NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Genetic Algorithms Implementation

Graduation Project
COM – 400

Student: Rami Raba

Supervisor: Assoc. Prof. Dr Adnan Khashman

Nicocia - 2002

ACKNOWLEDGEMENTS

First of all I would like to ode this work to my family who have supported and sponsored me to reach this point of study. My dearest father, who encouraged and helped me to study in order to reach a higher degree of education. My sweat heart mother, you are the best woman and the best mother in the world.

Secondly, I have to thank my supervisor **Assoc. Prof. Dr Adnan Khashman** for his overwhelming and the limitless help he had shown.

Thirdly, I thank the Near East University in general and specially **Prof. Dr Şenol Bektaş.**

Here I have to dedicate this work to the Palestinian babies and children those are in danger and to those fallen as martyrs.

Finally, I would like to thank my best friends Ahmad Qatanany, Nashaat Ghuneim, Wisam AbuRajab and Mohammad Bader for their help.

" I dedicate my project to all those people mentioned above wishing that: God bless you all".

ABSTRACT

In the real life millions of species, each with its own unique and behavior patterns and characteristics, abound. All of these plants and creatures are evolving since millions of years having adapted themselves to a constantly shifting and changing environment in order to survive. The weaker species have to die in the sake of surviving the stronger, so the laws of natural selection and Darwinian evolution dedicate their lives, up on these ideas the genetic algorithms are based.

Genetic algorithms are examples of the latest computer applications, which are under consideration by the scientists all over the world. With the development of the neural networks and genetic algorithms, machines like human can be created.

Nowadays, implementing the genetic algorithm with the integration of neural network on machines, gives them the ability to think and decide like that ability of the human being. With the creation of genetically modified things one can hardly guess what future will be like.

TABLE OF CONTANTS

INTRODUCTION

Overview

The increasing prominence of computers has led to a new way of looking at the world. This view sees nature as a form of computation. By applying Genetic Algorithms to the computers to solve difficult problems, whose solutions require human intelligence. Together with the neural networks another interesting algorithms approach, inspired by the biological genetic behavior; genetic algorithm is being applied in complicated computer systems, along with the neural networks.

Research Objects

The aim of this project is to show and define the genetic algorithms implementation in order to use it wisely and have a look on its problems and their solutions.

Project Structure

Research Objectives include the following:

- In chapter one brief history and background of genetic algorithm, with a brief comparison between genetics and genetic algorithms. Then a discussion of the structure of the genetic algorithms.
- In chapter two brief history of neural networks along with biological terminology and their benefits and their structures. A brief discussion of their structures, topologies and their applications.
- In chapter three comparison between genetic algorithms and neural networks. Then developing neural networks using genetic algorithms. Discussions of using genetic algorithms in the networks. Talking about the structures of the networks based on genetic algorithms and artificial intelligence.
- In chapter four the presentation of the applications in the genetic algorithm. Dealing with examples related to applying the genetic algorithms, and discussing the methods of solving some problems by and applying and without applying the genetic algorithms.

- In chapter five the application of genetic algorithm in telecommunication system design will be discussed.

CHAPTER ONE

Genetic Algorithms Background

1.1 Overview

Genetics are a study of the function and behavior of genes. Genes are bits of biochemical instructions found inside the cells of every organism from bacteria to humans. Offspring receive a mixture of genetic information from both parents. This process contributes to the great variation of traits that we see in nature, such as the color of a flower's petals, the markings on a butterfly's wings, or such human behavioral traits as personality or musical talent. Geneticists seek to understand how the information encoded in genes is used and controlled by cells and how it is transmitted from one generation to the next. Geneticists also study how tiny variations in genes can disrupt an organism's development or cause disease. Increasingly, modern genetics involves genetic engineering, a technique used by scientists to manipulate genes. Genetic engineering has produced many advances in medicine and industry, but the potential for abuse of this technique has also presented society with many ethical and legal controversies.

Genetic algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. As you can guess, genetic algorithms are inspired by Darwin's theory about evolution. Simply said, solution to a problem solved by genetic algorithms is evolved.

1.2 Background and History of Genetic Algorithms

Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolving.

Algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness-the more suitable they are the more chances they have to reproduce.

At Genetic Algorithms we study nature's search algorithm of choice, genetics and evolution, as a practical approach to solving difficult problems on a computer. Genetic algorithms (G.As) and evolutionary computation have been around since the cybernetics movement of 1950s, but they have undergone a renaissance since the mid-1980sto the point where many walks of human endeavor are benefiting from this approach. For example, problems as different as jet engine design, electromagnetic antenna-absorber optimization and design, analog and logic electronic circuit synthesis, structural optimization and layout, factory and project scheduling, control system synthesis, music composition, image recognition, and automated programming have been successfully tackled.

The mechanics of a genetic algorithm (G.A.) are conceptually simple:

1. Maintain a population of solutions coded as artificial chromosomes.
2. Select the better solutions for recombination (crossover) of the mating chromosomes.
3. Perform mutation and other variation operators on the chromosomes.
4. Use these offspring to replace poorer solutions or to create a new generation altogether. Theory and empirical results demonstrate that G.As can be guaranteed to solve a broad class of provably hard problems, quickly, reliably, and accurately.

Rothenberg [1] introduced idea of evolutionary computing in the 1960s in his work "Evolution strategies" (Evolutions strategy in original). Other researchers then developed his idea. Genetic Algorithms (GAs) were invented by John Holland [2] and developed by him and his students and colleagues. This lead to Holland's book "Adoptions in Natural and Artificial Systems" published in 1975.

In 1992 John Koza [3] has used genetic algorithm to evolve programs to perform certain tasks. He called his method "genetic programming" (GP). LISP programs were used, because programs in this language can expressed in the form of a "parse tree", which is the object the GA works on.

Unlike natural evolution, genetic algorithms do not require millions of years to produce results. However, the system may need to run for many hours or even days. Large, complex problems require a fast computer in order to obtain good solutions in a reasonable amount of time. Mining of large datasets by genetic algorithms has only recently become practical due to the availability of affordable high-speed computers such as the DEC Alpha.

In the 1950s and 1960s several computer scientists independently studied evolutionary systems with the idea that evolution could be used as optimization tool for engineering problems.

The idea in all these systems was to evolve a population of candiate solution to a given problem, using operator inspired by natural genetic variation and natural selection.

Genetic algorithms were invented by John Holland [3] in 1960s and were developed by Holland and his students and colleagues at the University of Michigan in 1960s and 1970s. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomena of adaptation as it occurs in nature and to develop ways in which the mechanism of natural adaptation might be imported into computer system.

In the 1960s Rechenberg [1] (1965,1973) introduce "evolutionary strategies", a method he used to optimize real valued parameters for devices such as airfoils. This idea was further developed by Schwefel [4] (1675, 1977). The field of evolution of strategies has remained an active area of research, mostly developing independently from the field of genetic algorithms.

Several other people working in 1950s and the 1960s developed evolution inspired algorithms for optimization and machine learning. Box (1957), Friedman (1959), Bledsoe (1962), Bermermann (1962), and Reed, Toombs, and Baricelli [5] (1967) all worked in this area, though their work has been given little or none attention or follow up that evolution strategies, evolutionary programming, and genetic algorithms have seen.

In the last several years there has been widespread interaction researchers studying various evolutionary computation methods, and the boundaries between Genetic Algorithms, evolutionary strategies, evolutionary programming, and other evolutionary approaches have been broken down to some extent.

1.3 Biological Basis For Genetic Algorithms

The strings of an artificial genetic system are analogous to chromosomes in biological systems. The total package of strings is called a structure. These structures decode to form a particular parameter set, solution alternative, or point (in the solution space). Strings are

composed of features, or detectors, which take on different values. Features may be located on different positions on the string.

In natural systems, one or more chromosomes combine to form the total genetic prescription for the construction and operation of some organism. The total genetic package is called the genotype. The organism formed by the interaction of the total genetic package with its environment is called the phenotype. The chromosomes are composed to genes, which may take on some number of values called alleles; the position of a gene, its locus, is identified separately from the gene's function. Thus, we can talk of a particular gene, for example an animal's eye color gene, its locus, position 10, and its allele value, blue eyes. Product of the interaction of all genes.

"Physiological Team" in which a gene can make a maximum contribution to fitness by elaborating its chemical "gene product" in the needed quantity and at the appropriate stage of development.

Natural selection will tend to bring together those genes that constitute a balanced system.

After scientists became disillusioned with classical and neo-classical attempts at modeling intelligence, they looked in other directions. Two prominent fields arose, connectionism (neural networking, parallel processing) and evolutionary computing. It is the latter that this essay deals with - genetic algorithms and genetic programming.

1.4 The Technology of the Genetic Algorithms

The genetic algorithm is a powerful search algorithm based on the mechanism of natural selection and uses operations of reproduction, crossover, and mutation on a population of strings.

A set (population) of possible solutions in this case, a coding of the fuzzy rules of a fuzzy logic controller for the mobile robot can be represented as a string of typically binary numbers. New strings are produced every generation by the repetition of a two-step cycle. Firstly, each individual string is decoded and its ability to solve the problem is assessed. Each string is assigned a fitness value depending on how well it performed. Secondly, the fittest strings are preferentially chosen for recombination to form the next generation. Recombination involves the selection of two strings, the choice of a crossover point in the

string, and the switching of the segments to the right of this point between the two strings (the crossover operation).

Mutation is used to maintain genetic diversity within a small population of strings. There is a small probability that any bit in a string will be flipped from its present value to its opposite (for example, 0 to 1). This prevents certain bits becoming fixed at a specific value due to every string in the population having that value, often a cause of premature convergence to a non-optimal solution.

The automatic inclusion of the best performing string of the parent generation in the new offspring generation is built into the GA. This procedure prevents a good string being lost by the probabilistic nature of reproduction and speeds convergence to a good solution. A number of parameters influence GAs; they are the size of the population and the probability of applying the genetic operators (that is, crossover and mutations).

Looking at the world around us, we see a staggering diversity of life. Millions of species, each with its own unique behavior patterns and characteristics, abound. Yet, all of these plants and creatures have evolved, and continue evolving, over millions of years. They have adapted themselves to a constantly shifting and changing environment in order to survive. Those weaker members of a species tend to die away, leaving the stronger and fitter to mate, create offspring and ensure the continuing survival of the species. The laws of natural selection and Darwinian evolution dictate their lives. And it is upon these ideas that genetic algorithms are based.

What exactly do we mean by the term Genetic Algorithms?

- Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics.
- Genetic algorithms are software, procedures modeled after genetics and evolution.

GAs exploits the idea of the survival of the fittest and an interbreeding population to create a novel and innovative search strategy. The GA maintains a population of strings, representing solutions to a specified problem. The GA then iteratively creates new populations from the old by ranking the strings and interbreeding the fittest to create new strings, which are (hopefully) closer to the optimum solution to the problem at hand. So in each generation, the GA creates a set of strings from the bits and pieces of the previous

strings, occasionally adding random new data to keep the population from stagnating. The end result is a search strategy that is tailored for vast, complex, multimode search spaces.

GAs is a form of randomized search, in that the way in which strings are chosen and combined is a stochastic process. This is a radically different approach to the problem solving methods used by more traditional algorithms, which tend to be more deterministic in nature, such as the gradient methods used to find minima in graph theory.

The idea of survival of the fittest is of great importance to genetic algorithms. GAs use what is termed as a fitness function in order to select the fittest string that will be used to create new, and conceivably better, populations of strings. The fitness function takes a string and assigns a relative fitness value to the string. The method by which it does this and the nature of the fitness value does not matter. The only thing that the fitness function must do is to rank the strings in some way by producing the fitness value. These values are then used to select the fittest strings. The concept of a fitness function is, in fact, a particular instance of a more general AI concept, the objective function. Genetic

The population can be simply viewed as a collection of interacting creatures. As each generation of creatures comes and goes, the weaker ones tend to die away without producing children, while the stronger mate, combining attributes of both parents, to produce new, and perhaps unique children to continue the cycle. Occasionally, a mutation creeps into one of the creatures, diversifying the population even more. Remember that in nature, a diverse population within a species tends to allow the species to adapt to its environment with more ease. The same holds true for genetic algorithms.

1.5 Definition of Genetic Algorithms

The GENETIC ALGORITHM is a model of machine learning, which derives its behavior from a metaphor of the processes of EVOLUTION in nature. This is done by the creation within a machine of a POPULATION of Individuals represented by Chromosomes, in essence a set of character string that is analogous to the base-4 chromosomes that we see in our own DNA. The individuals in the population then go through a process of evolution.

We should note that EVOLUTION (in nature or anywhere else) is not a purposive or directed process.

That is, there is no evidence to support the assertion that the goal of evolution is to produce mankind. Indeed, the processes of nature seem to boil down to different individuals competing for resources in the ENVIROMENT. Some are better than others. Those are More likely to survive and propagate their genetic material

. In nature, we see that the encoding for our genetic information (GENOME) is done in a way that admits asexual REPRODUCTION (such as by budding) typically results in OFFSPRING that are genetically identical to the PARENT. Sexual REPRODUCTION allows the creations of genetically radically different that are still have the same general flavor (SPECIES).

At the molecular level what occurs (wild oversimplification alert) is that a pair of Chromosomes bumps into one another, exchange chunks? Of genetic information and drift apart. This is the RECOMBINATION operation, which GA/Gpers generally refer to as CROSSOVER because of the way that genetic material crosses over from one chromosome to another.

The CROSSOVER operation happens in an ENVIROMENT where the SELECTION of who gets to mate is a function of the FITNESS of the INDIVIDUAL, i.e. how good the individual is at competing in its environment. Some GENETIC Algorithms use a simple function of the fitness measure to select individuals (probabilistically) to undergo genetic options such as crossover asexual REPRODUCTION (the propagation of genetic material unaltered). This is fitness-proportionate selection. Other implementations use a model in which certain randomly selected individuals in a subgroup compete and the fittest is selected. This is called tournament selection and is the form of selection we see in nature when stages rut to vie for the privilege of mating with a herd of hinds. The two processes that most contribute to EVOLUTION are crossover and fitness based selection/reproduction.

As it turns out, there are mathematical proofs that indicate that the process of FITNESS proportionate REPRODUCTION is, in fact, near optimal in some senses.

MUTATION also plays a role in this process, although how important its role is continues to be a matter of debate (some refer to it as a background operator, while others view it as playing the dominate role in the evolutionary process).

It cannot be stressed too strongly that the GTENETIC ALGORITHN (as a SIMULATION of a genetic process) is not random search for a solution to a problem (highly fit INDIVIDUAL). The genetic algorithm uses stochastic processes. But the result is distinctly non-random (better than random).

In practice, therefore, we can implement this genetic model of computation by having arrays of bits or characters to represent the CHROMOSOMEs. Simple bit manipulation operations allow the implementation of CROSSOVER, MUTATION and other operations. Although a substantial amount of research has been performed on variable-length strings and other structures, the majority of work with GENETIC ALGORITHMs is focused on fixed-length character string. We should focus on both this aspect of fixed-lengthiness' and the need to encode the representation of the solution being sought as a character string, since these are crucial aspects that distinguish GENETIC PROGRAMMING, which does not have affixed length representation and there is typically no encoding of the problem.

When the GENETIC ALGORITHM is implemented it is usually done in a manner that involves the following cycle: Evaluate the FITNESS of all of the INDIVIDUALs in the POPULATION. Create a new population by performing operations such as CROSSOVER, fitness-proportionate REPREDUCTION and MUTATION on the individuals whose fitness has just been measured. Discard the old population and iterate using the new population.

One iteration of this loop is referred as a GENERATION. There is no theoretical reason for this as an implementation model. Indeed, we do not see this punctuated behavior in POPULATIONs in nature as a whole, but it is a convenient implementation model.

The first GENERATION (generation 0) of this process operates on a POPULATION of randomly generated INDIVIDUALs. From there on, the genetic operations, in concert with FITNESS measure, operate to improve the population.

1.6 SUMMARY

The genetic algorithm is considered as a model of machine learning, this is done by the creation within a machine of a population of individuals represented by chromosomes, in essence a set of character string that is analogous to the base-4 chromosomes that we see in our own DNA. The individuals in the population then go through a process of evolution.

Genes are bits of biochemical instruction found inside the cells of every organism from bacteria to humans.

Genetic algorithms are a part of evolutionary computing which is a rapidly growing area of the artificial intelligence.

Genetic algorithm offers solution to the problem solved by genetic algorithm is evolved.

Genetic engineering has produced many advances in medicine and industry, but the potential for abuse of this technique has also presented society with many ethical and legal controversies.

In chapter two neural networks will be presented.

CHAPTER TWO
NEURAL NETWORKS

2.1 Overview

Neural networks have a large appeal to many researchers due to their great closeness to the structure of the brain, a characteristic not shared by more traditional systems. In an analogy to the brain, an entity made up of interconnected neurons, neural networks are made up of interconnected processing elements called units, which respond in parallel to a set of input signals given to each. The unit is the equivalent of its brain counterpart, the neuron. A neural network consists of four main parts:

- Processing units, where each unit has certain activation level at any point in time.
- Weighted interconnections between the various processing units, which determine how the activation of one unit leads to input for another unit.
- An activation rule which acts on the set of input signals at a unit to produce a new output signal, or activation.
- Optionally, a learning rule that specifies how to adjust the weights for a given input/output pair.

2.2 Definition of Neural Networks

There are many definitions for a Neural Network:

- A massively parallel-distributed processor that has a natural propensity for experiential knowledge and making it available for use.
- A machine that is designed to model the way in which the brain performs a particular task or function of interest, the network is usually importanted using electronic components or simulated in software on digital computers.
- Are also referred to in the literature as neurocomputers, connectionist network, parallel-distributed processors, etc.

- Are different paradigms for computing:

  A-Von Neumann machines is based on the processing/memory abstraction of human information processing.

  B-Neural networks are based on the parallel architecture of animal brains.

- Are forms of multiprocessor computer system, with

  A) Simple processing elements.

  B) A high degree of interconnection.

  C) Simple scalar messages.

  D) Adaptive interaction between elements.

- A mathematical model composed of a large number of processing elements organization in to layers.

2.3 Why Neural Network Now?

Current technology has run into a lot of bottlenecks-sequential processing, for one. When a computer can handle information only one small piece at a time, there are limits to how fast you can push a lot of information through. Even with many processors working in a parallel, much time is wasted waiting for sequential operation to complete. It's also difficult to write programs that can use parallel operation effectively.

2.4 Benefits of Neural Networks

The use of neural networks offers the following useful properties and capabilities:

- **Nonlinearity.** A neuron is basically a nonlinear device. Consequently, a neural network, made up of an interconnection of neurons is itself nonlinear.
- **Input-output mapping.** Popular paradigm of learning called supervised learning involves the modification of the synaptic weights of a neural network by applying a set of labeled training samples or task examples.
- **Adaptively.** Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment.
- **Evidential response.** In the context of pattern classification, a neural network can be designed to provide information may be used to reject ambiguous pattern,

should they arise, and there by improve the classification performance of the network.

- **Contextual information.** Knowledge is represented by the very structure and activation state of a neural network.

2.5 Neural Networks Learning

Among the many interesting properties of a neural network, the property that is of primary signification is the ability of the network to learn from its environment, and to improve its performance through learning; the improvement in performance takes place over time in accordance with some prescribed measure. A neural network learns about its environment through an iterative process of adjustments applied to its synaptic weights and thresholds. Ideally, the network becomes more knowledgeable about its environment after each iteration of the learning process. Machine learning refers to computer models that improve their performance in signification ways based upon data.

2.5.1 Supervised learning

This is usually performed with feed forward nets where training patterns are composed of two parts, an input vector and an output vector, associated with the input and output nodes respectively. A training cycle consists of the following steps. An input vector is presented at the inputs together with a set of desired responses, one for each node, at the output layer. A forward pass is done and the errors or discrepancies, between the desired and actual response for each node in the output layer, are found. These are then used to determine weight changes in the net according to the prevailing rule. The term `supervised' originates from the fact that the desired signals on individual output nodes are provided by an external `teacher'. The best-known examples of this technique occur in the back propagation algorithm, the delta rule and Perceptron rule [6].

Supervised learning divided into two parts:

1) Feedback nets:

A) Back propagation through time

B) Real time recurrent learning

C) Recurrent extended kalman filter

2) Feed forward –only net: -

A) Perceptron

B) Adeline, Madeline

C) Time delay neural network

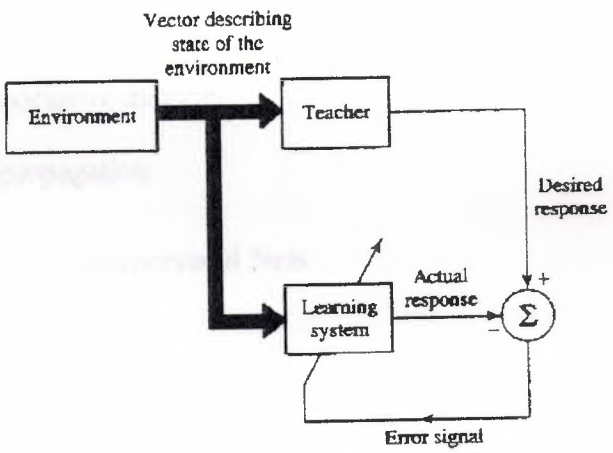

Figure 2.1 supervised learning

2.5.2 Unsupervised learning

This is usually found in the context of recurrent and competitive nets. There is no separation of the training set into input and output pairs. Typically a training cycle will consist of the following steps: a vector is applied to the visible nodes (or in the case of competitive learning, the input nodes); the net is allowed to reach equilibrium; weight changes are made according to some prescription. It is the amalgamation of input-output pairs, and hence the disappearance of the external supervisor providing target outputs, that

gives this scheme its name. This kind of learning is sometimes referred to as self-organization.

Unsupervised divided into two parts:

1) Feedback nets:

A) Discrete hop filed

B) Analog adaptive resonance theory

C) Additive gross berg

2) Feed forward –only nets

A) Learning matrix

B) Linear associative memory

C) Counter propagation

2.5.2.1 Applications for Unsupervised Nets

Clustering data:

Exactly one of a small number of output units comes on in response to an input.

Reducing the dimensionality of data:

Data with high dimension (a large number of input units) is compressed into a lower dimension (small number of output units). Although learning in these nets can be slow, running the trained net is very fast - even on a computer simulation of a neural net.

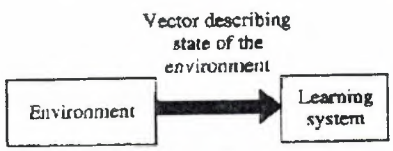

Figure 2.2 unsupervised learning

2.5.3 Hebbian learning

Hebb's postulate of learning is the oldest and most famous of all learning rules; it is named in honor of the neuropsychologist Hebb [7] (1949). Quoting from Hebb's book, the organization of behavior (1949,p.62):

When an axon of cell A is near enough to a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B is increased.

Hebb proposed this change as a basis of associative learning (at the cellular level), which would result in an enduring modification in the activity pattern of a spatially distributed "assemble of never cells."

2.6 Network Architectures

The manner in which the neurons of a neural network are structured is intimately linked with the algorithm used to train network. In general we may identify four different classes of network architectures:

2.6.1 Single-Layer Feed forward Networks

A layered neural network is a neurons organized in the form of layers. In the simplest form of a layered network, we just have an input layer of source nodes that project onto an output layer of neurons (computation nods), but not vice versa.

In other words, this network is strictly of a feed forward type. It is illustrated in **figure 2.3** for the case of four nodes in both the input layers. Such a network is called a single-layer network, with the designation "single layer" referring to the output layer of source nodes, because no computation is performed there. A linear associative memory is an example of a single-layer neural network.

In such an application, the network associates an output pattern (vector) with an input pattern (vector), and information is stored in the network by virtue of modification made to the synaptic weights of the network.

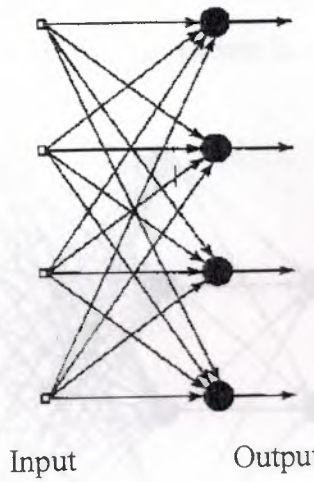

Figure 2.3 feed forward network with a single layer of neurons.

2.6.2 Multilayer Feed forward Networks

The second class of a feed ward neural network distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. The function of the hidden is to intervene between the external input and the network output. By adding one more hidden layer, the network is enabled to extract higher-order statistics, for (in a rather loose sense) the network acquires a global perspective despite is local connectivity by virtue of the extra set of synaptic connections and the extra dimension of neural interaction.

The ability of hidden neurons to extract higher-order statistics is particularly valuable when the size of the input layer is large. The source node in the input layer of the network supply respective elements of the activation pattern (input vector), which constitute the input signals applied to the neurons (computation nodes) in the second layer (i.e., the first hidden layer).

The output signals of the second layer are used as input to the third layer, and so on for the rest of the network. Typically, the neurons in each layer of the network have as their inputs the output signals of the preceding layer only.

The neural network of **Figure 2.4** is said to be fully connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. If, however, some of the communication links (synaptic connections) are missing from the

network, we say that the network is partially connected. A form of partially connected multilayer feed forward network of particular interest is a locally connected network.

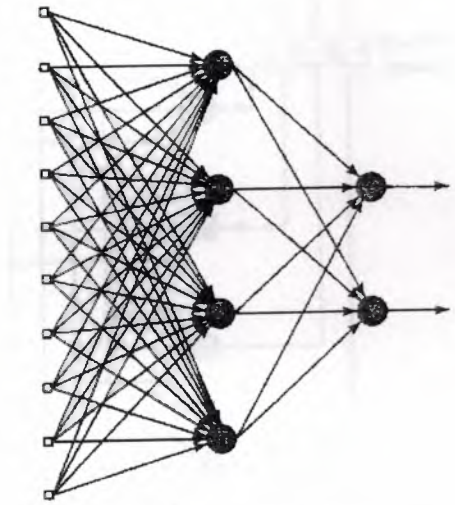

Figure 2.4 one hidden layer and output

2.6.3 Recurrent Networks

A recurrent neural network distinguishes itself from a feed forward neural network in that it has at least one feedback loop. For example, a recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons, as illustrated in the architectural graph of **Figure 2.5**

In the structure depicted in this figure there are no self-feedback loops in the network; self-feedback refers to a situation where the output of a neuron is fed back to its own input. The recurrent network illustrated in **Figure 2.5** also has no hidden neurons.

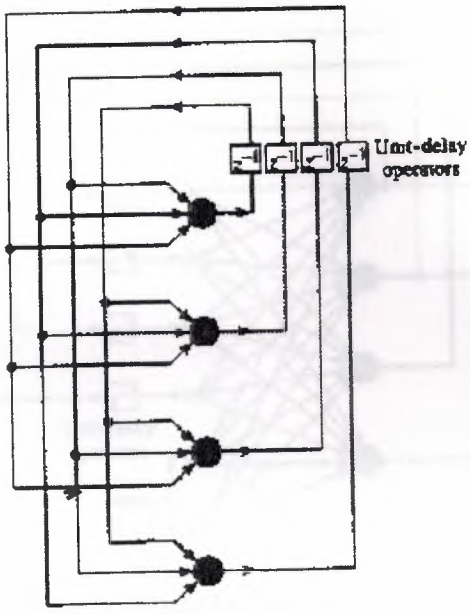

Figure 2.5 Recurrent network with no self-feedback loops and no hidden neurons

2.6.4 Lattice Structures

A lattice consists of a one-dimensional, two-dimensional, or higher-dimensional array of neurons with a corresponding set of source nodes that supply the input signals to the array; the dimension of the lattice refers to the number of dimensions of the space in which the graph lies.

The architectural graph of **Figure 2.6** depicts a one-dimensional lattice of 3 neurons fed from a layer of 3 source nodes. The hidden layer learns to recode (or to provide a representation for) the inputs. More than one hidden layer can be used.

The architecture is more powerful than single-layer networks: it can be show that any mapping can be learned, given two hidden layers (of units). The units are a little more complex than those in the original perception: their input/output graph is

As a function:

$Y = 1 / (1+ \exp (-k. (\&sum\ W_{in} * X_{in}))$

The graph shows the output for *k=0.5, 1, and 10,* as the activation varies from -10 to 10.

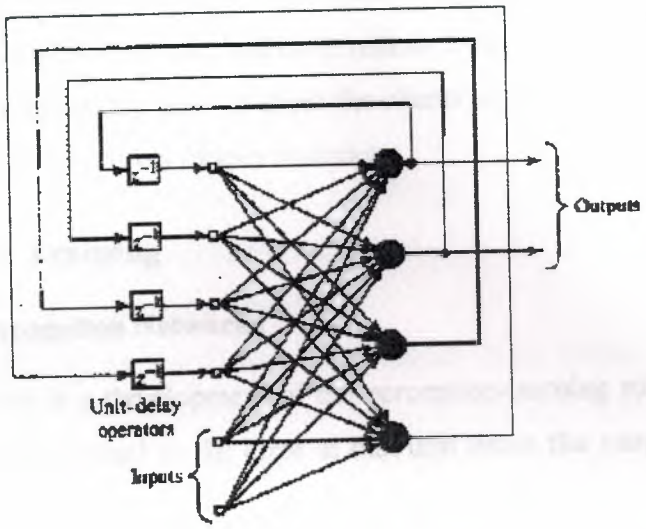

Figure 2.6 input layer of source nodes

2.6.5 Typical Radial Basis Function Architecture:

Like BP, RBF nets can learn arbitrary mappings: the primary difference is in the hidden layer. RBF hidden layer units have a receptive field, which has a *center:* that is, a particular input value at which they have a maximal output. Their output tails off as the input moves away from this point.

2.6.6 Neural Network Fields of Application

- **In investment analysis:** To attempt to predict the movement of stocks currencies etc., from previous data. There, they are replacing earlier simpler linear models.
- **In signature analysis:** As a mechanism for comparing signatures made (e.g. in a bank) with those stored. This is one of the first large-scale applications of neural networks in the U.S.A., and is also one of the first to use a neural network chip.
- **In process control:** There are clearly applications to be made here: most processes cannot be deterring computable algorithms. Newcastle University Chemical Engineering Department is working with industrial partners (such as Zeneca and BP) in this area.
- **In monitoring:** Networks have been used to monitor the state of aircraft engines. By monitoring vibration levels and sound, early warning of engine problems can be given.

- **In marketing:** Networks have been to improve marketing mailshots. One technique is to run a test mailshot, and look at the pattern of returns from this. The idea is to find a predictive mapping from the data known about the clients to how they have responded. This mapping is then used to direct further mailshots.

2.7 Neural Networks Training

2.7.1 Training Back Propagation Networks

The weight change rule is a development of the perception-learning rule. Weights are changed by an amount proportional to the error at that unit times **the output of the unit** feeding into the weight.

Running the network consists of: -

Forward pass:

The outputs are calculated and the error at the output units calculated.

Backward pass:

The output unit error is used to alter weights on the output units. Then the error at the hidden nodes is calculated (by back-propagating the error at the output units through the weights), and the weights on the hidden nodes altered using these values.

For each data pair to be learned a forward pass and backwards pass is performed.

This is repeated over and over again until the error is at a low enough level

2.7.2 Training Radial Basis Function Networks.

RBF networks are trained by

- Deciding on how many hidden units there should be
- Deciding on their centers and the sharpnesses (standard deviation)
- Training up the output layer.

Generally, the centers and SDs are decided on first by examining the vectors in the training data. The output layer weights are then trained using the Delta rule. BP is the most widely applied neural network technique. RBFs are gaining in popularity.

Nets can be

Trained on classification data (each output represents one class), and then used directly as classifiers of new data.

RBFs have the advantage that one can add extra units with centers near parts of the input, which are difficult to classify. Both BP and RBFs can also be used for processing time-varying data: one can consider a window on the data:

2.7.3 Back-Propagated Delta Rule Networks (BP)

Is a development from the simple Delta rule in which extra **hidden layers** (layers additional to the input and output layers, not connected externally) are added? The network topology is constrained to be feed forward: i.e. loop-free - generally connections are allowed from the input layer to the first (and possibly only) hidden layer; from the first hidden layer to the second... and from the last hidden layer to the output layer.

2.7.4 Kohonen clustering Algorithm:

In the winner-take-all competitive learning network, only connections to the winner are updated and updating of the weights dose not rely in any way on the spatial relations among the units in the competition layer. Therefore, as expected, the winner-take-all network cannot develop any spatial organization [8].

Kohonen demonstrated the formation of a topographic map of the first type by unsupervised self-organization. Unit in the output layer start off by responding randomly to the input signal. Kohonen identified two key mechanisms for a network to self-organization spatially.

1- locates the unit that best responds to the given input. This unit is called the winning unit.

2- Modify the connections to the winning unit and connections to units in its neigh borhood.

2.8 Building A Neural Network

Since 1958, when psychologist Frank Rosenblatt proposed the "Perceptron," a pattern recognition device with learning capabilities, the hierarchical neural network has been the most widely studied form of network structure. A hierarchical neural network is one that links multiple neurons together hierarchically. The special characteristic of this type of network is its simple dynamics. That is, when a signal is input into the input layer, it is propagated to the next layer by the interconnections between the neurons.

Simple processing is performed on this signal by the neurons of the receiving layer prior to its being propagated on to the next layer. This process is repeated until the signal reaches the output layer completing the processing process for that signal. The manner in which the various neurons in the intermediary (hidden) layers process the input signal will determine the kind of output signal it becomes (how it is transformed). As you can see, then, hierarchical network dynamics are determined by the weight and threshold parameters of each of their units. If input signals can be transformed to the proper output signals by adjusting these values (parameters), then hierarchical networks can be used effectively to perform information processing. Since it is difficult to accurately determine multiple parameter values, a learning method is employed. This involves creating a network that randomly determines parameter values. This network is then used to carry out input-to-output transformations for actual problems. The correct final parameters are obtained by properly modifying the parameters in accordance with the errors that the network makes in the process. Quite a few such learning methods have been proposed. Probably the most representative of these is the error back-propagation learning method proposed by D. E. Rumelhart [9] in 1986. This learning method has played a major role in the recent Neurocomputing boom.

The back-propagation paradigm has been tested in numerous applications including bond rating, mortgage application evaluation, protein structure determination, backgammon playing, and handwritten digit recognition. Choosing the right methodology, or back propagation algorithm, is another important consideration.

In working with the financial applications, many have found that the back-propagation algorithm can be very slow. Without using advanced learning techniques to speed the process up, it is hard to effectively apply back propagation to real-world problems. Over fitting of a neural network model is another area, which can cause beginners difficulty. Over fitting happens when an ANN model is trained on one set of data, and it learns that data too well.

This may cause the model to have poor generalization abilities - the model may instead give quite poor results for other sets of data. For an in-depth coverage of other neural

network models and their learning algorithms, please refer to the Technical Reading at the end of this User's Guide, the Technical Reference (sold separately), those papers listed in the Reference, or any other reference books on neural networks and relevant technology.

2.9 How Neural Networks Work?

They point out that neural nets can use extrapolation for classification problems or interpolation for computational problems. This paper specifically sets out to explain how a neural net can solve computational problems. The authors choose examples from chaos theory to demonstrate how neural nets can be used for predicting time series. Several examples of chaotic systems in physics and the natural sciences are provided. The authors cite the works of others that have demonstrated that two hidden layers are sufficient to solve most problems.

The authors use back propagation to train the neural net. The process of back propagation is explained thoroughly. The neural net takes the inputs and the outputs of the training data set and assigns weights to the connections. Next, the neural net calculates the sum of squared errors.

Iterations occur which adjust the weights of the connections in order to minimize the sum of squared errors. After this, the non-training data set inputs are entered into the neural net so that the neural net can calculate the predicted outputs. The authors used the Glass-Mackey equation to generate time series. They generated data sets with 500 observations. They then applied neural nets to make predictions for events. They used a neural net to make predictions at various future observations (t + P, where P is the number of periods ahead of observation t).

Although they found that the neural net because less accurate as the number of periods in the future increased, their results for the neural net were still an order of magnitude more accurate at increased intervals than the other methods they employed.

The authors demonstrate graphically why the neural net is successful for computational problems.

The neural net uses a squashing function that is flat then descends in a curve to a lower flat surface. The neural net then adds a squashing function that has the same absolute value but is of opposite sign to the original squashing function; this sum looks like a ridge. The next

step is to add the sum of two squashing functions that are perpendicular to the original summed squashing functions.

This results in a bump. The edges of the bump are not flat, however. So the neural net adds a bias term (or set of bias terms) in order to flatten the edges of the bump. The greatest height of the bump or lowest point in the bump will be the optimized solution to the problem.

2.10 SUMMARY

A neural network is a massively parallel-distributed processor that has a natural propensity for experiential knowledge and making it available for future use.

A neural network is applied in the situation where other methods cannot run because it uses the principle of the human brains.

The applications of the neural network are limitless and follow the new technology but these applications are limited and successfully used in the pattern recognition and development of the systems.

As we mentioned there are two types of learning algorithms, first supervised learning which includes the input vectors and output vectors and the law of operating (e.g. the perceptron, the back propagation algorithm, the Hopfield network, the hamming network), second unsupervised learning which depends on the figure of merit (e.g. Kohonen's self-organizing map, Competitive learning, Adaptive resonance theory).

The architecture of the neural network depended on the mission of the neural network and can be extended large or small included the Hidden layers that indicates the complexity of the neural network.

The reasons behind using the neural network:

Nonlinearity, Input-output mapping, adaptively, evidential response, contextual information.

A neural network learns about its environment through an iterative process of adjustment applied to its synaptic weights and thresholds.

The network becomes more knowledgeable about its environment after each iteration of the learning process. Machine learning refers to computer models that improve their performance in the signification ways based up on data.

In chapter three the genetic algorithm implementation in neural network will be presented.

CHAPTER THREE

COMPARISON BETWEEN NEURAL NETWORKS AND GENETIC ALGORITHMS

3.1 Overview

Structured Genetic Algorithms were developed by Dasgupta and McGregor and have proved to be a successful method to simultaneously optimize the neural network architecture and its weights using hierarchically structured chromosomes. The recombination phase is the same as in the standard genetic algorithm. During evaluation however 'high-level' genes act as switches to activate or deactivate lower level genes. In two leveled chromosomes were used. The top level defines the connectivity of the network, the bottom level the values of the weights and biases. The connectivity matrix represents the network connectivity with that part of the chromosome treated as a binary string.

In most of the Genetic Algorithm approaches where both network structure and weights are subject to genetic operations, the chromosome representing both parts is thought of as a long binary string and is subject to the genetic operators of the algorithm. As far as the genetic operators are concerned there is no distinction between the structural and the parametric (weight) part. This distinction is only made when the chromosome is translated into the actual neural network.

It is possible to make a distinction between the structural and the weight part of the chromosome. When different coding (i.e. binary and real-valued) is used for the two parts, this distinction must be made and non-homogeneous chromosomes are needed. In the best performance of the Genetic Algorithm was observed when the weights and biases were coded as real-valued genes, as opposed to the binary coded structural part of the chromosome. Genetic operations like crossover and mutation can now be thought of as being either structural or parametric changes to the network depending on what part of the

chromosome they operate on. When the changes are structural, the resulting offspring can inherit the set of weights from its parent.

This process is called 'weight transmission' and will be described in the next section. A structural crossover is illustrated.

3.2 Developing Neural Network using Genetic Algorithm

A neural network is biologically motivated approach to machine learning, inspired by ideas from neuroscience. Recently some efforts have been made to use genetic algorithms to evolve aspects of neural networks.

Neural network in its simplest "feed forward" form is shown in **Figure 3.1**, a neural network is a collection of connected activatable units ("neurons") in which the connections are weighted, usually with real-valued weights. The network is presented with an activation pattern on its input units, such a set of numbers representing features of an image to be classified (e.g., the pixels in an image of a handwritten letter of the alphabet). Activation spreads in a forward direction from the input units through one or more layers of middle ("hidden") units to the output units over the weighted connections. Typically, the activation coming into a unit from other units is multiplied by the weights on the links over which it spreads, and then is added together with other incoming activation. The result is typically threshold (i.e., the unit "turns on" if the resulting activation is above that unit's threshold). This process is meant to roughly mimic the way activation spreads through networks of neurons in the brain. In a feed forward network, activation spreads only in a forward direction, from the input layer through the hidden layers to the output layer. Many people have also experimented with "recurrent" networks, in which there are feedback connections as well as feed forward connections between layers.

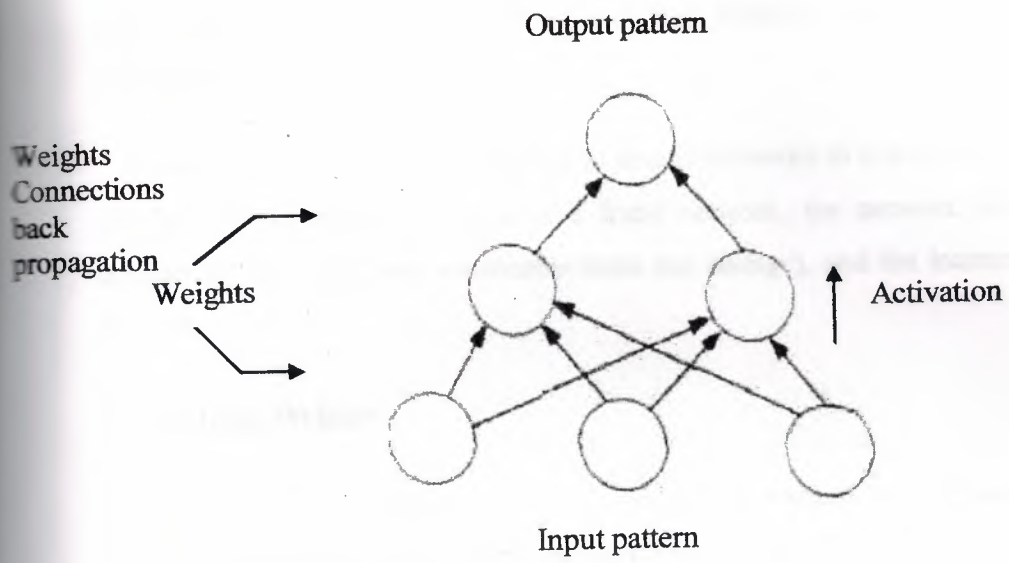

Figure 3.1. A schematic diagram of a simple feed forward neural network and the back-propagation process by which weight values are adjusted.

Activation spreads through a feed forward network; the resulting activation pattern on the output units encodes the networks 'answer" to the input (e.g., a classification of the input pattern as the letter A). In most applications, the network learns a correct mapping between input and output patterns via a learning algorithm. Typically the weights are initially set to small random values. Then a set of training inputs is presented sequentially to the network. In the back-propagation learning procedure (Rumelhart, Hinton, and Williams 1986), after each input has propagated through the network and an output has been produced, a "teacher" compares the activation value at each output unit with the correct values, and the weights in the network are adjusted in order to reduce the difference between the network's output and the correct output. Each iteration of this procedure is called a "training cycle," and a complete pass of training cycles through the set of training inputs is called a "training epoch." (Typically many training epochs are needed for a network to learn to successfully classify a given set of training inputs.) This type of procedure is known as "supervised learning," since a teacher supervises the learning by providing correct output values to guide the learning process.

In "unsupervised learning" there is no teacher, and the learning system must learn on its own using less detailed (and sometimes less reliable) environmental feedback on its performance.

Genetic Algorithms can be applied to neural networks in many ways. Some aspects that can be evolved are the weights in a fixed network, the network architecture (i.e., the number of units and their interconnections can change), and the learning rule used by the network.

3.3 Evolving Weights

This is the most common use of a genetic algorithm in conjunction with a neural network. Since genetic algorithms are excellent at searching a state-space, searching neural network weights is an ideal application.

Simply set up the genetic algorithm to evolve a string of floating point numbers (within the range that you specify) that can be used as weights for the network. The biggest trouble with using a GA is specifying the range of the weights. Since we generally don't know the range we have to estimate, and use a trail and error method to correct/optimize them. Generally network weights should not be too big - for example, the XOR net weights don't get larger in the absolute than 3.

Another problem with genetic algorithms and neural networks is figuring out an appropriate method of reproducing and crossing over the weights. It all depends on how you have set your weights up. Going back to our example with the XOR net, our network is small and the weights are easily represented by a 3x3 array:

For this kind of set up, I swap over groups of weights And, I select two population examples, one from the lower error (LE) half, and another from the higher error (HE) half. I then give the HE the LE weights for the final layer (the blue weights in the above diagram), and put it back in the population.

Yet if your neural network is a lot more complicated, then your representation of the weights will not be as simple. However you decide to do it, it is recommended that you keep weights grouped together; otherwise the resultant weights are as a good as random.

Mutation is also a genetic operator you should consider. In the example program, there is a rather high (10%) chance of mutation, and then the weights are altered by anything between -1 and 1.

3.3.1 When to Use

Genetic algorithms are an option, but they are not by any stretch always the best option. The genetic algorithm is *a lot* slower than back-propagation when applied to the XOR problem. The GA gives better results than the BP example used:

The genetic algorithm finds more accurate results, but the back-propagation is close to instantaneous, whereas the genetic algorithm will take anything between 5-20 seconds (233 Mhz test computer).

3.3.2 Architecture

Since the overall architecture of the network is imperative the operation, a lot of research has focused on using evolutionary techniques to evolve the best architecture (much like the evolution of our own brains). One simple method is to use a Boolean NxN matrix, where N is the number of neurons in the network. Any given place on the matrix refers to a connection between neuron X and neuron Y. For example, for the XORNet: This is a very simple method, and gets inefficient for the large networks that architectural optimization is often applied to.

3.3.3 Developing Weights in a Fixed Network

David Montana and Lawrence Davis [10] *took* the first approach evolving the weights in a fixed network. That is, Montana and Davis were using the Genetic Algorithm *instead* of back-propagation as a way of finding a good set of weights for a fixed set of connections. Several problems associated with the back-propagation algorithm (e.g., the tendency to get

stuck at local optima in weight space, or the unavailability of a "teacher" to supervise learning in some tasks) often make it desirable to find alternative weight-training schemes.

These scientists were interested in using neural networks to classify underwater sonic "lofargrams" (similar to spectrograms) into two classes: "interesting" and "not interesting." The overall goal was to "detect and reason about interesting signals in the midst of the wide variety of acoustic noise and interference which exist in the ocean." The networks were to be trained from a database containing lofargrams and classifications made by experts as to whether or not a given lofargram is "interesting."

Each network had four input units, representing four parameters used by an expert system that performed the same classification.

Each network had one output unit and two layers of hidden units (the first with seven units and the second with ten units). The networks were fully connected feed forward networks that are; each unit was connected to every unit in the next higher layer. In total there were 108 weighted connections between units. In addition, there were 18 weighted connections between the non-input units and a "threshold unit" whose outgoing links implemented the threshold for each of the non-input units, for a total of 126 weights to evolve.In genetic algorithm, each chromosome was a list (or "vector") of 126 weights. Figure *3.2* shows (for a much smaller network) how the encoding was done: the weights were read off the network in a fixed order (from left to right and from top to bottom) andplaced in a list. Notice that each "gene" in the chromosome is a real number rather than a bit. To calculate the fitness of a given chromosome, the weights in the chromosome were assigned to the links in the corresponding network, the network was run on the training set (here 236 examples from the database of lofargrams), and the sum of the squares of the errors (collected over all the training cycles) was returned. Here, an "error" was the difference between the desired output activation value and the actual output activation value. Low error meant high fitness.

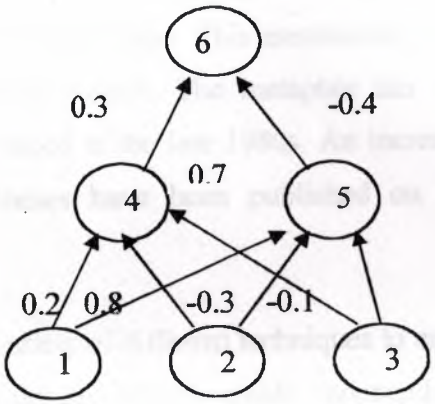

Figure 3.2 Illustration of Montana and Davis's encoding

Figure 3.2 Illustration of Montana and Davis's encoding of network weights into a list that serves as a chromosome for the Genetic Algorithm. The units in the network are numbered for later reference. The real-valued numbers on the links are the weights.

3.4 Developing Network Architectures

Montana and Davis's Genetic Algorithm evolved the weights in a fixed network. As in most neural network applications, the architecture of the network—the number of units and their interconnections is decided ahead of time by the programmer by guesswork, often aided by some heuristics (e.g., "more hidden units are required for more difficult problems") and by trial and error. Neural network researchers know all too well that the particular architecture chosen can determine the success or failure of the application, so they would like very much to be able to automatically optimize the procedure of designing an architecture for a particular application. Many believe that Genetic Algorithms are well suited for this task. There have been several efforts along these lines, most of which fall into one of two categories: direct encoding and grammatical encoding. Under direct encoding, network architecture is directly encoded into a Genetic Algorithm chromosome. Under grammatical encoding, the Genetic Algorithm does not evolve network architectures; rather, it evolves grammars that can be used to develop network architectures.

3.5 Evolving Artificial Neural Networks by Genetic Algorithms

The idea of evolving artificial neural networks by genetic algorithms is based on a powerful metaphor: the evolution of the human brain. This mechanism has developed the highest form of intelligence known from scratch. The metaphor has inspired a great deal of research activities that can be traced to the late 1980s. An increasing amount of research reports, journal papers and theses have been published on the topic, generating a continuously growing field.

Researchers have developed a variety of different techniques to encode neural networks for the GA, with increasing complexity. Mostly small, independent research groups that scarcely cooperate with each other drive this young field. This paper will attempt to analyze and to structure the already performed work, and to point out the shortcomings of the approaches.

The Problem of GANN Neural networks proved to be a powerful problem solving mechanism with the ability to learn. The success and speed of training, however, is based on the initial parameter settings, such as architecture, initial weights, learning rates, and others. In GANN systems, the genetic algorithm is used to find the optimal parameter settings for a given task.

We can distinguish two separate issues for the genetic algorithm: on the one hand architecture optimization and on the other hand weight training. A few GANN

Systems focus solely on weight training, whereas others focus on architecture. There are also many ways to combine the two issues: The GA may set initial weight information while weight training is left to established learning rules for NN, such as back- propagation. Or, the evolution of the NN can take place in two stages: One that finds the optimal architecture, the other that finds a weight setting for each architecture.

Weight encoding A few different complexities of weight encoding has been investigated in this field. The simplest case is one-bit connectivity information. Or, two bits are used for encoding if a connection is disconnected, inhibitory or excitatory. Maniezzo proposes that

the number of encoding bits should increase during the evolution, allowing fine-tuning at a later stage. A study suggests that a high number of encoding bits per weight improve GA weight training, whereas gray coding has no impact. Architecture encoding the task to encode the architecture of a network is far more complex than weight encoding. The classical view that the genome is a bit string of fixed length does not easily fit to the complex interconnected structure of a GA. Thus, most recent researchers are using more complex data types.

Their smallest defining unit or the "allele" can classify the different encoding strategies: connections, nodes, layers or pathways, or they can be indirectly encoded which is a completely different approach. The classes are roughly ordered to the complexity of the strategies. Connection-based encoding - The vast majority. The genome is a string of weight values or pure connectivity information. This requires a fixed maximal architecture, which is typically either fully connected or layered.

Node-based encoding - An advantage over connection-based encoding is that more flexibility can be obtained by using nodes as basic units. In this approach, the genome is a string or tree of node information. The code for each node may include relative position, backward connectivity, weight values, threshold function and more. Crossover and mutation is mostly restricted to cuts between node information. Layer-based encoding - With layer-based encoding one can obtain larger networks. The encoding scheme is a very complicated system of descriptions of connectivity between lists of layers. Therefore, special GA operators are required.

Pathway-based encoding - Pathway-based encoding is proposed for recurrent NN. In this instance, the network is viewed as a set of paths from an input to an output node. Again, special GA operators are used.

Indirect encoding - This completely separate strategy can be traced back to 1990; with more recent approaches receiving great acclaim in the community. The basic idea is to encode a grammar-based construction program in the genome, instead of directly encoding properties of the GA. Boers and Kuiper used a grammar based on Linden Mayer systems.

Gruau applied cellular encoding. Basically all encoding techniques developed so far, can be classified into one of these classes.

If the genomes are too large, the efficiency of the GA, and this problem has to be addressed when larger networks are involved. More complex encoding strategies are motivated by this lack of scalability, because not every connection or node has to be individually represented in the genome. With increasing complexity and inhomogenity of the genome, the classical GA operator's mutation and crossover are replaced by parameter-oriented operators such as "randomly add neuron.

Tasks for GANN systems Thanks to mutation operators, every optimal NN will be ultimately found by any GANN system. The quality of an encoding strategy, however, has to be measured by the speed of convergence to sufficiently good solutions. Most papers report good results on small-scale toy problems such as xor, sine, parity, or low-bit adding. The convergence time competes well with back- propagation.

The largest problems that have been tackled successfully range from high-bit parity, pole balancing, and simplified pattern recognition to high- order classification tasks. If convergence times are reported at all, they are usually quite high: 3 days on 11 workstations or one week, in conversation, for the mentioned tasks. These are tasks that can be solved faster by simple back-propagation on arbitrary NN chosen by rules of thumb. So, some doubts have to be raised regarding the efficiency of recent GANN systems.

Fundamental Problems In order to improve the performance of GANN systems, some of the fundamental problems of the approach must be resolved. Lack of theoretical insight - In general, the field Lacks theoretical insight. It is rare that general criteria for encoding strategies are proposed, such as by Gruau: completeness (every can be encoded) and closure (only meaningful NN are encoded) among others.

These are fulfilled by almost all strategies. Also, there have been only a few properties of GANN systems to be identified and discussed. The argument over the Baldwin effect or the permutation problem (see below) has just started. Permutation problem - Almost always, the same (or similar) networks may be encoded in quite different ways. The

crossover of two different encodings of similar networks can result in a completely divergent NN. This broadly acknowledged problem is also known as structural-functional mapping problem or competing conventions problem. There have been a few proposals to solve it. However, these studies also indicate that the problem has less impact than expected. GA parameters - There are numerous parameters for the genetic algorithms, the neural networks and the GANN systems that have to be set: population size, mutation and crossover rates, the use of distributed GA, fitness evaluation, learning rate, momentum term, number of epochs, activation functions, encoding details, et cetera.

Although many researchers describe the benefit of the application of GA to NN as the avoidance of rules of thumb in NN design, rules of thumbs and time-intensive experimental optimizations have to be used for these parameters. Still, it is not at all clear, if these parameters are robust or if they depend on the specific problems. In a few of the GANN systems, some of these parameters are encoded in the genome, for instance: the learning rate or number of encoding bits per weight. This encoding solves part of the problem at the cost of an increase of the search space and thus the convergence time. Comparison of encoding techniques - the inexistence of a theoretical account makes the comparison of different GANN systems difficult. The empirical results, however, do not clearly indicate that more complex systems show a better performance.

The fundamental problem of the poor performance of GANN systems on real-world scale tasks has been recognized. One magic word for the solution is modularity, but unfortunately encoding strategies with high modularity like indirect encoding, have not been a breakthrough.

Another idea is to partially develop NNs for subtasks and compose them to a more complex structure. This was investigated for different parts of the nervous system of an artificial being.

At this stage, however, the efficient application of GANN systems seems to be restricted to (a) weight training problems where no error information is available, such as for the pole-balancing problem, or the recurrent networks, and (b) architecture optimization for classes of problems that can use equal NN architectures.

3.6 Using Neural Networks and Genetic Algorithms for Technical Analysis

Genetic algorithms can be combined with neural networks to enhance their performance by taking some of the guesswork out of optimally choosing neural network parameters, inputs etc. In general, genetic algorithms can be used in conjunction with neural networks in the following four ways. They can be used to choose the best inputs to the neural network, optimize the neural network parameters (such as the learning rates, number of hidden layer processing elements, etc.), train the actual network weights (rather than using back propagation), or to choose/modify the neural network architecture. In each of these cases, the neural network error is the fitness used to rank the potential solutions. Trading Solutions can use genetic algorithms to pick the best inputs, optimize neural network parameters, and to optimize trading thresholds.

Now these concepts will be tied together to explain how neural networks and genetic algorithms can be used for technical analysis. In particular, three general methods will be discussed.

First, and probably most common, a neural network could be used to create a model for predicting the future price of a financial instrument, given the current and previous prices and other technical and/or fundamental data. The predicted price could then be used within an entry/exit system to produce a signal indicating when to buy or sell. The simplest entry/exit system would be: buy if the predicted price is greater than the current price and sell otherwise. Genetic algorithms could be used to optimize the neural networks inputs and parameters in order to produce the best model possible.

A second method would be to train a neural network to produce an entry/exit (buy/sell) signal. The desired entry/exit signal for the training phase could be produced by looking into the future to determine the optimum time to buy and sell given certain constraints and commission assumptions. The neural network could then be trained to produce this optimal signal using only current and historical data as inputs.

Since the output of this type of neural model is an entry/exit signal, it can be used directly without the need for further processing by an entry/exit system. Again, genetic algorithms could be used to optimize the inputs and parameters to the neural network in addition to the entry/exit system thresholds.

Finally, a neural network could be used to create a model for predicting the performance of a stock for a certain period into the future. For example, the inputs to the model could be the current price, the percent gain over the last week, the percent gain over the last month, etc. and the desired output could be the percent gain for the next week. If this type of model was created for a number of stocks, such as the stocks in the S&P 500, these stocks could be sorted by the projected percent gain for the next week. The top ten could be purchased and held for that week then re-chosen again each week according to the models' projections. This is only one simple way in which this type of model could be used for trading. As with the previous two methods for using neural networks, the networks inputs and parameters could be optimized using genetic algorithms.

These are just some of the ways neural networks and genetic algorithms can be used for technical analysis. Many other possibilities exist. Because Trading Solutions allows you the flexibility to decide the inputs and outputs, you are essentially limited only by your imagination.

3.7 Biological metaphors for finding good neural network architectures including genetic algorithm

Until now, artificial neural networks (ANNs) have been used successfully on small pattern recognition problems. With larger applications scalability problems appear to occur. In order to partly solve this scalability problem, we take biology as guideline. In biology, the evolutionary process has developed *modular* structures in the brains of living creatures, which are apparently suited for large applications.

In this research project an attempt is made to generate modular ANNs with genetic algorithms (GAs).

In order to mimic nature as closely as possible and maintain scalability, a context sensitive graph grammar (a graph generating L-system) is coded in the "chromosomes" of the GA. This graph grammar enables the generation of ANNs with a modular architecture.

The ANNs represented in the population of the GA are trained on a specific problem. The training result is fed back as fitness function for the GA. This process can be seen as a *reverse engineering* of nature.

A lot of research has to be done concerning the right parameters in the coding of the graph grammar and good/better crossover rules for the genetic algorithm. Until now, back propagation has been used as learning algorithm. Currently, the use of CALM-modules and corresponding learning algorithm is studied. The possibilities of using a combination with the cascade correlation algorithm (adapted for modular architectures) are also investigated. By being able to add extra nodes during learning, the search space for the genetic algorithm is smoothened (Baldwin effect), making it possible for the genetic algorithm to converge faster.

Apart from this, research is being done into improvements of ANNs, especially learning algorithms, which alter the architecture of ANNs as integral part of the learning algorithm. Research is also being done to obtain more insight into the possibilities of neural nets and genetic algorithms.

3.8 Genetic Algorithms and Genetically Evolved Neural Networks

Many studies have shown that ANNs have the capability to learn the underlying mechanics of time series, or, in the case of trading applications, the market dynamics. However, it is often difficult to design good ANNs, because many of the basic principles governing information processing in ANNs are hard to understand, and the complex interactions among network units usually makes engineering techniques like divide and conquer inapplicable.

When complex combinations of performance criteria (such as learning speed, compactness, generalization ability, and noise resistance) are given, and as network applications continue

to grow in size and complexity, the human engineering approach will not work and a more efficient, automated solution will be needed.

GA is reminiscent of sexual reproduction in which the genes of two parents combine to form those of their children. When it is applied to problem solving, the basic premise is that we can create an initial population of individuals representing possible solutions to a problem we are trying to solve.

Each of these individual has certain characteristics that make them more or less fit as members of the population. The most fit members will have a higher probability of mating than the less fit members, to produce offspring that have a significant chance of retaining the desirable characteristics of their parents. This method is very effective at finding optimal or near optimal solutions to a wide variety of problems, because it does not impose many of the limitations required by traditional methods. It is an elegant generate and test strategy that can identify and exploit regularities in the environment, and converges on solutions that were globally optimal or nearly so.

GA have been increasingly applied in ANN design in several ways: topology optimization, genetic training algorithms and control parameter optimization. In topology optimization, GA is used to select a topology (number of hidden layers, number of hidden nodes, interconnection pattern) for the ANN, which in turn is trained using some training scheme, most commonly back propagation. In genetic training algorithms, the learning of a ANN is formulated as a weights optimization problem, usually using the inverse mean squared error as a fitness measure. Many of the control parameters such as learning rate, momentum rate, tolerance level, etc., can also be optimized using GA. In addition, GA have been used in many other innovative ways, for instance, creating new indicators based on existing ones, selecting good indicators, to evolve optimal trading systems and to complement other techniques such as fuzzy logic.

There are four stages in the genetic search process: initialization, evaluation and selection, crossover and mutation. In the initialization stage, a population of genetic structures, which are randomly distributed in the solution space, is selected as the starting point of the search.

In the second stage, each structure is evaluated using a fitness function and assigned a fitness value. On the basis of their relative fitness values, structures in the current population are selected for reproduction. A stochastic procedure ensures that the expected number of offspring associated with a given structure s is u(s)/u(P), where u(s) is the observed performance of s and u(P) is the average performance of all structures in the current population.

Thus structures with high performance are more likely to be chosen for replication while poor performing structures are eventually removed from the population. In the absence of other mechanisms, such a selective process would cause the best performing structures in the initial population to occupy an increasingly larger proportion of the population over time.

The selected structures are recombined using crossover, with two complementary search functions. First, it provides new points for further testing of structures already present in the population; Secondly, it introduces instances of new structures into the population.

Generally, crossover draws only on the information present in the solutions of the current population in generating new solutions for evaluation. If specific information is missing (due to storage limitation or loss incurred during the selection process of a previous generation), then crossover is unable to produce new structures that contain this piece of information. A mutation operator, which arbitrarily alters one or more components of a selected structure, provides the means for introducing new information into the population. However, mutation functions as a background operator with a very low probability of application. The presence of mutation ensures that the probability of reaching any point in the search space is never zero.

3.8.1 Genetically Evolved Neural Networks

One method of automating ANN architecture design using GA is described below. It comprises two adaptive processes: genetic search through input data window, forecast horizon, network architecture space and control parameters to select the best performers, and back propagation learning in individual networks to evaluate the selected architectures.

The method begins with an initial population of randomly generated networks which are represented by overlapped tree structures as illustrated in **Fig. 3.3** below.

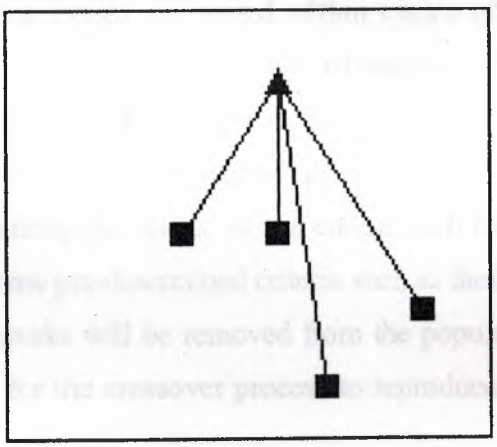

Figure3.3 One of the randomly generated network.

In **Fig3.3**, each rectangle represents an input node and the triangle represents an output node. The networks will grow and their hidden nodes will be inserted into the networks as the population evolves. The input nodes take a random combination of input data from an input data window, which picks up data items from the training data file and supplies them to the network. The output node randomly selects a forecast horizon from an output window and uses the associated data item as target value. Both windows slide down the training data file sequentially or randomly. **Fig3.4** illustrates an input window of size 3, and an output window with forecast horizons ranging from 1 to 3 steps ahead.

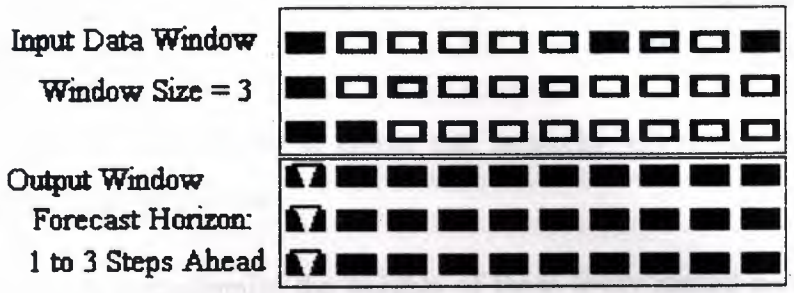

Figure3.4 The initial setting of the input and output windows used by all the networks.

The initial population of networks then goes through the first evolution cycle. As in real biological systems, learning cycles are nested within cycles of evolution in populations. Each learning cycle involves the entire population of networks with the set of input output pairs provided by the input and output windows. The networks' outputs are compared with the desired target, and the connection weights are adjusted to achieve the desired input output mapping by minimizing the errors. At the end of each learning cycle, the networks are ranked according to some pre-determined criteria such as their generalization capability. The poor-performing networks will be removed from the population, while the fitter ones are retained and selected for the crossover process to reproduce the offspring for the next generation.

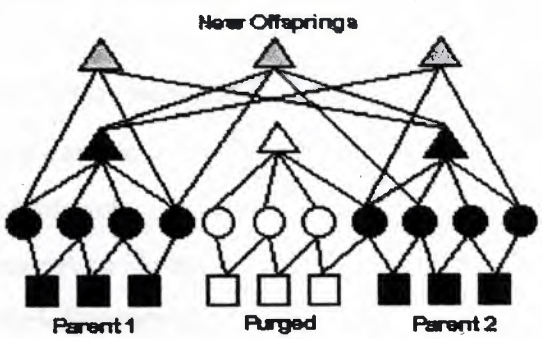

Figure3.5. Mating of two fittest parent networks to produce new offspring.

There are several ways to cross the parent nets. One way is to produce new offspring by mating two most fit parents as illustrated in **Fig3.5** The output nodes of the parents are used

as the hidden nodes of the offspring which will then inherit the knowledge already acquired by their parents.

Occasionally mutation is introduced to ensure that networks will not be trapped in local minima during the learning process.

One way to mutate is to randomize the weights of those lowly ranked networks, change their input combination in the input data window and/or their forecast horizon.

After each evolution cycle, an image of the fittest network is kept. The image includes the current input data combination, forecast horizon, interconnection patterns and weights of the fittest network, such that the fittest network can go through the next evolution and training cycles together with the rest of the newly formed population.

Training Cycle

Increment Iteration Count

Neural Net Learning

Check Criteria To Stop? If Yes –> Stop

- Check Time To Evolve? If Yes –> **Evolution Cycle**
  - Evaluate Network Population
  - Rank Network Population
  - Store Image of Fittest Network
  - Select Most Fit Parents
  - Crossover & Mutation
  - Increment Generation Count
  - Update Network Population
  - Continue training cycle

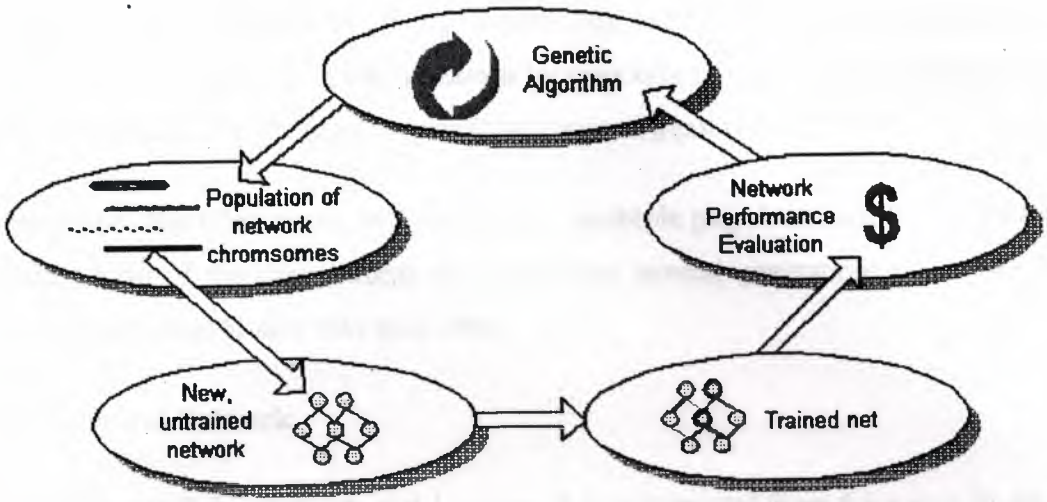

Figure3.6 The training cycle with the nested evolution cycle.

The image will remain intact during subsequent evolution cycles until another fittest network emerges. A complete evolution cycle, with the nested training cycle, is illustrated in **Fig3.6.**

3.9 Using Genetic Algorithms in Computer Learning

Essentially, **Genetic Algorithms** (GA) are a method of "breeding" computer programs and solutions to optimization or search problems by means of **simulated evolution**. Processes loosely based on **natural selection, crossover, and mutation** are repeatedly applied to a **population** of binary strings which represent potential solutions. Over time, the number of above-average individual's increases, and better-fit individuals are created, until a good solution to the problem at hand is found.

In NNUGA, we are using GA to find the solution to a **classification problem** with a **neural network** (NN). The neural network is a structure, which is able to respond with *True* (1) or *False* (0) to a given input vector. We are trying to "teach" our neural network to correctly classify a set of input vectors, which can be thought of as **learning a concept**.

We then expect that when the neural network will be presented with a vector P not from this set, it will tend to exhibit **generalization** by responding with an output similar to target vectors for input vectors close to the previously unseen input vector P.

Our genetic algorithm works with not one but **multiple populations**, all of which evolve separately most of the time, except for once every several generations where we allow different populations to mix with each other.

3.9.1The Neural Network

Our neural network has 2 inputs and 1 output. It is constructed from 6 neurons in the first layer and 1 neuron in a second layer.

In the first layer we have 2 neurons with atan transfer functions:

$$\text{Atan}\ (P * W + b)$$

2 with linear transfer functions:

$$P * W + b$$

And 2 with hardlim transfer functions:

$$P * W + b > 0$$

where P is the input vector presented to the network, W is the vector of weights and b is the bias of the neuron.

The output function of the neuron in the second layer is:

$$A * W + b > 0$$

Where A is the vector of the outputs of the neurons in the first layer, W is the vector of weights and b is its bias. The output of this neuron is also the output of the network.

3.9.2 The Learning Method

As stated earlier, the neural network learns using genetic algorithms. This means that the weights and biases of all the neurons are joined to created a single vector. A certain set of vectors is a correct solution to the classification problem at hand. We are hoping to find one of these vectors using an evolutionary algorithm.

The Canonical GA (pseudo code):

```
choose initial population
evaluate each individual's fitness
    repeat
        select individuals to reproduce
        mate pairs at random
        apply crossover operator
        apply mutation operator
        evaluate each individual's fitness
    until terminating condition
```

The learning loop can terminate either if a satisfactory solution is found, or the number of generations pass a preset limit, suggesting that a complete solution will not be found with this set of individuals.

In our case, we have several seperated populations, all of which evolve seperately. Once every several generations we allow cross-over from different populations.

3.9.3 Limitations

Sometimes it could happen that though the NN could theoretically solve a certain classification problem, our system will not return a correct solution.

This is because of the random nature of the algorithm and its reliance on natural selection, mutation and cross-overs.

Naturally, it could happen that a certain flow of events that would lead to a correct solution will not occur and thus a solution will not be found. However, by using several unrelated populations we have decreased the probability of this happening, since if some population has poor individuals the solution could still be found at another.

3.9.4 Implementation

The network is implementing with 2 inputs. The input for the neural network can be taken from a graphic user interface, by clicking on points in a board. A click with the left mouse button generates a '+' sign on the board; marking that it's a point where the NN should respond with 'True'. A click with the right mouse button generates a '-' sign on the board, marking that it's a point where the NN should respond with 'False'. When enough points have been entered, the user can click on 'Start', which will introduce these points as inputs to the NN, have it learn these input vectors and show a group of lines which corresponds to the division of the plane into regions of opposite neuron response. While the learning takes place, a textual indication of the learning process is presented on the standard output; this includes the fitness of the best individual in each population on each generation, and a schematic textual division of the plane once every 50 generations, to allow the user to inspect the progress.

3.10 Summary

The theory of natural selection offers some compelling arguments that individuals with certain characteristics are better able to survive and pass on those characteristics to their offspring. A genetic algorithm is a general search procedure based on the ideas of genetics and natural selection, and its power lies in the fact that as members of the population mate, they produce offspring that have a significant chance of retaining the desirable characteristics of their parents, perhaps even combining the best characteristics of both parents. In this manner, the overall fitness of the population can potentially increase from generation to generation as we discover better solutions to our problem.

When applied to the optimization of ANNs for forecasting and classification problems, GAs can be used to search for the right combination of input data, the most suitable forecast horizon, the optimal or near optimal network interconnection patterns and weights among the neurons, and the control parameters (learning rate, momentum rate, tolerance level, etc.), based on the training data used and the pre-set criteria. Like ANNs, GAs do not always guarantee you a perfect solution, but in many cases, it can arrive at an acceptable solution without the time and expense of an exhaustive search.

In most of the Genetic Algorithm approaches where both network structure and weights are subject to genetic operations, the chromosome representing both parts is thought of as a long binary string and is subject to the genetic operators of the algorithm. This is the most common use of a genetic algorithm in conjunction with a neural network. Since genetic algorithms are excellent at searching a state-space, searching neural network weights is an ideal application.

In the fourth chapter the applications of genetic algorithm will be presented.

CHAPTER FOUR

APPLICATION OF GENETIC ALGORITHM

4.1 Overview

Traditional methods of search and optimization are too slow in finding a solution in a very complex search space, even implemented in supercomputers. Genetic Algorithm is a robust search method requiring little information to search effectively in a large or poorly understood search space. In particular a genetic search progress through a population of points in contrast to the single point of focus of most search algorithms. Moreover, it is useful in the very tricky area of nonlinear problems. Its intrinsic parallelism (in evaluation functions, selections and so on) allows the uses of distributed processing machines. Genetic Algorithm (GA) is an artificial intelligence procedure. It is based on the theory of natural selection and evolution. This search algorithm balances the need for: exploitation - Selection and crossover tend to converge on a good but sub-optimal solution. Exploration - Selection and mutation create a parallel, noise-tolerant, hill-climbing algorithm, preventing a premature convergence.

Basically, Genetic Algorithm requires two elements for a given problem encoding of candidate structures (solutions).Method of evaluating the relative performance of candidate structure, for identifying the better solutions.

Genetic Algorithm codes parameters of the search space as binary strings of fixed length. It employs a population of strings initialized at random, which evolve to the next generation by genetic operators such as selection, crossover and mutation. The fitness function evaluates the quality of solutions coded by strings. Selection allows strings with higher fitness to appear with higher probability in the next generation. Crossover combines two parents by exchanging parts of their strings, starting from a randomly chosen crossover point. This leads to new solutions inheriting desirable qualities from both parents. Mutation

flips single bits in a string, which prevents the GA from premature convergence, by exploiting new regions in the search space. GA tends to take advantage of the fittest solutions by giving them greater weight, and concentrating the search in the regions, which lead to fitter structures, and hence better solutions of the problem.

Finding good parameter settings that work for a particular problem is not a trivial task. The critical factors are to determine robust parameter settings for population size, encoding, selection criteria, genetic operator probabilities and evaluation (fitness) normalization techniques.

If the population is too small, the genetic algorithm will converge too quickly to a local optimal point and may not find the best solution. On the other hand, too many members in a population result in long waiting times for significant improvement.

Coding the solutions is based on the principle of meaningful building blocks and the principle of minimal alphabets, by using the binary strings.

The fitter member will have a greater chance of reproducing. The members with lower fitness are replaced by the offspring. Thus in successive generations, the members on average are fitter as solutions to the problem.

Too high mutation introduces too much diversity and takes longer time to get the optimal solution. Too low mutation tends to miss some near-optimal points. Two-point crossover is quicker to get the same results and retain the solutions much longer than one point crossover.

The fitness function links the Genetic Algorithm to the problem to be solved. The assigned fitness is used to calculate the selection probabilities for choosing parents, for determining which member will be replaced by which child.

4.2 State Assignment Problem

This State Assignment Problem (SAP) belongs to a broader class of combinatorial optimization problems, including the Traveling Salesman Problem (TSP). The aim is to

3.4 Scheduling

Genetic Algorithm is used for inspection and repair of oil tanks and pipelines. The implementation is built on Peter Ross' PGA testbed and the data is taken from the Expert Systems for the Inspection of Tanks and Pipelines SITA and SIGO. The fitness function evaluates the constraints: level of production, condition and location of installations, type of products, human resources, the dates and costs of inspection and repairs. A good inspection schedule for oil installations is constructed. A good schedule ensures that repair times are kept to a minimum and faults are found before they become too serious. An automatic way of assigning maintenance activities to inspectors is devised in such a way as to minimize the loss in capacity, while keeping within resource constraints.

The schedule is evaluated taking into account the following priorities:

A tank, which requires urgent maintenance, is checked early in the schedule (very good).

A tank or pipeline requiring a periodic maintenance or inspection is included in the schedule (good), given higher priority to the first case. Because of several tanks in one location being out of service simultaneously, the capacity of that location for a certain time drops significantly (very bad). Some inspectors have more work to do than the others in the same area (bad). The application distributes the repairs such that the available capacity is always larger than the required minimum, then the production is not affected. Moreover, the assignment of activities is appropriate; it reduces the cost of unbalanced distribution. A robust schedule of activities is obtained.

4.5 Computer-Aided Design

Genetic Algorithm uses the feedback from the evaluation process to select the fitter designs, generating new designs through the recombination of parts of the selected designs. Eventually, a population of high performance designs is resulted.

4.6 Genetic evolution - A way to solve complex optimization problems

- The complexity of a problem lies in the complexity of the solution space that may be searched through. This complexity arises due to:

Size of the problem domain; non-linear interactions between various elements (epitasis); domain constraints; performance measure with dynamics and many independent and codependent elements; and incomplete, uncertain, and/or imprecise information.

- Systems of nature routinely encounter and solve such problems. Good examples include genetic evolution of species, which become better and better suited to the environment generation by generation.
- A *Darwinian machine* or an *evolutionary program* emulates this process and does not require an explicit description of how the problem is to be solved. It tends to evolve optimal solutions automatically.
- Evolutionary computing technology:

Maps an engineering system directly onto a "genetically" encoded (numerical or character) string of system parameters and structures; can convert an automatic design problem to a simulation problem to solve;

Solves a difficult design problem by intelligently evaluating performance and evolving optimal candidates based upon the evaluations.

4.7 Some practical applications

This section presents some practical applications of G.A. in real life.

1. The genetic algorithm (GA) has been applied to game playing machines, pattern recognition, and optimization of pipeline systems, the traveling salesman problem, and numerous other optimization problems, with varying degrees of success. This paper demonstrates how to apply the genetic algorithm to nonlinear recursive system

identification problems on which gradient descent methods fail miserably. A previous attempt to use the algorithm for identification purposes was stymied by slow convergence and biased estimates. A simple modification to the algorithm is proposed which increases the speed of convergence, and appears to solve the bias problem. Analysis, in terms of the convergence of a particular martingale difference sequence indicates why the modifications are successful.

Application of the genetic algorithm requires that the problem be coded in parameter strings (called genes), and that there be an evaluation criterion, which can determine a fitness value associated with each gene. As usually implemented, there are two phases to the genetic algorithm. During the initial phase, characterized by a highly diverse genetic pool and dominated by matings of highly diverse "parents", the parameter estimates move rapidly towards the desired parameterization. In the second phase, when the gene pool is highly uniform and the incorporation of "new" genetic material must rely on the mutation process, the motion of the parameters is painfully slow. Our modification essentially monitors the diversity of the gene pool, and forces a mass "extinction and immigration" whenever the diversity has fallen below a preset value. Several examples illustrate the method, including linear and nonlinear, FIR and IIR identification. The method is also applied to identify the parameters in layered feed forward and feedback (recurrent) neural network structures.

2. A variant of the Genetic Algorithm is used to place sensors optimally on a Large Space Structure for the purpose of modal identification. The selection and reproduction schemes of the Genetic Algorithm are modified and a new operator called forced mutation is introduced. These changes are shown to improve the convergence of the algorithm and to lead to near optimal sensor locations. Two practical examples are investigated; sensor placement for an early version of the Space Station and an individual Space Station photovoltaic array. Simulated results are also compared with previous results obtained by the Effective Independence method. The Genetic Algorithm based sensor configuration estimates the target mode response more accurately.

This method of sensor placement first appeared in the *Journal of the American Institute of Aeronautics and Astronautics,* in October 1993

4.7 Applications of Genetic Algorithms to Solar Coronal Modeling

Genetic algorithms are efficient and flexible means of attacking optimization problems that would otherwise be computationally prohibitive. Consider a model that represents an observable quantity in terms of a few parameters, with an associated \chi^2 measuring goodness of fit with respect to data. If the modeled observable is non-linear in the parameters, there can exist a degeneracy of minimum \chi^2 in parameter space. It is then essential to understand the location and extent of this degeneracy in order to find the global optimum and quantify the degeneracy error around it. Traditional methods of spanning parameter space such as a grid search or a Monte Carlo approach scale exponentially with the number of parameters, and waste a great deal of computational time looking at ``un-fit" solutions. Genetic algorithms, on the other hand, converge rapidly onto regions of minimum \chi^2 while continuously generate ``mutant solutions", allowing an efficient and comprehensive exploration of parameter space. Our aim has been to develop an approach that simultaneously yields a best-fit solution and global error estimates, by modifying and extending standard genetic algorithm-based techniques.

We fit two magneto static models of the solar minimum corona to observations in white light. The first model allows horizontal bulk currents and the second also allows sheet currents enclosing and extending out from the equatorial helmet streamer.

Using our genetic algorithm approach, we map out the degeneracy of model parameters that reproduce observations well.

The flexibility of genetic algorithms facilitates incorporating the additional observational constraint of photospheres magnetic flux, reducing the degeneracy of solutions to a range represented by reasonable error bars on the model predictions. By using genetic algorithms we are able to identify and constrain the degeneracy inherent to the models, a task, which,

particularly for the more complex second model, would be impractical using a traditional technique. The ultimate result is a greater understanding of the large scale structure of the solar corona, providing clues to the mechanisms heating the corona and accelerating the solar wind.

4.7.1 PART I: METHODOLOGICAL ASPECTS

- Using Genetic Algorithms for Optimization: Technology Transfer in Action.
- An Introduction to Evolutionary Computation and Some Applications.
- Evolutionary Computation: Recent Developments and Open Issues.
- Some Recent Important Foundational Results in Evolutionary Computation.
- Evolutionary Algorithms for Engineering Applications.
- Embedded Path Tracing and Neighborhood Search Techniques.
- Parallel and Distributed Evolutionary Algorithms.
- Evolutionary Multi-Criterion Optimization.
- ACO Algorithms for the Traveling Salesman Problem.
- Genetic Programming: Turing's Third Way to Achieve Machine Intelligence.
- Automatic Synthesis of the Topology and Sizing for Analog Electrical Circuits. Using Genetic Programming.

4.7.2 PART II: APPLICATION-ORIENTED APPROACHES

- Multidisciplinary Hybrid Constrained GA Optimization.
- Genetic Algorithm as a Tool for Solving Electrical Engineering Problems.
- Genetic Algorithms in Shape Optimization: Finite and Boundary Applications.
- Genetic Algorithms and Fractals.
- Three Evolutionary Approaches to Clustering.

4.7.3 PART III: INDUSTRIAL APPLICATIONS

- Evolutionary Algorithms Applied to Academic and Industrial Test Cases.
- Optimization of an Active Noise Control System inside an Aircraft, Based on the Simultaneous Optimal Positioning of Microphones and Speakers, with the Use of a Genetic Algorithm.
- Generator Scheduling in Power Systems by Genetic Algorithm and Expert System.
- Efficient Partitioning Methods for 3-D Unstructured Grids Using Genetic Algorithms.
- Genetic Algorithms in Shape Optimization of a Paper Machine Head box.
- A Parallel Genetic Algorithm for Multi-Objective Optimization in Computational Fluid Dynamics.
- Application of a Multi Objective Genetic Algorithm and a Neural Network to the Optimization of Foundry Processes.
- Circuit Partitioning Using Evolution Algorithms.

4.8 SUMMARY

Genetic Algorithm is a robust search method requiring little information to search effectively in a large or poorly understood search space. And also the Genetic Algorithm has been used successfully in many practical applications especially Economics, scheduling, pattern recognition, optimization of pipeline systems, the traveling salesman problem and industrial fields. Genetic algorithms are efficient and flexible means of attacking optimization problems that would otherwise be computationally prohibitive. A variant of the Genetic Algorithm is used to place sensors optimally on a Large Space Structure for the purpose of modal identification.

In chapter five, the genetic algorithm for telecommunication system design will be presented.

CAPTER FIVE

GENETIC ALGORITHM FOR TELECOMMUNICATION DESIGN

5.1 Overview

Many of design problems in telecommunications could be treated as optimization problems that include some kind of searching among a set of potential solutions. The choice of the method depends mostly on the problem complexity. If the number of possible solutions is not too big, one could enumerate them all, evaluate their goal functions, and select the best solution. If the function to be optimized is done by a derivative continuous function, analytical methods could be applied. In all other cases, where the problem space is too big and analytical methods are not applicable, some sort of heuristic search for optimal solution could be applied.

Genetic algorithms, to be presented in this chapter, could be classified as guided random search evolution algorithms that use probability to guide their search. Genetic algorithms are created by analogy with the processes in the reproduction of biological organisms.

By natural selection or by forced selection in laboratories, new generations of organisms are produced. As a consequence of crossover and mutation processes on chromosomes and genes, the children could possess either better or worse features than their parents. The "better" organisms are those that have a greater chance than the "worse" ones to survive and to produce a new generation.

5.2 Genetic Algorithm Fundamentals

The functioning of a genetic algorithm (GA) could be described by using specific data structure and by defining the set of genetic operators, as well as the reproduction procedure. In this section the explanations are mostly referred to the restricted class of simple genetic algorithms (SGA) including possible extensions to the more complex ones [14]. The terms used are a mixture taken from both biological and computational domains.

The biological organism to be-specified is defined by one or by a set of chromosomes. The overall set of chromosomes is called genotype, and the resulting organism is called phenotype. Every chromosome consists of genes. The gene position within the

chromosome refers to the type of organism characteristic, and the coded content of each gene refers to an attribute within the organism type.

In GA terminology, the set of strings (chromosomes) forms a structure (genotype). Each string consists of characters (genes). In SGA the character is two-valued, represented by a bit. In a search procedure, a string represents a full solution and a character (bit) represents a variable to be found, perhaps, as optimal.

The quality of a single solution in GA is determined by a fitness function. The role of the fitness function could be twofold: to award the goodness of a solution on the one band, and to punish the solution not satisfying constraints on the other. One solution is better than some other, if it has a higher fitness value. For an optimization procedure. The fitness function is equivalent to the goal function that could have a global optimum and a number of local optima.

In order to provide a reproduction process, producing a new generation of strings from the old one, in SGA the following genetic operators are used: reproduction, crossover, and mutation.

5.3 Calls and Service Processing in Telecommunications

5.3.1 Parallel Processing of Calls and Services

Call is a generic term related to the establishment, utilization, and the release of a connection between a calling user and the called user for the purpose of exchanging information. The coil is also defined as an association between two or more users, or between a user and a network established by using network capabilities. A network offers Service to its users, in order to satisfy a specific telecommunication requirement.

Users initiate calls and services. Each call/service causes a number of processing requests. The call/service can be represented by a sequence of requests occurring in random intervals. The first request initiates the call, while further requests start different communication and information operations - named call phases. On this level of abstraction, unsuccessful calls shorten the call because some phases are skipped. The services shorten, modify (some phases are replaced by others), or enlarge the call (new phases are inserted), so the number of requests varies from case to case. The mean number

of requests depends on implemented set of services as well as on traffic and other conditions in the network.

The call/service decomposition on phases depends on the interaction with the environment; information from the environment (request input data) is needed to start a specific call phase, and/or information (result) is sent to the environment after completing a call phase. Different types of calls and services represent different types of independent, cooperating, or mutually excluding tasks to be run on the network.

It should be pointed out that, when discussing call and service handling, control plane of the telecommunication network must be taken into consideration. Considering the aspect of control, the system is reactive. After receiving a request, and reflecting defined time conditions, a response has to be produced; otherwise, a call/service will be lost.

5.3.2 Scheduling Problem Definition

Each call/service processing task consists of a number of processes. The question is what could be considered as a basic call/service grain - what the elementary task (ETs) is or how simple/complex it is. It is well known that telecommunication applications are programmed as if they were composed of a large number of small agents.

The solution with higher parallelism generally could be obtained with smaller ETs, but the proper boundary also depends on the way in which the concurrent programming language supports the creation and destruction of parallel activities, as well as interprocess communication. The programming reasons are against unconstrained parallelism; i.e., parallel activities at the instruction level [15].

An application of genetic algorithm for scheduling implies a careful definition of GA constructs, especially the following.

Schedule representation

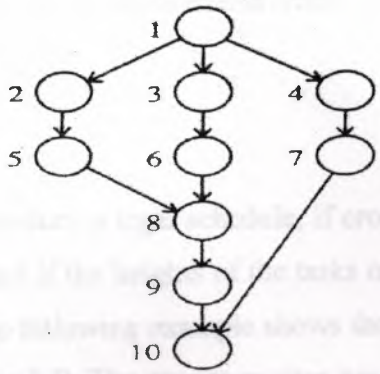

Figure 5.1 Graph representation of a task

An application of genetic algorithm for scheduling implies a careful definition of GA constructs, especially the following.

Schedule representation

The representation of a schedule must accommodate the precedence relation between ETs. The schedule is represented by means of several lists of ETs, one for each processor. The order of tasks in the list corresponds to the order of execution. Every task appears only once in the schedule. A schedule satisfying all these conditions is the legal schedule; Let me consider two processors, P1 and P2, and the graph representation of a task pattern shown in Figure5.1 Legal schedules A and B, both with the finishing time FT, equal to deltat, are, for instance,

	P1	P2	
A	[1 2 3 4 7 6]	[2 5 8 9 10]	FT=8
Height	0 1 1 2 2	1 2 3 4 5	
B	[1 2 4 6 7 10]	[3 5 8 9]	FT=8
Height	0 1 1 2 2 5	1 2 3 4	

Evidently, the schedule representation is more complex than string representation used in SGA, as described in Section 5.3.1 the genetic operators must maintain the

precedence relation in each ET list (processor) and ensure a unique appearance of all ETs in the schedule.

Crossover

Crossover operator will produce a legal schedule, if crossover sites always lie between ETs with different heights, and if the heights of the tasks on the first right positions of the crossover sites are equal. The following example shows the result of a crossover operation producing new schedules *C* and *D*. The crossover sites are placed between following pairs of ETs: in processor P1, schedule *A* (ET4, ET7), in P1, *B* (ET4, FT6), in *P2, A* (ET8, and ET9), and in *P2, B* (ET8, ET9).

	P1	P2	
C	[1 3 4 6 7 10]	[2 5 8 9]	FT=7
Height	0 1 1 2 2 5	1 2 3 4	
D	[1 2 4 7 6]	[3 5 8 9 10]	FT=8

In this example, schedule *C* has a better fitness value than the parent schedules.

Mutation

Mutation operator will produce legal schedule, if ETs with identical heights are mutated, exchanging their positions in the schedule. For example, by using mutation operator on ET7 and ET6 of schedule *A,* a new schedule E is produced.

	P1	P2	
E	[1 3 4 6 7]	[2 5 8 9 10]	FT=7
Height	0 1 1 2 2	1 2 3 4 5	

In this example, the new schedule E has a better fitness value than old schedule *A*.

5.4 Analysis of Call and Service Control in Distributed Processing Environment

5.4.1 Model of Call and Service Control

According to the call and service-processing concept described in Section 5.3.1, a model of call and\service control is introduced. It consists of a processing system and its environment representing the telecommunication network and its users, Requests Coming from the environment enter the request queue limited to *AT* places. If the number of requests is greater than *N*, a loss occurs. Requests wait until the processing of the previous request has been completed. After that, all requests from the queue are moved to the task pattern formatting stage where a set of elementary tasks with their precedence relations is associated to every request. Further on, using GA. Finally, performs the elementary task scheduling, elementary tasks enter processor queues.

5.4.2 Simulation of Parallel Processing

A genetic algorithm is applied for finishing time determination in a simulation method used for the analysis of parallel processing in telecommunications. The method includes three steps: request flow generation, finishing time determination, and response time calculation, that are repeated as many times as pointed out by defined number of simulation samples. The simulation is based on a generation of processing request flow with random structures with respect to the number of ETs and their precedence relations. A stochastic nature of processes is defined with probability functions for arrival characteristics of processing requests, and the number of ETs per request

5.4.3 Genetic Operators

Reproduction

Reproduction is the basic genetic operator. Reproduction forms a new generation by choosing those individuals from the old population that have the highest value of the fitness function. The choice is based on the fact that the individuals with the highest value of the fitness function have greater chances to survive in the next generation. In our application it means that better task schedules have higher value of the fitness function and, because of that, they have to be preserved in the next generation. The choice is made by the roulette wheel selection technique. Since the schedules with a higher value of the fitness function take a greater part of the wheel, they have a higher probability to be chosen in the next generation- Also, an elitism procedure can be added, so that the best schedule from the old generation always moves into the new generation without a random choice.

Crossover

Two schedules *A* and B from the population are taken. Changing the parts of the schedules A and B according to the following crossover procedure can make two new schedules C and D:

The left parts of the schedules remain unchanged, and the right parts are subjected to the crossover in such *a* way that the right part of the schedule *A* becomes the right part of the schedule *B* and vice versa.

Mutation

Mutation is considered as a stochastic alternation of the genetic code value in selected places in the schedule. Some modifications in the mutation operation defined in Section 1 are proposed because GA used for finishing -time determination - manipulates the ET sets. The modified procedure follows:

1. The mutation is going to be performed by choosing a phase at random between the first and the last phase in the best schedule.

2. The processors with the highest and the lowest load (number of ETs) are determined in the chosen phase.
3. The ETs are redistributed between these processors in a way that half of them are allocated to each one.

5.4.4 Complete Algorithm and Analysis Results

The flow chart of the complete genetic algorithm, whose parts have been described previously, is shown in Figure 5.2.

The start is defined by assigning the number of the generations explored by a genetic algorithm.

Afterward, the generation of the initial population is performed randomly, according to the assigned number of processors and the number of schedules in a population. Also, mutation and crossover probabilities, that are going to be used later, are defined.

After that, a fitness function is calculated for all schedules in the population.

For the schedule with the highest value of the fitness function, an operation of the modified mutation is done with the assigned probability that improves the best schedule.

The reproduction, the key operation of the genetic algorithm, follows. A new population, having the same number of schedules as the old one, is obtained. The best schedule is moved directly from the old generation into a new one.

Afterward, a new population is subjected to the crossover operation with assigned probability. Fitness function calculation shows the features of a new generation.

A resulting generation containing the best schedules is obtained when the defined number of generations is reached. After fulfilling convergence criterion, the algorithm terminates.

For the illustration of the genetic algorithm analysis in the given examples, an obtained finish time is considered as a complement of the fitness function in certain iterations.

A series of experiments was done to evaluate a performance of the genetic algorithm itself. Different problems, from regular to a heavy request flow, including request bursts were analyzed. Also, the initial population with respect to generated strings (schedules) and size has been evaluated. The comparison criterion was the best string in the population (the string with the lowest finishing time).

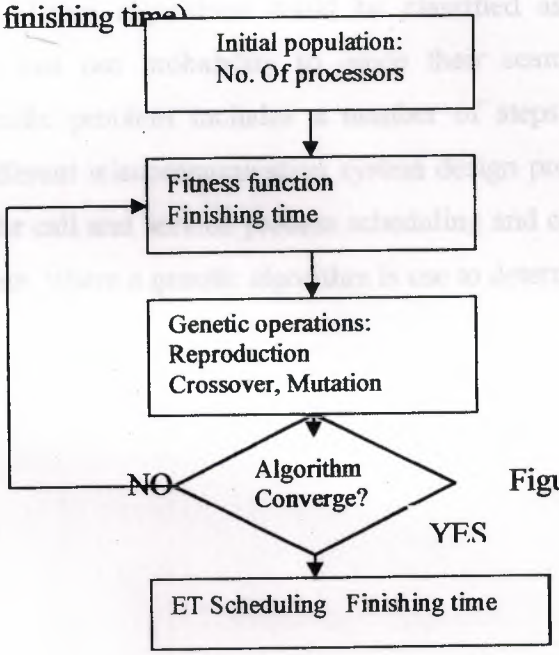

Figure 5.2 Flow chart of G.A

5.5 SUMMARY

In this chapter an application of genetic algorithms in telecommunications is described. Genetic algorithms are based by analogy with the processes in the reproduction of biological organisms. These algorithms could be classified as guided random search evolution algorithms that use probability to guide their search. A genetic algorithm application to a specific problem includes a number of steps and some of them are discussed in three different telecommunication system design problems. Two of them are related to a method for call and service process scheduling and call and service control in distributed environment, where a genetic algorithm is use to determine a response time.

CONCLUSION

Genetic algorithms are the future, no body can say for sure that what they will bring in our life, but it can be stated that their results will be beyond any ones expectations. The integration of Genetic Algorithms can be applied in many scientific complicated research areas, which can lead to the solution of many unanswered questions that require human intelligence. Systems can be trained to work and think like humans, e.g. modeling of natural systems, encoding and decoding of parameters, machine learning, evolution of strategies.

Chapter One was brief history and background of Genetic Algorithm with a brief comparison between genetics and genetic algorithms then a discussion of the structure of the genetic algorithms. Genetic algorithms are a part of evolutionary computing which is a rapidly growing area of the artificial intelligence. Genetic engineering has produced many advances in medicine and industry, but the potential for abuse of this technique has also presented society with many ethical and legal controversies. Chapter Two was a brief discussion on history of neural networks, with their evolution in biological terminology, As we mentioned there are two types of learning algorithms, first supervised learning which includes the input vectors and output vectors and the law of operating (e.g. the perceptron, the back propagation algorithm, the Hopfield network, the hamming network), second unsupervised learning which depends on the figure of merit (e.g. Kohonen's self-organizing map, Competitive learning, Adaptive resonance theory).In chapter three developing neural network using genetic algorithm, evolving weights and using genetic algorithms in computer learning was discussed. In chapter four application of genetic algorithms in economics, scheduling, computer-aided design and the way of solving complex optimization problem by genetic evolution was discussed. In chapter five the genetic algorithm for telecommunication system design was discussed.

Genetic Algorithms can offer great benefits for mankind in terms of simulating human perception in machine, there are optional problems related to this novel approach in the Artificial Intelligence. These problems have so for been ethical.

This project is about "Genetic Algorithm Implementation" and it is definitely related with neural network. So that we discussed a lot of points about neural network and also the subjects, which have related with the application of neural network during our studies we found the definition of neural network is a massively parallel-distributed processor that has a natural propensity for experiential knowledge and making it available for use.

So from discussed topics we can conclude one from more benefits of Genetic Algorithm Implementation, which is (machine learning).

The machine can be trained but machines don't have feelings as humans do, this can lead to the invention of machines of massive destruction, which can have negative effect on the human race. On the other hand such machines can turn on their own inventors during their training periods.

REFERENCES

[1] Rechenenberge and D.W. Clark, ***An Adaptive Attentive Learning Algorithm for Single Layer Neural Networks,*** in Learning AlgorithmsII, Proceedings of the First Annual IEEE Conference on Neural Networks, vol. I, pp. 431-440, 1988.

[2] John Holland, ***An Algorithm for Linear Inequalities and its Application,*** IEEE Trans. Elec. Comp. , EC-14 (5), PP. 683-688, 1965.

[3] John & koza, ***Threshold Logic and its Applications,*** John Wiley and Sons, New York, 1972

[4] Schwefel and J. Song, ***Adaptive Ho-Kashyap Rules for Perceptron Training,*** IEEE Trans. Neural Networks, vol. 3, no. 1,1992.

[5] Toombs & Baricell, ***Adaptive Dynamic Heterassociative Neural Memories for Pattern Classification,*** in Optical pattern Recognition, H-K. Liu, ed., Proc. SPIE, Vol. 1053, pp 75-83, 1989.

[6] Assoc. Prof. Dr Adnan Kashman, ***Neural networks Lecture Notes.*** Nicosia November 2001.

[7] D. Hebb, (1992) ***How Neural Network Learn from Experience.*** Scientific American 267, 144-151.

[8] T. Kohonen ***A Simple Neural Network Generating an Interactive Memory,*** Mathematical Biosciences,1972.

[9] Rumelhart, ***The Physical Background of Perception,*** Clarendon Press, Oxford, 1946.

[10] Montana & David, ***Self Organization and Associative Memories,*** Springer-Verlag, New York, 1984.

[11] Kargupta, and Harik, ***Foundation of Genetic Algorithms,*** MIT Press MA, England, 1991.

[12] Utragpom, Rumelhart DE and Mcclelland JL (1986) ***Parallel distributed Processing: Explorations in the microstructure of cognition.*** MIT Press, Cambridge, Vols. I and II. This is the book that started the explosion of uses on ANNs in real world.

[13] Kenneth De Jong, ***The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain,*** Psychological Review 65:386-408, 1956.

[14] Goldberg, D.E. (1989). ***Genetic Algorithm in Search Optimization and Machine Learning,*** Addison- Wesley, Reading.

[15] Dacker, B. (1993), ***Erlong-A New programming Language,*** Ericsson Revie, Vol. 70,No.2.