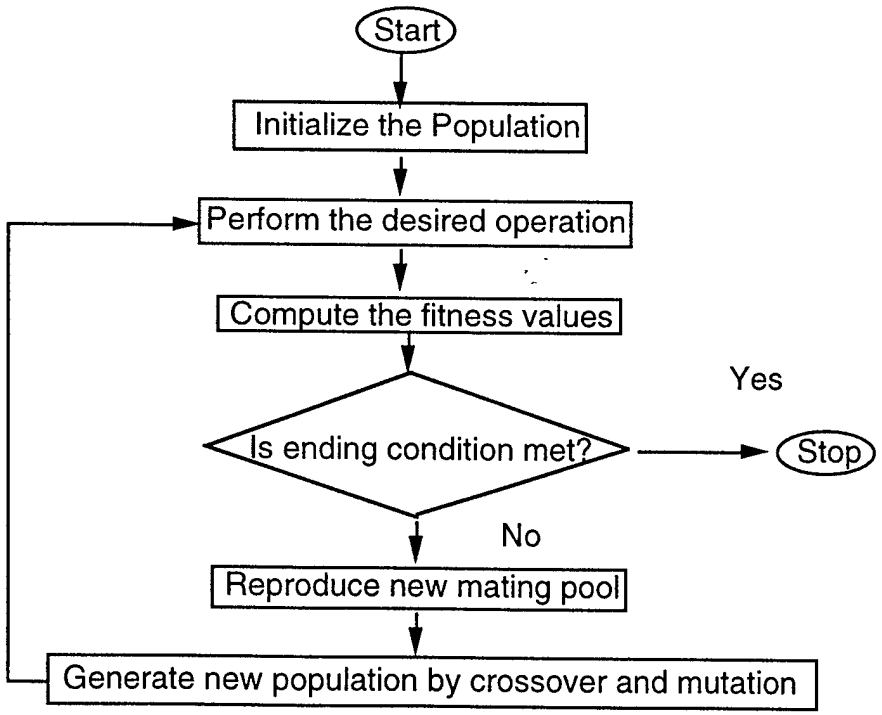



Figure 3-4
Input selection in the GA scheduler.

In this figure, the textboxes reflect the changed statistics. The route length of the randomly selected cities is shown along with corresponding names. The user now can click on the Optimize button and start optimization. Once the optimization process begins, it follows the algorithm below to the best result.



There are various ways of implementing ending conditions. This program compares the number of iterations since the last improvement was counted with a preset threshold value. When the ending condition is not satisfied, a new mating pool is created based on the performance of the previous population, and genetic operations such as crossover and mutation are performed to generate a new layer of population. The new population then goes back to the cycle. When the ending condition is met, the optimization process ends, and the program displays the result as in Figure 3-5.

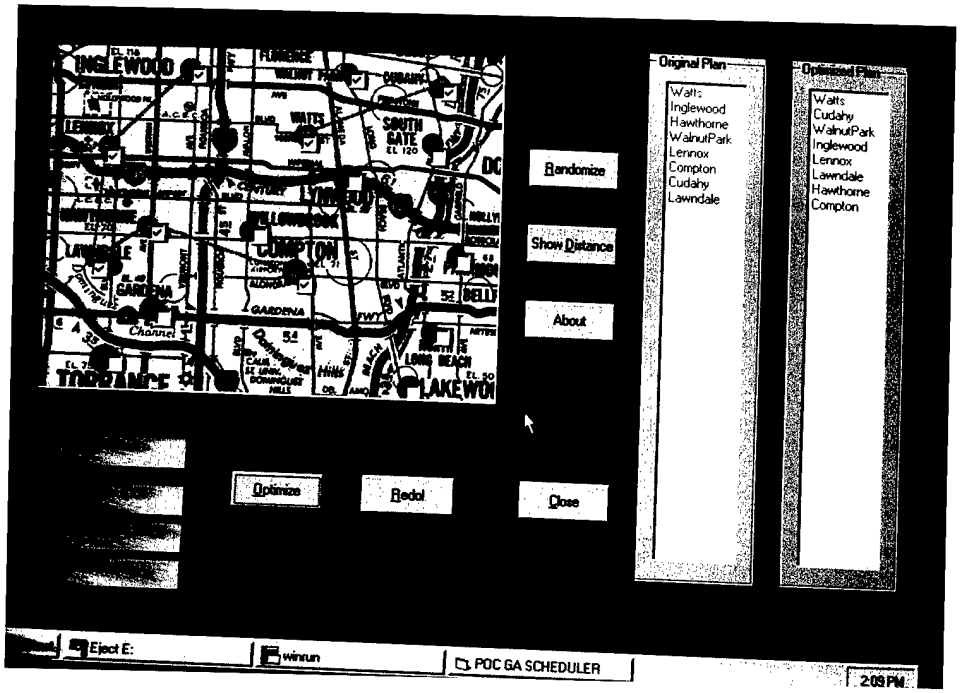



Figure 3-5
Final result of optimization.

In this figure, the optimized route is displayed with other relevant statistics. The lowest textbox shows the mileage saved by the optimization; the editor box on the right side displays the optimized itinerary. At this point, the user can start another round of optimization by clicking Redo, or end the program by selecting Close.

## 4.0 COMBINING TWO EVOLUTIONARY COMPUTING METHODS - GENETIC ALGORITHM AND NEURAL NETWORK

Genetic algorithms (GAs) and neural networks (NNs) are both modeled after natural biological systems; GAs imitate genetic evolution, and NNs mimic the brain. Each technique is suited to different types of problems. The GA is primarily a search mechanism, testing thousands of possible solutions to a problem, and evaluating the results. For the GA to be applicable, some sort of model or function must be available against which to evaluate the output of each set of inputs the GA tries. Since the appropriate function for evaluation is given, the GA is useful in finding the best mix, best order, or best grouping for achieving the best result. On the other hand, the neural network tries to make sense out of given inputs and outputs by building some kind of internal

model or function to connect both sides. This makes it a good candidate for pattern recognition and prediction applications where, given data, a prediction model must be built.

Given these differences, we can find cases where GAs can be used to enhance a neural network application. In the current research, we used GA to find the best input combination for the neural network training, and the optimal neural network architecture for any given problem.

## 4.1 EVOLVING NEURAL NETWORK

The powerful search capabilities of a genetic algorithm can be combined with the learning abilities of a neural network to search through data to find the set of variables that contribute to the most accurate models, saving a large amount of training time. POC is currently building software to automate much of the neural network design and development chores a developer used to do, usually by hand, using trial and error. Some of these tedious tasks include selecting testing/training data sets and determining which input variables to use. In our scheme, genetic algorithm will be used to evolve neural network structures and to select which input variables are keys to success. This evolving, learning, adapting artificial life capability is a powerful problem-solving paradigm that can be used to meet many real-world challenges.

This system emerged from the need to easily and quickly discover the best data elements and neural network architectures to build effective neural network applications. Previously, many hours were spent attempting to find the best networks manually. It is clear that an effective automation tool is needed to transfer these hours of effort onto computers; POC is applying GA for that purpose.

## 4.2 STRUCTURE OF THE SYSTEM

The system combines GA and NN in such a way that the former determines the optimal structure of the latter. Specifically, it performs the following tasks:

- Build and validate training and test data sets
- Create a population of candidate input variables and neural structures
- Build the neural networks
- Train them
- Evaluate them
- Select the best networks
- Pair up the genetic material representing the inputs and neural structure of these networks
- Exchange genetic material among them
- Put in a few mutations for a flavor of random search
- Go back into the training/testing cycle.

This continues for a defined number of cycles (generations), for a defined period of time, or until an accuracy goal is reached.

In this scheme, we represent the architecture of a network of N units by a *connection control matrix* C of dimension N×(N+1) (see Figure 4-1). The first N columns of matrix C represent the

connectivity relationship between units in the neural network, and the final (N+1) columns stand for the threshold bias for the unit. For example, in the figure below the unit that receives two inputs will have the threshold bias of 1, otherwise 0.

| | | Origin Node | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **bias** |
| Destination Node | **1** | 0 | 0 | 0 | 0 | 0 | 0 |
| | **2** | 0 | 0 | 0 | 0 | 0 | 0 |
| | **3** | 1 | 1 | 0 | 0 | 0 | 1 |
| | **4** | 1 | 1 | 0 | 0 | 0 | 1 |
| | **5** | 0 | 0 | 1 | 1 | 0 | 1 |

000000000000110001110001001101

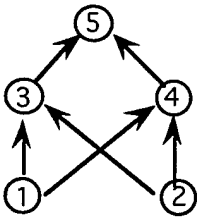



Figure 4-1
The conversion process from connection control matrix at top, to bit strings, center, to network architecture at the bottom

Each entry $C_{i,j}$ in the matrix C is a member of the *connection control set* S, and indicates the nature of the connection from unit to unit. Thus, column i of C represents the fan-out of connections from the unit i. Similarly, row j represents the fan-in of connections to unit j.

The bit string in the middle of Figure 4-1, created by concatenating successive rows of the matrix C, is the population to be processed by the genetic operators.

It should be noticed that many neural network architectures are implemented in a given generation of a population. The automation of neural network design in our scheme is made by two adaptive processes: genetic evolution through generations of network architecture spaces, and backpropagation learning in individual networks to evaluate the selected architectures. Thus, cycles of learning in individual architecture are embedded within cycles of evolution in populations. Each learning cycle presents an individual neural network with the set of input and output pairs defining the task. The backpropagation learning algorithm then compares the network's connection weights, so that it performs the input/output mapping task more accurately. Each evolution cycle processes one population of network designs according to their associated fitness values to yield an offspring population of more highly adapted network designs.

The backpropagation algorithm follows.

**Backpropagation Algorithm**

1. Initialize the weights to small random values.
2. Randomly choose an input pattern $\mathbf{x}^{(\mu)}$.
3. Propagate the signal forward through the network.
4. Compute $\delta_i^L$ in the output layer ($o_i = y_i^L$)

$$\delta_i^L = g'(h_i^L)[d_i^u - y_i^L],$$

where $h_i^l$ represents the net input to the $i$th unit in the $l$th layer, and $g'$ is the derivative of the activation function $g$.

5. Compute the deltas for the preceding layers by propagating the errors backward;

$$\delta_i^l = g'(h_i^l)\sum_i w_{ij}^{l+1}\delta_j^{l+1},$$

for

$$l = (L-1),....,1.$$

6. Update weights using

$$\Delta w_{ji}^l = \eta \delta_i^l y_j^{l-1} \quad .$$

7. Go to Step 2 and repeat for the next pattern until the error in the output layer is below a prespecified threshold or a maximum number of iterations is reached.

This process is clearly computationally intensive; but the effectiveness of the genetic algorithm combinatorial search capabilities is high. For example, a problem consisting of finding the best combination (subset) of 20 inputs and up to 15 hidden nodes in a backpropagation neural network is a combinatorial problem with over 16 million permutations. To train each network in a hard, full search would be a good project for a supercomputer. But with genetic algorithms, a very good solution very often appears in fewer than 1500 evaluations, which is 0.009% of the total possible configurations. Using some statistical data analysis, highly fit networks are often found in the first generation evaluated. This is clearly an efficient means for discovering effective network structure/input combinations.

It should be recognized that by the nature of genetic algorithms these networks are not necessarily optimal, but they do find good solutions.

## 4.3 PROSPECTIVE COMMERCIAL APPLICATION

The GA-assisted neural network, like most other leading neural network tools, can be used in a wide variety of applications, including financial prediction, medical diagnosis, market

classification, modeling of manufacturing processes, quality control, classification of biological organisms, job cost estimating, fraud detection, and many others. What will set POC's system apart is that its fuzzified GA will eliminate many of the chores of the ordinary neural network system.

## 5.0 SUMMARY OF RESEARCH FOR PREVIOUS REPORTING PERIOD

In the previous reporting period, POC has performed the following tasks:

1. Refined the Phase I design
2. Optimized the fuzzy logic-based GA control rules
3. Evaluated existing neural network training methods versus the genetic algorithm
4. Selected a multi-DSP board.

In addition, POC has made major progress in commercializing the optimization software package based on the GA algorithm. POC developed a software module that allows the Mathematica™ kernel to call a GA-based optimization routine. This package has been shipped to Wolfram Research, Inc. We first describe the Mathlink version of the GA Optimizer.

### 5.1 Mathlink Version of GA Optimizer

POC's commercialization efforts for the GA Optimizer have been fruitful: a mathematical library that links the optimization module to Mathematica was written and favorably received at Wolfram Research, Inc. A sample run of the GA module in Mathematica is shown in Figure 5-1.

```
link = Install ["e:\\pocsoft\\minimax"]
LinkObject [e:\pocsoft\minimax, 2, 2]

MiniMize2 ["sin(x1_ -x2*cos(x2)", -10.0, 10.0,
0.1,
                -10.0, 10.0, 0.1]
{-1.5625, -9.53125, -10.4772]

?MiniMize2
MiniMize2 [f_String, L1_Real, U1_Real, T1_Real, L2_Real,
   U2_Real, T2_Real minimum value of the 2-argument
   function f with sets of the lower bou bounds, and the
   tolerances:  Property of Physical Optics Corporation.

Uninstall[link]
e:\pocsoft\minimax
```

Figure 5-1
Sample run of GA Optimizer routine in Mathematica.

### 5.1.1 Description of Mathlink Version of GA Optimizer

To use the GA Optimizer in Mathematica, the user needs only to copy the executable file *minimax.exe* (which accompanied the previous report) to any designated directory; start Mathematica; and then connect to the file by giving the command shown in the first line in Figure 5-1:

```
link = Install ["e:\\pocsoft\\minimax"]
```

Here it is assumed that *"minimax.exe"* resides in the *"e:\pocsoft"* directory. The path can be modified as needed. Once the link is made, one can find the minimum value of the function with any number of arguments up to ten by the command, such as (the first-order function is used here for the purpose of illustration):

MiniMize1 ["second_order_function here", L1, U1, T1]
where L1: the lower bound of the argument
U1: the upper bound of the argument
T1: the tolerance

Note that the user will type the following command for the second-order function:

MiniMize2 ["second_order_function here", -10.0, 10.0, 0.1, -10.0, 10.0, 0.1]

The output is given in the form of a list such as:

{value_of_arg1_at_minimum,
value_of_arg2_at_minimum,
the minimum value}

One can find the maximum value of the function and the values of the arguments at that point in a similar way by calling the *maximize* function:

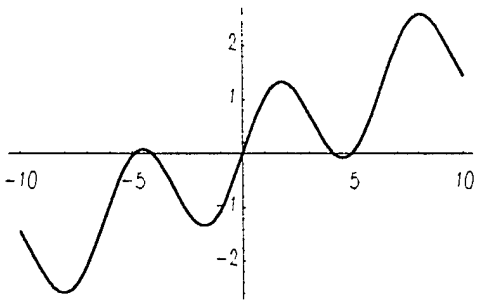
Maximize2 ["200 - (x1 ^ 2 + x1 - 11) ^2 - (x1 + x2 ^ 2 -7) ^2",
-6.0, 6.0, 0.1, -6.0, 6.0, 0.1]

Multiple searching sessions are possible. After the completion of the task, the connection is disconnected as follows:

Uninstall [link]

Figure 5-2 shows a plot of a function and its minimum value as found by one of the GA Optimizer's routines from within Mathematica.

```
In[6]:=
   Plot[Sin[x] + x/5, {x, -10, 10}]
```




```
Out[6]=
   -Graphics-
In[7]:=

   MiniMize1["sin(x1) + x1/5", -10.0, 10.0, 0.01]
Out[7]=
   {-8.05664, -2.59086}
```

Figure 5-2
Mathlink version of GA Optimizer at work.

### 5.1.2 Advantages of POC Optimizer Module

One of the major features of the genetic algorithm is its ability to find a globally optimal value without falling into local optima. While Mathematica can search for the global maxima and minima only for linear functions, the GA Optimizer can do so for virtually any type of function.

Furthermore, POC's Mathlink version of the GA Optimizer includes ten routines for finding maximum values: MaxiMize1,..., MaxiMize10. Mathematica has no corresponding functions. Figure 5-3 demonstrates the use of POC's GA Optimizer in finding maximum values for a modified form of Himmelblau's function.

```
  MaxiMize2 ["200 - (x1^2 + x2 - 11)^2 - (x1 + x2^2 -
7)^2",
                    -10.0, 10.0, 0.01, -10.0, 10.0, 0.01]

Out[26]=
   {-2.80273, 3.134765625, 199.999324689125}
In[27]:=

  MaxiMize2 ["200 - (x1^2 + x2 - 11)^2 - (x1 + x2^2 -
7)^2",
                     -10, 10.0, 0.01, -10.0, 10.0, 0.01]
Out[27]=
       {2.998046875, 2.001953125, 199.999870330066}

  MaxiMize2 {"200 - (x1^2 + x2 - 11)^2 - (x1 + x2^2 -
7)^2",
                       -10.0, 10.0, 0.01, -10.0, 10.0,
0.01]

Out[28]+
   {-3.7793, -3.28125, 199.999835462295}
```

Figure 5-3
Sample run of MaxiMize for Himmelblau's function.

As shown in this figure, we can run Optimizer more than once to find multiple sets of argument values, if there are any for the same optima. This version of the GA Optimizer has been shipped to Wolfram Research, Inc.

## 5.2 Refinement of the Software

The previous version of the GA program had many non-trivial limitations. First, the fact that it was DOS-oriented made it difficult to use. Second, the list of fitness functions was predetermined, restricting its practical applicability. Third, the dynamic range of each gene was limited to positive integer values. We have addressed these problems and revamped the whole software package to make it amenable to virtually any application that needs an optimization module.

### 5.2.1 Introduction of Windows DLL Module

Windows™ DLLs are very similar in concept to DOS libraries. Unlike static DOS libraries, however, DLLs (Dynamic Link Libraries) are linked to the main program at run-time so that the routines can be used by one or more programs. During compilation and linking, a program is informed of the presence of a DLL that contains the routines the program needs. After the program is loaded, these routines are dynamically linked. DLLs have significant advantages over static library routines, in that they can be simultaneously linked to multiple applications.

Another improvement being made in Phase II is the inclusion of a module that allows the user to input any desired fitness function. Considering that the fitness functions were predetermined in the previous version (five functions, including the banana function), this is a considerable advantage, making the software available for real-life application. Thus, the user can select one of the preset functions sorted in the data file, or type in any other fitness function as needed. This way, the user can put frequently used functions into a data file, but still can use the system for any other custom-made function.

The GUI was shown in the previous report.

### 5.2.2 Description of GA Optimizer

In entering a function, the arguments must be called x1, x2,..., up to a maximum x10. Algebraic notation is used to enter an expression involving the arguments. Parentheses may be used, as well as exponentiation ^. Built-in functions include abs, sin, if-else, and so on (see Table 5-1). If a string is entered incorrectly, the program will warn of a parsing failure.

Table 5-1 Built-in Functions in GA Optimizer

| | |
|---|---|
| abs | one argument |
| atan | one argument |
| $atan^2$ | two arguments |
| ceiling | one argument |
| cos | one argument |
| exp | one argument |
| floor | one argument |
| if else | three arguments x,y,z. Returns x,y,z |
| ln | one argument |
| log | one argument |
| pi | no arguments |
| round | one argument |
| sin | one argument |
| sqrt | one argument |

Once a function is selected or entered, the arguments are assumed to be bounded by 0 and 10. The user can modify the range as needed. The search for extrema will be carried out to a tolerance of $\pm 0.1$ on each argument, or the user can change the tolerance range as well. The bounds and tolerances are set by a dialog reached by clicking the Gene Ranges button; the current gene ranges (argument ranges) are displayed in red in the upper right panel.

The optimization is initiated by clicking the Optimize button. For each iteration, the iteration count is displayed, as well as information regarding the actions taken by the genetic algorithm in that iteration.

While the optimization is active, the Cancel button is shown. This button can be used to abort a lengthy optimization. The final results of a converged optimization process are shown in the lower

right panel. Multiple sessions can be initiated by File/New, even though only one problem can be optimized at a time.

The File/Number of Maximum Iterations function is selected in cases where the user wants to set a maximum number of iterations. Overall, POC's GA Optimizer has been designed to be easier to use, even for a user with minimal background in computing.

### 5.2.3 Optimization of the Fuzzy Logic-Based GA Control Rules

Optimization based on GA conducts a search from a population of points, rather than a single point. While this is a strength of a GA-based approach in the sense that it will not be trapped in a local optimal point, it requires a huge search space. This is where the fuzzy-logic decision module comes in. It is based on the simple reasoning that if the parent gene and its immediate offspring are sufficiently similar after reproduction, additional crossover operations are likely to help in finding a better solution. In this case, mutation is needed to try other avenues. The block diagram in Figure 5-4 illustrates this concept.

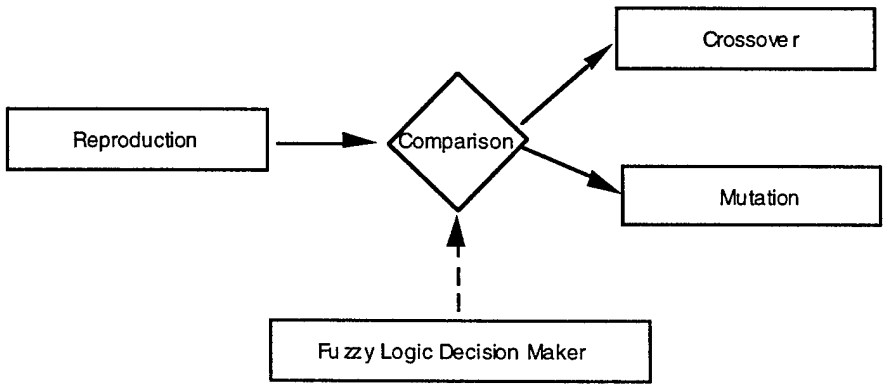



Figure 5-4
Integration of fuzzy logic in GA Optimizer.

To get a good result from the implementation of fuzzy logic, we need to find an optimal threshold value. The computer simulation indicates that 66% of the difference in bit values marks the turning point for the choice between crossover and mutation. Thus, when we set p as follows;

$$p = \frac{\text{Number of identical bits}}{\text{Total bit number of chromosome}} \tag{5-1}$$

we do the operations in the order shown in Table 5-2.

Table 5-2 Relationship Between Genetic Operation and Fuzzy Logic

| | |
|---|---|
| 0 2 p 2 0.66 | Reproduction crossover |
| 0.66 2 p 2 1 | Reproduction mutation |

By incorporating fuzzy logic into the GA, we can reduce the search space by 30%. The programming code for this is shown in Figure 5-5.

```
if (doFuzzyLogic) {
    if (dbit <= (int)(slength/fac)) {
inform (iterateMinimization: doing
crossover);
        crossover();
        crossover1();
        decigene();
        refitfun(f);
        remax1();
        remin();
        intpat();
    } else{
inform (iterateMinimization: doing
mutation                    );
        mutation();
        mutval();
        mutmin();
        mutminval();
        decigene();
        refitfun(f);
        remax1();
        remin();
        intpat();
    }
} else{
    crossover();
    crossover1();
    mutation();
    mutval();
    mutmin();
    mutminval();
    decigene();
    refitfun(f);
    remax1();
    remin();
    intpat();
}
```

Figure 5-5
Implementation of fuzzy logic in GA Optimizer.

## 5.3 Evaluation of the Existing Neural Network Training Methods Using Genetic Algorithm

One of the weaknesses of current neural network training methods is their long convergence time. That the genetic algorithm can alleviate this has been recently suggested in the literature [1-2]. Neural network training is basically a process of finding information that minimizes its difference from the target. It is thus apparent that the genetic algorithm can contribute to optimization in neural network training. Here, we evaluate the integration of GA with neural networks, as suggested by Adeli and Hung (1993). The first half of their algorithm can be summarized in the block diagram shown in Figure 5-6.

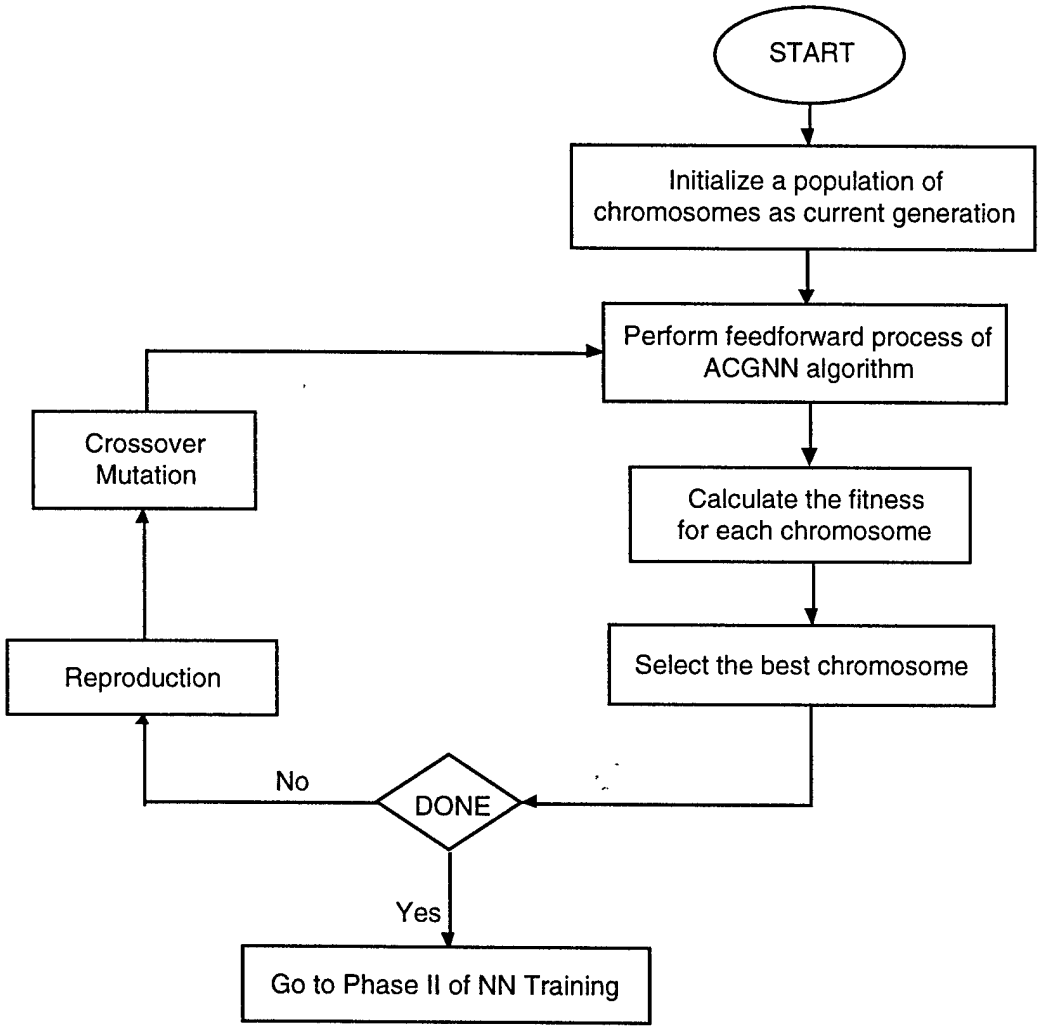



Figure 5-6
Integration of GA with neural network.

The best weight matrices produced through these procedures are fed into the second training stage, which employs the ACGNN training method. In short, optimizing the initial weight matrix permits NN training time to be used far more efficiently than it is under conventional methods.

There is, however, an important factor that must be considered here: the GA/ACGNN algorithm needs an extra stage of training -- optimization of the initial weight matrix. If this stage is not sufficiently optimized, the overhead of this extra stage may cancel out any gain achieved by using the algorithm. In this regard, POC's GA algorithm integrated with fuzzy logic can significantly reduce any overhead introduced by the GA module. The NN/ACGNN algorithm can be modified to achieve this as shown in Figure 5-7.

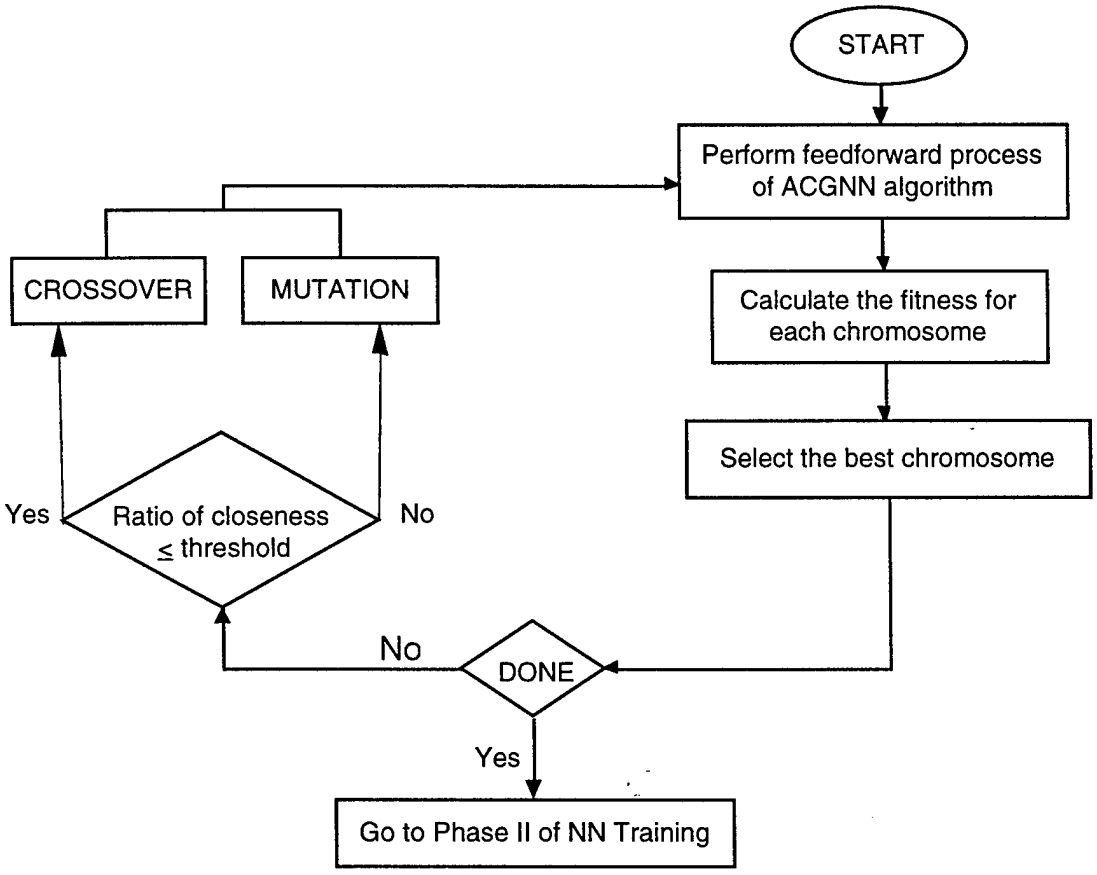



Figure 5-7
Modified GA/ACGNN algorithm.

## 6.0 CONCLUSIONS AND FUTURE WORK

In the first year of this Phase II research program, POC refined the GA Optimizer by rewriting it in DLL format and optimizing the fuzzy logic-based GA control rules. We also evaluated existing neural network training methods using the GA, and examined areas in which our algorithm can improve their performance. Since the GA can perform high-speed decision making, it can be

applied to optoelectric areas with huge advantages in bandwidth and accuracy. POC's commercialization effort has led to a Mathlink version of the GA Optimizer. It complements the shortcomings of existing Mathematica optimizing routines, and offers a fast and reliable way of finding global optima.

POC also initiated another line of optimization development tools, the route optimizer. Initial development efforts demonstrate the proof of concept, and show that the algorithm can be used in many optimization problems including financial predictions, medical diagnosis, market classification, modeling manufacturing processes and resulting product quality, classification of biological organisms, job cost estimating, fraud detection, and many others. Finally, POC began developing the protocol of a development tool combining fuzzified GA and a neural network. It will find the optimal structure for the neural network by training the various combinations of the input data, and will optimize it by using NN performance as a fitness value.

In future research, POC will continue its efforts in the development of the protocol of the GA/NN system, and will seek another application area of GA in an NN, such as neural network weight optimization.

## 7.0 REFERENCES

1. Adeli, H. and Hung, S.L. (1993). "Fuzzy Neural Network Learning Model for Image Recognition," Integrated Computer-Aided Engineering, pp. 43-55.
2. Hung, S.L., Adeli, H. (1994). "A Parallel Genetic/Neural Network Learning Algorithm for MIMD Shared Memory Machines," IEEE Transactions on Neural Network, Vol. 5, No. 6, pp. 900-909.
3. Booker, L. (1987). "Improving Search in Genetic Algorithm," in L. Davis (ed.) Genetic Algorithms and Simulated Annealing. London: Pitman Publishers.
4. Harp, S., Samad T., and Guha, A. (1989b). "Towards the Genetic Synthesis of Neural Networks," in J.D. Schaffer (ed.), Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann.
5. Harp, S., Samad T., and Guha, A. (1990). "Designing Application-Specific Neural Networks Using the Genetic Algorithm," Advances in Neural Information Processing Systems, 2.
6. Miller, G., Todd, P., and Hegde, S. (1989). "Designing Neural Networks Using Genetic Algorithms," in J.D. Schaffer (ed.), Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann.
7. Montana, D.J. and Davis, L. (1989). "Training Feedforward Neural Networks Using Genetic Algorithms," Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pp. 762-767.
8. Whitely, D. and Hanson, T. (1989). "Optimizing Neural Networks Using Faster More Accurate Genetic Search," in J.D. Schaffer (ed.), Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann.
9. Whitley, D. (1988). "Applying Genetic Algorithms to Neural Net Learning," Tech Report Number CS-88-128, Department of Computer Science, Colorado State University.