# NEAR EAST UNIVERSITY

## FACULTY OF ENGINEERING

### Department Of Computer Engineering

## Genetic Algorithm Based Optimization

Graduation Project
COM 400

Student: Gauhar Ayub (960669)

Supervisor: Asst. Prof. Dr. Rahib Abiyev

Lefkosa-2000

# Dedication:

Dedicated to my parents and teachers for their continued guidance and prayers.

# ACKNOWLEDGEMENTS

First of all I am happy to complete the task which I had given with blessing of God and also I am grateful to all the people in my life who have, supported me, advised me, taught me and who have always encouraged me to follow my dreams and ambitions. My dearest parents, my brothers and sister, my friends and my tutors. They have taught me that no dream is unachievable. As in the words of Walt Disney "If you can dream it, you can do it."

I wish to thank my advisor, Assist. Prof. Dr. Rahib Abiyev, for intellectual support, encouragement, and enthusiasm, which made this project possible, and his patience for correcting both my stylistic and scientific errors.

My sincerest thanks must go to my friends, Muhammad Wajid, Mr.Tong Zhengrong, Waqar Ahmed, Shahzad Naeem, Malik Shahbaz and Nauman Shaukat Chaudhry who shared their suggestions and evaluations throughout the completion of my project. The comments from these friends enabled me to present this project successfully.

And above, I thank God for giving me stamina and courage to achieve my objectives.

## ABSTRACT

By increasing complexity of processes, it has become very difficult to control them on the base of traditional methods. In such condition it is necessary to use modern methods for solving these problems. One of such method is global optimisation algorithm based on mechanics of natural selection and natural genetics, which is called Genetic Algorithms. In this project the application problems of genetic algorithms for optimisation problems, its specific characters and structures are given. The basic genetic operation: Selections, reproduction, crossover and mutation operations are widely described .The affectivity of genetic algorithms for optimisation problem solving is shown. After the representation of optimisation problem, structural optimisation and the finding of optimal solution of quadratic equation are given.

The practical application for selection, reproduction, crossover, and mutation operation are shown. The functional implementation of GA based optimisation in MATLAB programming language is considered. Also the multi-modal optimisation problem, some methods for global optimisation and the application of Niching method for multi-modal optimisation are discussed.

## CONTENTS

# INTRODUCTION

The GENETIC ALGORITHM [1] 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 strings that are 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.

Genetic algorithms (GAs) seek to solve optimisation problems using the methods of evolution, specifically survival of the fittest. In a typical optimisation problem, there are a number of variables which control the process, and a formula or algorithm which combines the variables to fully model the process. The problem is then to find the values of the variables which optimise the model in some way. If the model is a formula, then we will usually be seeking the maximum or minimum value of the formula. There are many mathematical methods which can optimise problems of this nature (and very quickly) for fairly "well-behaved" problems. These traditional methods tend to break down when the problem is not so "well-behaved." 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 ENVIRONMENT. Some are better than others. Those that are better 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 creation of genetically radically different offspring that are still having the same general flavour (SPECIES). At the molecular level what occurs (wild oversimplification alert!) is that a pair of Chromosomes bump into one another, exchange chunks of genetic information and drift apart. This is the RECOMBINATION operation, which GA/GP errs 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 ENVIRONMENT where the ELECTION 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 operations such as crossover or asexual REPRODUCTION (the ropagation 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 stags 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 election/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 ba a matter of debate (some refer to it as a backgroud operator, while others view it as playing the dominant role in the evolutionary

process). It cannot be stressed too strongly that the GENETIC ALGORITHM (as a SIMULATION of a genetic process) is not a 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).GENETIC ALGORITHMs are used for a number of different application areas.

An example of this would be multidimensional OPTIMIZATION problems in which the character string of the CHROMOSOME can be used to encode the values for the different parameters being optimized. 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 focussed on fixed-length character strings. We should focus on both this aspect of fixed-lengthness 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 a fixed 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 REPRODUCTION 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 to 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 the FITNESS measure, operate to improve the population.

## Description:

Genetic algorithms use a vocabulary[ 1 ] borrowed from natural genetics. A candidate solution is called an individual. Quite often this individual called also truing or chromosome. This might be a little bit misleading; each cell of every organism of a given species carries a certain number of chromosomes, however, we talk about one-chromosome individuals only. Chromosomes are made of units-genes-arranged in linear succession; every gene controls the inheritance of one or several characters

Each gene can assume a finite number of values, called alleys (feature values). In binary representation chromosome is a vector, consisting of the bits succession, ie. the succession of zeroes and ones. A set of chromosomes makes a population. A number of chromosomes in population define a population size. The genetic algorithm evaluates a population and generates a new one iteratively, with each successive population referred to as a generation. The population undergoes a simulated evolution; at each generation the relatively "good" solutions reproduce while the relatively "bad" solutions die. To distinguish between different solutions

we use an objective (evaluation) function, which plays the role of an environment. Quite often the objective function is called also fitness function

# 1. APPLICATIONS OF GENETIC ALGORITHM

## 1.1 Genetic Algorithms For Dynamic Test Data Generation

In software testing, it is often desirable to find test inputs that exercise specific program features. To find these inputs by hand is extremely time-consuming, especially when the software is complex. Therefore, numerous attempts have been made to automate the process. Random test data generation consists of generating test inputs at random, in the hope that they will exercise the desired software features. Often, the desired inputs must satisfy complex constraints, and this makes a random approach seem unlikely to succeed. In contrast, combinatorial optimisation techniques, such as those using genetic algorithms, are meant to solve difficult problems involving the simultaneous satisfaction of many constraints. In this paper, we discuss experiments with a test generation problem that is harder than the ones discussed in earlier literature- we use a larger program and more complex test adequacy criteria. We find a widening gap between a technique based on genetic algorithms and those based on random test generation [3].

## 1.2 Time Dependent Optimization with a Folding Genetic Algorithm

Time dependent optimisation [3] has revealed to be a promising gap for the entire Genetic Algorithms community since it has numerous applications. This paper extends previous work related to the use of meta-genes in the so-called Dual Genetic Algorithms (DGAs). A more generic framework, involving a variable number of genes, Folding Genetic Algorithms are thus

proposed as a new class of genetic algorithms which effectiveness is investigated on two well known models of dynamical environments and compared to Simple Genetic Algorithms and DGAs. Eventually, further analysis of these results enlightens the ability of FGAs to evolve a metrics over the search space (i.e. a kind of encoding scheme) along with potential solutions. These particularly encouraging results open us interesting perspectives as FGAs should be applied to other fundamental problems investigated by the GA community in order to measure the benefits of this really meta level of evolution.

## 1.3 Genetic Algorithm Based Test Generation for Sequential Circuits

In [ 3 ] a new sequential depth measure called sequential element teachability, which may be used as one of parameters of GA is proposed. For fitness functions, we proposed a new dynamic testability measure, which can be evaluated in parallel for multiple individuals. The test generation is divided to three sub-problems: initialisation, test vector generation for a group of faults and test sequence generation for a target fault.

## 1.4 Using Genetic Algorithms to Design Mesh Network

Designing mesh [3] communication networks is a complex, multi-constraint optimisation problem. The design of a network connecting 10 Chinese cities demonstrates the elegance and simplicity that genetic algorithms offer in handling such problems. Designs for mesh

communication networks must meet conflicting, interdependent requirements. This sets the stage for a complex problem with a solution that targets optimal topological connections, routing, and link capacity assignments. These assignments must minimise cost while satisfying traffic requirements and keeping network delays within permissible values. Since such a problem is NP-complete (one which has a solution in polynomial time, but can only be solved by non deterministic algorithms), we must use heuristic techniques to handle the complexity and solve practical problems with a modest number of nodes.

The heuristic methods used to design mesh networks include branch exchange, cut saturation, and Mentor algorithms. Another heuristic technique, genetic algorithms, 1,2 appear ideal to design mesh networks with capability of handling discrete values, multi objective functions, and multi constraint problems.3 Existing applications of genetic algorithms to this problem, 4-6 however, have only optimised the network topology. They ignore the difficult sub problems of routing and capacity assignment, a crucial determiner of network quality and cost. We present a total solution to mesh network design using a genetic algorithm approach. Not only does our method optimise network topology, it also optimises routing and capacity assignment. In the following design for a proposed communications network, genetic algorithms produced a solution that costs 9 percent less and has two-thirds the delay of a typical design method. In our method, each optimisation level uses genetic algorithms as its core, a similarity that reduces the complexity of system design. The

advantages of this approach are not only its elegance and algorithmic simplicity, but also its ability to handle complicated issues such as continuous and discrete link capacities, linear or discrete cost structures, additional constraints, and various constraint models. We believe our genetic-algorithm approach to network design is novel and better than existing methods. Our 10-city network demonstrates that this method can be used for networks of reasonable size with realistic topology and traffic requirement.

## 1.5 Dynamic Phase-Only Array Beam Control Using a Genetic Algorithm

In [ 3 ] two approaches to evolvable antenna array beams are described. The first approach uses a genetic algorithm for adaptive phase-only nulling with phased arrays. A genetic algorithm adjusts some of the least significant bits of the beam steering phase shifters in order to minimise the total output power. Using a few bits for nulling speeds convergence of the algorithm and limits pattern distortions. Various results are presented to show the advantages and limitations of this approach. A second problem is a switched beam linear array in which two beams with specified shapes, a narrow beam and a wide beam, are to be produced. The goal of the design effort is to determine a set of complex excitation coefficients such that switching between beams is accomplished by changes in the phase weights alone. Excellent results are obtained by simultaneous, multi-objective optimisation based design using a GA instead of sequential a GA optimisation for the narrow and wide beam cases individually

## 1.6 Empirical Study Of The İnterdependencies Of Genetic Algorithm Parameters

It is recognised that the performance of evolutionary systems such as genetic algorithms (GAs) is affected by the parameters that are employed to implement them, there is hardly any work known to us that has shed much light on the interdependencies and interactions between these parameters. Most studies on the effects of these parameters on the performance of GA-based systems have focused on a parameter at a time without considering the effect of other parameters on that parameter and vice versa. Consequently there is hardly any theory about the interactions and interdependencies of these parameters. This paper contributes towards correcting the situation mentioned above by examining empirically the relationship between two parameters of genetic algorithms (GAs): population size and replacement methods in the performance of GA-based systems. Results are presented that appear to show a link between replacement strategy and an appropriate population size when applying genetic algorithms to a particular problem. It is suggested that, in the domain of application considered in this paper one can infer that the more individuals that are replaced during reproduction the larger the population size that is needed for all optimum performance of GA-based systems. It is suggested that directing our efforts towards establishing the interdependencies and interactions between parameters of evolutionary systems will enhance the advancement of this new technology [3].

## 1.7 Partitioning and Allocation of Objects in Heterogeneous Distributed Environments Using the Niched Pareto Genetic-Algorithm

As the importance of middleware-based distributed object computing environments (e.g. CORBA and DCOM) increases, there is considerable interest in incorporation of object-orientation (OO) and distributed systems. One important aspect of distributed object systems is effective distribution of software components, to achieve some performance goals, such as balancing the workloads, maximising the degree of concurrency and minimising the entire communication costs. Although there have been a lot of works on partitioning and allocation for distributed system, they are not directly applicable to OO system. We developed a partitioning and allocation model for mapping OO applications to heterogeneous distributed environments, and evaluated it using genetic algorithm (GA). Our model applies the graph-theoretics approach, dealing with a lot of characteristics of OO paradigm. The Niched Pareto GA is adopted to experiment our model because a partitioning and allocation problem is multi objective problem with non-commensurable objectives.

## 1.8 A Genetic Algorithm for Register Allocation

In [3] a new genetic algorithm for register allocation is introduced. A merge operator is used to generate new individual solutions. The number of steps required examining all pairs in the population matrix to generate $n^2$ ($n$ is the population matrix size). Generating an offspring from the parents needs $m$ steps ($m$ number of nodes). The total number of steps

required by the algorithm is $n2\ m$; that is, the genetic algorithm has a linear time complexity in terms of number of nodes. The experimental results show optimal solutions in many of the graphs used for testing.

## 1.9 Task scheduling in distributed computing systems with a genetic algorithm

Scheduling a directed acyclic graph (DAG) that represents the precedence relations of the tasks of a parallel program in a distributed computing system (DCS) is known as an NP-complete problem except for some special cases. Many heuristic-based methods have been proposed under various models and assumptions. A DCS can be classified in two types according to the characteristics of the processors on a network: a distributed homogeneous system (DHOS) and a distributed heterogeneous system (DHES). The paper defines a general model for a DHOS and a DHES and presents a genetic algorithm (GA) to solve the task-scheduling problem in the defined DCS. The performance of the proposed GA is compared with the list-scheduling algorithm in a DHOS and with the one-level reach-out greedy algorithm in a DHES. The proposed GA has shown better performance in various environments than other scheduling methods [3].

# 2. BASIC GENETIC ALGORITHM:

## 2.1 Basics Of Genetic Algorithms.

The three most important aspects of using genetic algorithms are:

(1) Definition of the objective function.

(2) Definition and implementation of the genetic representation.

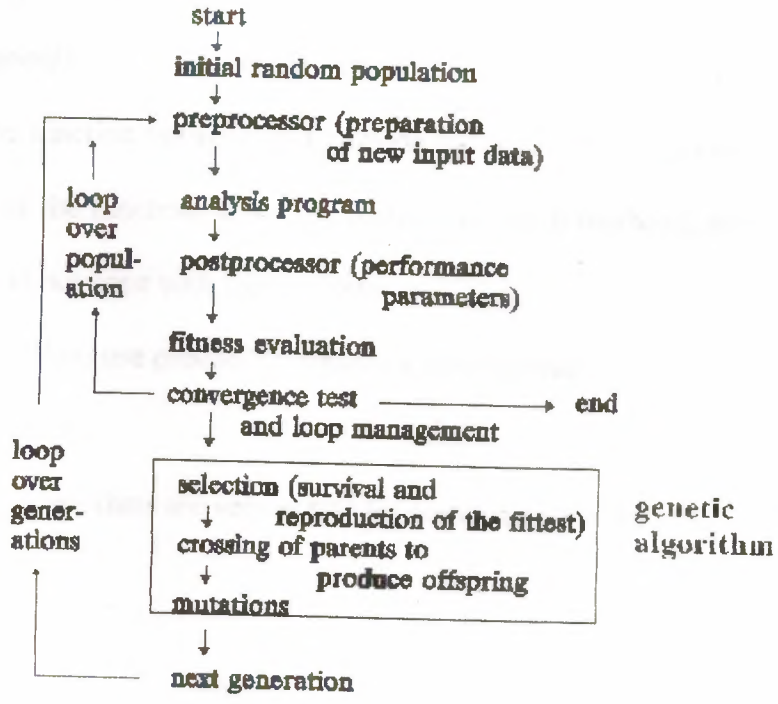
(3) Definition and implementation of the genetic operators. Once these three have been defined.

The generic genetic algorithm should work fairly well. Beyond that you can try many different variations to improve performance, find multiple optima (species - if they exist), or parallels the algorithms.

```
Algorithm GA is
        // start with an initial time
        t := 0;
        // initialize a usually random population of individuals
        initpopulation P (t);
        // evaluate fitness of all initial individuals of population
        evaluate P (t);
        // test for termination criterion (time, fitness, etc.)
        while not done do
    // increase the time counter
     t := t + 1;
        // select a sub-population for offspring production
         P' := selectparents P (t);
```

```
// recombine the "genes" of selected parents
 recombine P' (t);
// perturb the mated population stochastically
      mutate P' (t);
// evaluate it's new fitness
  evaluate P' (t);
// select the survivors from actual fitness
  P := survive P,P' (t);
    od
      End GA.
```

## 2.2 What Does A Genetic Algorithms Do And How?




GAs considers a whole population of tentative solutions and makes it evolve in a fashion which is similar to that in which living species evolve. The population

thus manipulated evolves from one generation to the next, explores the solution space in a very efficient manner and tends to cluster near its optima, with a preference for the global optimum.

## 2.3 Specific Characters Of Genetic Algorithms:

Genetic algorithms (GA)[4] are global optimization algorithms based on the mechanics of natural selection and natural genetics. GA has a number of specific peculiarities by which they differ from the other methods of optimization. These are the following once:

1) Genetic algorithms employ only the objective function, not the derivative one or some other information on the object. It is very convenient in case that the function is neither differentiable nor discrete.

2) Genetic Algorithms employ a parallel multi-point search strategy by maintaining a population of potential solutions, which provides wide information of the function behavior and exclude the possibility of sticking in local extreme of the function, while the traditional search methods, such as gradient, etc, can not cope with this problem.

3) Genetic Algorithms use probability-transitive rules instead deterministic ones.

Besides, Genetic Algorithms are very simple for computer solution.

## 2. 4 The Basic Structure Of Genetic Algorithms:

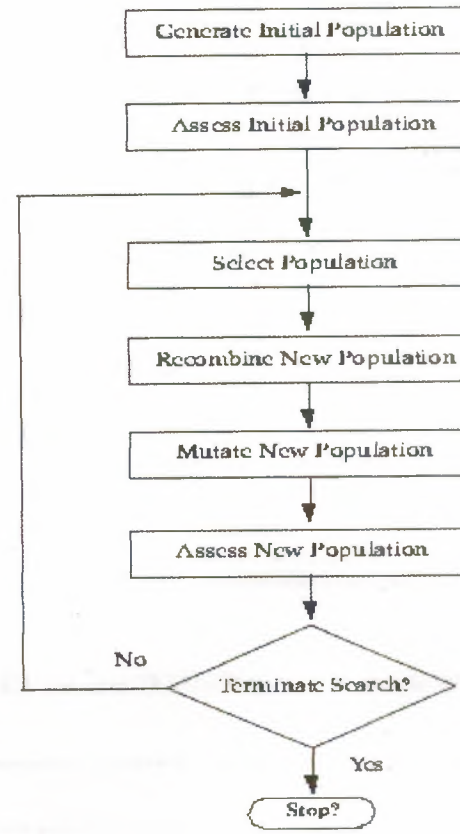



Figure 20: The basic structure of a genetic algorithm.

[5]

## 2.5 Genetic Algorithm Performance

Figure shows the progress [5] of a GA on the two-dimensional Rosen Brock function,

$$f = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

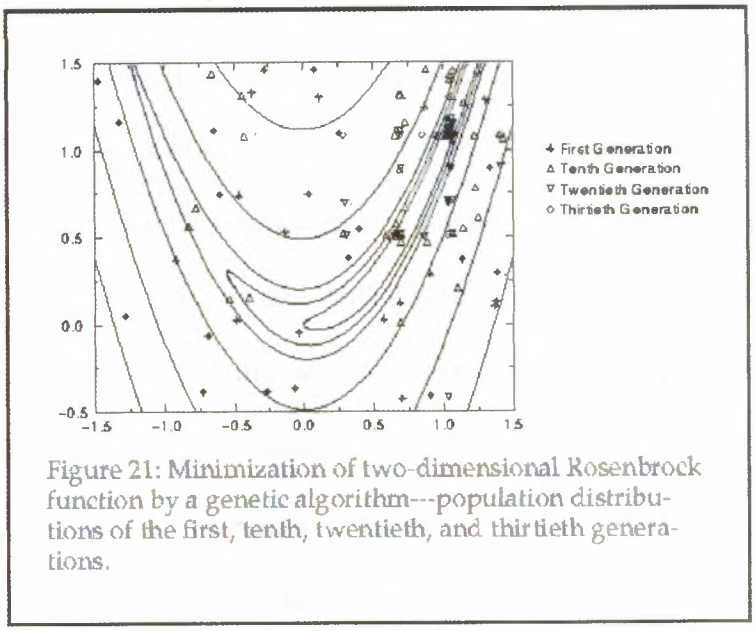



Figure 21: Minimization of two-dimensional Rosenbrock function by a genetic algorithm---population distributions of the first, tenth, twentieth, and thirtieth generations.

This is presented purely for purposes of comparison. Each member of the 1st,

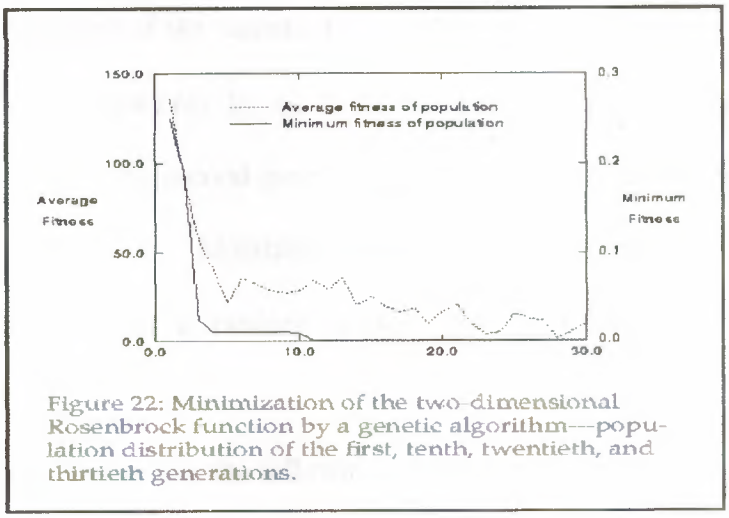



Figure 22: Minimization of the two-dimensional Rosenbrock function by a genetic algorithm---population distribution of the first, tenth, twentieth, and thirtieth generations.

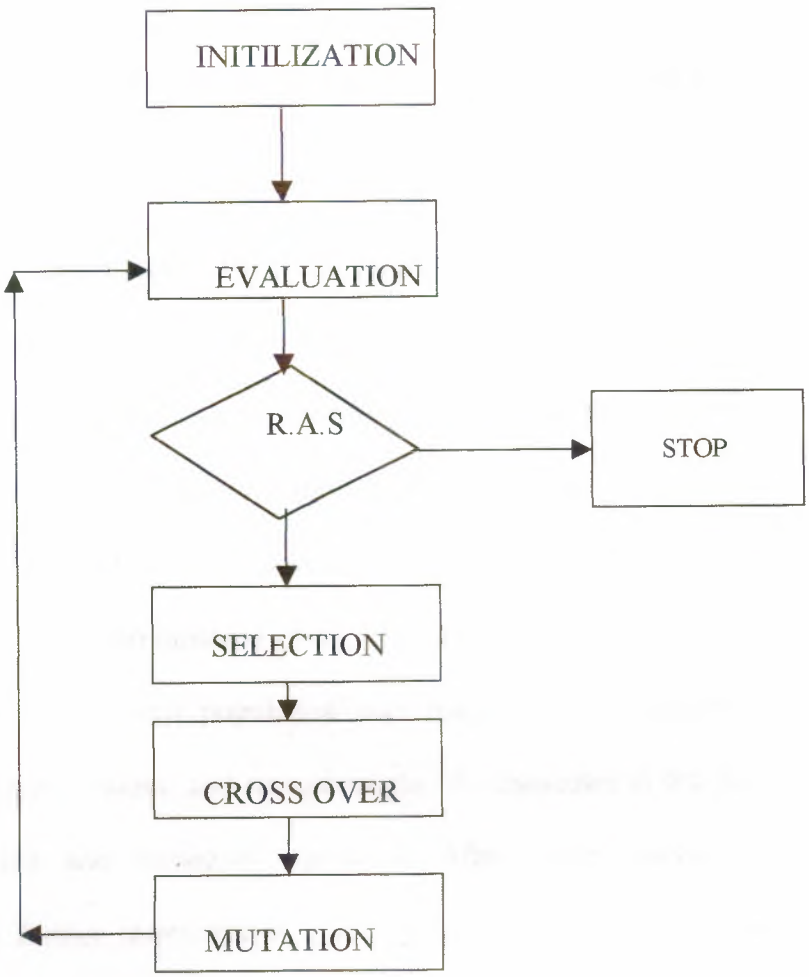
10th, 20th and 30th generations are shown (by a symbol). The convergence of the population to the neighbourhood of the optimum at (1,1) is readily apparent

## 2.6 The Genetic Operators

The initial population is chosen at random. GAs simulates genetic evolution of a population of tentative solutions (individuals) by means of selection and survival of the fittest, crossover and mutations. Every individual is typically represented as a bit sequence, which makes up its "genetic code". The function to be optimised provides "fitness" values. The structure of a simple genetic algorithm is the same as the structure of any volution program. During iteration t, a genetic algorithm maintains a population of potential solutions (chromosomes, vectors), $G(t) = \{ x_1^t , \ldots, x_n^t \}$, Each solution $x_i^t$ is evaluated to give some measure of its ""“fitness""” Then, a new population (iteration t+1) is formed by selecting the more fit individuals. Some members of this new population undergo reproduction by means of crossover and mutation, to form new solutions. Crossover combines the features of two parent chromosomes to form two similar offspring by swapping corresponding segments of the parents. For example, if the parents are represented by five-dimensional vectors $(a_1, b_1, c_1, d_1, e_1)$ and $(a_2, b_2, c_2, d_2, e_2)$, then crossing the chromosomes after the second gene would produce the offspring $(a_1, b_1, c_2, d_2, e_2)$ and $(a_2, b_2, c_1, d_1, e_1)$. Mutation arbitrarily alters one or more genes of a selected chromosome, by a random change with a probability equal to the mutation rate.

For concrete problem GA has the following block-schema .We discuss the actions of a genetic algorithm for a simple parameter optimization problem. Now suppose we wish to maximize a function of k variables, $f(x_1, \ldots, x_k)$: $R^k \rightarrow R$.

If the optimization problem is to minimize a function f, this is equivalent to maximizing a function g, where g=-f, i.e., $\min \{f(x)\} = \max\{g(x)\} = \{-f(x)\}$.




2-1 The structure of a simple GA

Suppose further that each variable x can take values from a domain Di $=[a_i,b_i] \subseteq R$ and f() >0 for all $x_i$ Di. We wish to optimize the function f with some required precision; suppose sex decimal places for the variables' values are desirable.

It is clear that to achieve such precision each domain Di, should be cut into $(b_i - a_i)\cdot 10^6$ equal size ranges. Let us denote by mi the smallest integer such us $(b_i - a_i)\cdot 10^6 \leq 2m_i - 1$ . Then, a representation having each variable $x_I$ coded as a binary string of length mi clearly satieties the precision requirement. Additionally, the following formula interprets each such string: $x_i = a_i$ + decimal(1001///0012) Where decimal ($string_2$) represents the decimal value of that binary string. Now,

each chromosome (as a potential solution) is represented by a binary string of length $m = \sum_{i=1}^{k} m_i$ the first m1 bits map into a value from the range $[a_1, b_1]$, the next group of $m_2$ bits map into a value from the range $[a_3, b_3]$, and so on; the last group of mk bits map into a value from the range $[a_k, b_k]$. To initialize a population, we can simply set some ps number of chromosomes randomly in a bit wise fashion. However, if we have some knowledge about the distribution of potential optima, we may use such information in arranging the set of initial (potential) solutions. The rest of the algorithm is straightforward, in each generation we evaluate each chromosome (using the function f on the decoded sequences of variables), select new population with respect to the probability distribution based on fitness values, and recombine the chromosomes in the new population by mutation and crossover operators. After some number of generations, when no further improvement is observed, the best chromosome represents an (possibly the global) optimal solution. Often we stop the algorithm after a fixed number of iterations depending on speed and resource criteria. For the selection process (selection of a new population with respect to the probability distribution based on fitness values), we must implement the following actions at first; Calculate the fitness value eval(vi) for each chromosome vi (i=1, …, ps).

- Find the total fitness of the population

$$F = \sum_{i=1}^{ps} \text{eval}(v_i)$$

- Calculate the probability of a selection $p_n^2$ for each chromosome vi (i=1, …, ps):

$$p_n^i = \text{eval}(v_i)/F$$

- Calculate a cumulative probability $p^i_{cum}$ for each chromosome vi (i=1, ..., ps):

$$p^i_{cum} = \sum_{j=1}^{i} p^j_n$$

The selection process is implemented ps times; each time we select a single chromosome for a new population in the following way:

- Generate a random (float) number r from the range [0,1].
- If $r < p^i_{cum}$ then select the first chromosome (v1); otherwise select the I-tj chromosome vi ($2 \leq i \leq ps$ ) such that

$$p^{i-1}_{cum} < r < p^i_{cum}$$

Obviously, some chromosomes would be selected more than once; the best chromosomes get more copies; the average stay even, and the worst die off.

Now we are ready to apply the first recombination operator, crossover, to the individuals in the new population. One of the parameters of a genetic system is probability of crossover $p_c$ This probability gives us the expected number $p_c$ ps of chromosomes which undergo the crossover operation. We proceed in the following way:

For each chromosome in the (new) population:

- Generate a random (float) number r from the range [0,1];
- If $r < p_c$ , select given chromosome for crossover;

Now we mate selected chromosomes randomly: for each pair of coupled chromosomes we generate random integer number pos from the range [1,m-1] (m is the total length-number of bits – in a chromosome). The number pos indicate the position of the crossing point. Two chromosomes

$(b_1\ b_2\ \ldots\ b_{pos}\ b_{pos+1}\ \ldots\ b_m)$

$(c_1\ c_2\ \ldots\ c_{pos}\ c_{pos+1}\ \ldots\ c_m)$

are replaced by a pair of their offspring:

$(b_1\ b_2\ \ldots\ b_{pos}\ c_{pos+1}\ \ldots\ c_m)$

$(c_1\ c_2\ \ldots\ c_{pos}\ b_{pos+1}\ \ldots\ b_m)$

The intuition behind the applicability of the crossover operator is information exchange between different potential solutions.

The next recombination operator, mutation, is performed on a bit-by- bit basis. Another parameter of the genetic system, probability of mutation $p_m$, gives us the expected number of mutated bits $p_m \cdot m \cdot ps$. Every bit ( in all chromosomes in the whole population) has an equal chance to undergo mutation i.e. change from 0 to 1d of vice versa. So we proceed in the following way.

For each chromosome in the current (i.e., after crossover) population and for each bit within the chromosome;

- Generate a random (float) number r from the range [0,1];
- If $r < p_m$ mutate the bit.

The intuition behind the mutation operator is the introduction of some extra variability into the population.

Following selection, crossover, and mutation, the new population is ready for its next evaluation. This evaluation is used to build the probability distribution (for the next selection process). The rest of evolution is just cyclic repetition of the above steps.

However, as it frequently occurs, in earlier generations the fitness values of some chromosomes are better than the value of the best chromosome after a finite number of generations.

It is relatively easy to keep track of the best individual in the evolution process. It is customary (in genetic algorithms implementations) to store "the best ever" individual at a separate location; in that way, the algorithm would report the best value found during the whole process (as opposed to the best value in the final population).

It is necessary to note, that classical GA may employ roulette wheel method for selection, which is a stochastic version of the survival-of-the fittest mechanism. In this method of selection, candidate strings from the current generation G(t) are selected to survive to the next generation G(t=1) by designing a roulette wheel where each string in the population is represented on the wheel in proportion to its fitness value. Thus those strings, which have a high fitness, are given a large share of the wheel, while those strings with low fitness are given a relatively small portion of the roulette wheel. Finally, spinning the roulette wheel ps times and accepting as candidates those strings, which are indicated at the completion of the spin, make selections.

Example 4.1 As an example, suppose ps=5, and consider the following initial population of strings; G (0)= {(10110),(11000),(11110),(01001),(00110)}. For each string $v_i$ in the population, the fitness may be evaluated: eval ($v_i$). The appropriate share of the roulette wheel to allot the i-th string is obtained by dividing the fitness of the i-th string by the sum of the fatnesses of the entire population:

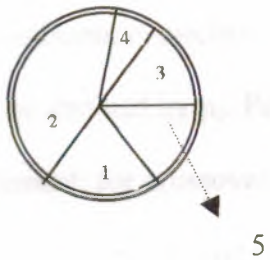
$$\frac{\text{eval}(v_i)}{\sum_{i=1}^{ps} \text{eval}(v_i)}$$

Figure 2-2 shows a listing of the population with associated fitness values and the corresponding roulette wheel.

To compute the next population of strings, the roulette wheel is spun five times [3]. The strings chosen by this method of selection, though, are only candidate strings for the next population. Before actually being copied into the new population, these strings must ungergo crossover and mutation.

| | String $V_i$ | Fitness eval($v_i$) | Relative Fitness |
|---|---|---|---|
| $V_1$ | 10110 | 2.23 | 0.14 |
| $V_2$ | 11000 | 7.27 | 0.47 |
| $V_3$ | 11110 | 1.05 | 0.07 |
| $V_4$ | 01001 | 3.35 | 0.21 |
| $V_5$ | 00110 | 1.69 | 0.11 |

(a)




(b)

In figure A listing of the five-string population and the associated fitness values. (b) Corresponding roulette wheel for string selection.

The integers shown on the roulette wheel correspond to string labels.

| | | |
|---|---|---|
| 110101 | 1101 \| 01 | 110100 |
| 100100 | 1001 \| 00 | 100101 |
| (a) | (b) | (c) |

An example (figure) of a crossover for two 6-bit strings.

(a) Two strings are selected for crossover.

(b) A crossover site is selected at random. In this case, k =4.

(c) Now swap the two strings after the k- th bit.

Pairs of the ps (assume ps even) candidate strings, which have survived selection, are next chosen for crossover, which is a recombination mechanism. The probability that the crossover operator is applied will be denoted by $p_c$. Pairs of string are selected randomly from G (t), without replacement, for crossover. A random integer k, called the crossing site, is chosen from {1,2, ... m-1}, and then the tits from the two chosen strings are swapped after the k-th bit with a probability pc. This process is repeated until G (t) is empty. For example, Figure 11.3. Illustrates a crossover for two 6-bit strings. In this case, the crossing site k is 4, so the bits from the two strings are swapped after the fourth bit.

Finally, after crossover, mutation is applied to the candidate strings. The mutation operator is a stochastic bit-wise complementation applied with uniform probability $p_m$. That is, for each single bit in the population, the value of the bit is flipped from 0 to 1 or from 1 to 0 with probability $p_m$. As an example, suppose $p_m$=0. 1, and the string v=11100 is to undergo mutation. The easiest way to determine which bits, if any, to flip is to choose a uniform random number $r \in [0,1]$ for each bit in the string. If $r \leq p_m$, then the bit is flipped; otherwise, no action is taken. For the string v above, suppose the random numbers (0.91, 0.43, 0.03,0.67,

0.29) were generated, and then the resulting mutation is shown bellow. In this case, the third bit was flipped.

Before mutation: 11100

After mutation: 11000

After mutation, the candidate strings are copied into the new population of strings G (t+1), and the whole process is repeated [4]

## 2.7 General Properties of Genetic Algorithms:

GAs provide an efficient, robust, non-exhaustive way of exploring virtually any parameter space where a real single-valued function is defined, in pursuit of its global optimum. GAs is useful in finding the "best" values of virtually any continuous or discontinuous function defined over a set of continuous or discrete parameters in the presence of multiple local optima. GAs process good partial bit sequences with exponentially increasing trials in subsequent generations (fundamental theorem). GAs explore a number of partial bit sequences which is proportional to the cube of the population size (implicit parallelism).

There is no guarantee that the GA will actually find the global optimum (or that it will find all global optima, if there are more than one), since the whole method is based on probabilities. But, given enough time, it will find the global optimum in a much shorter time than that necessary for an exhaustive search, with a very high probability. From an engineering point of view, a GA will find the best solutions available that can be found within the allotted time. [1]

## 2.8 Control Parameters

The efficiency of a GA is highly dependent on the values of the algorithm's control parameters. Assuming that basic features like the selection procedure are predetermined, the control parameters available for adjustment are:

- The population size *N*,
- The crossover probability P*c* , and
- The mutation probability *Pm*.

De Jong made some recommendations based on his observations of the performance of GAs on a test bed of 5 problems, which included examples with difficult characteristics such as discontinuity's, high dimensionality, noise and multi modality. His work suggested that settings of

(N, Pc, Pm) = (50,0.60,0.001) 2-1

Would give satisfactory performance over a wide range of problems.

Grefenstette went one stage further and used a GA to optimise these parameters for a test bed of problems. He concluded that

(N, Pc, Pm) = (30,0.95,0.010) 2-2

Resulted in the best performance when the average fitness of each generation was used as the indicator, while

(N, Pc, Pm) = (80,0.45,0.010) 2-3

Gave rise to the best performance when the fitness of the best individual member in each generation was monitored. The latter is, of course, the more usual performance measure for optimisation routines. [7]

## 2.9 Genetic Algorithms Module

The Genetic Algorithms Module was created for the purpose of giving beginning students a brief introduction to genetic algorithms. It is divided into five sections: Introduction, Advanced Topics, Research, Applications, and Resources.

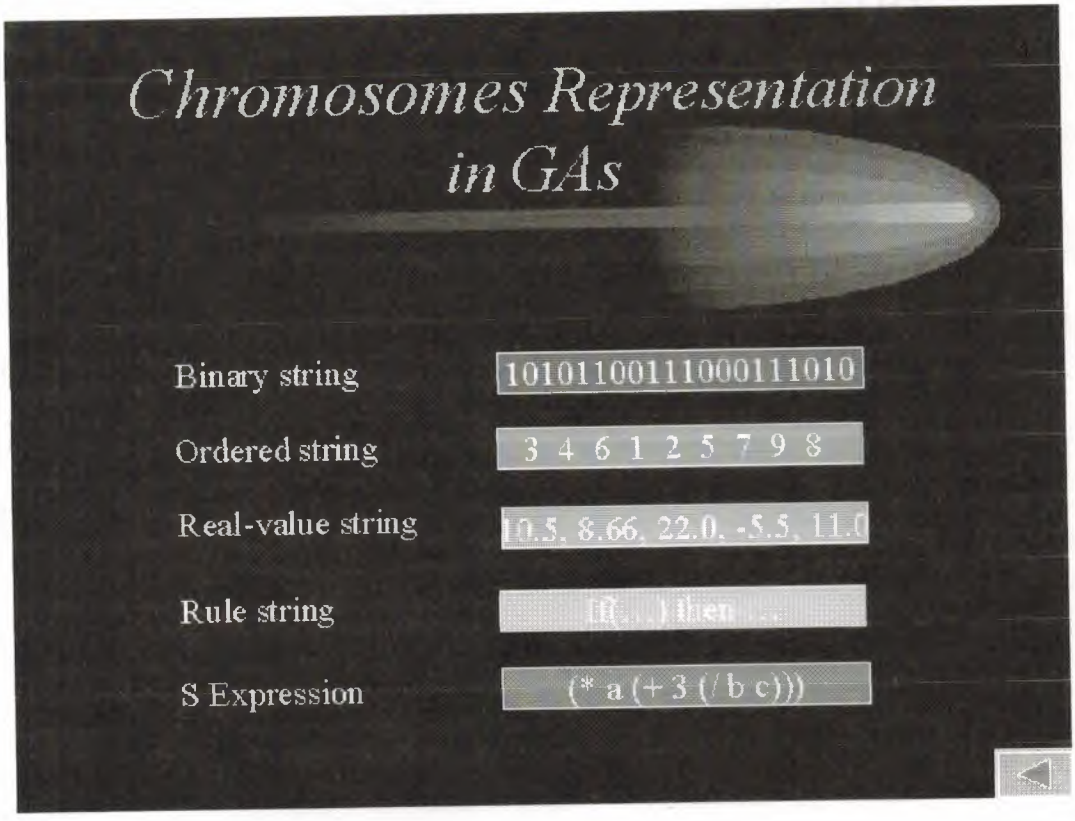
We suggest that you go through the Introduction section first because this is the section geared to familiarising you to genetic algorithms. Then as you become

more familiar with the terms and concepts being presented, you may proceed to the Advanced Topics Section Genetic Algorithms (GAs) are based on the Theory of Evolution - 'survival of the fittest'. In nature, individuals that are fit are more likely to breed and pass their characteristics on to future generations. Genetic Algorithms take their cue from Nature, modelling complex and difficult to solve problems as genetic objects. Genetic Algorithms breed solutions. An initial population is built using individuals representing random solutions. Each subsequent population that is built (each subsequent *generation*) uses the previous population as a base - taking the more fit individuals to breed better solutions. These algorithms have been used in the past to help solve very complex problems not easily solved using standard, problem-specific methods.

The one advantage that GAs have over problem-specific solutions it that problem-specific solutions are...problem-specific. A solution for one problem may not apply to another - the new and possibly completely different solution would have to be developed. They are 'context specific'. With GAs, because they are so generic, very little work must be done to apply a GA to a completely different problem. [8]

## 2.10 Chromosome Representation In Genetic Algorithms:

In genetic algorithm chromosomes are represented as




To use a genetic algorithm, you must represent a solution to your problem as a *genome*. The genetic algorithm then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s) [9].

## 2.11 All the 'SHUNS' - Selection, Reproduction, and Mutation

The concepts behind GAs are anything but complex:

### 2.11.1 Selection

Selection [10] is the process of determining which individuals are chosen to reproduce from one generation to the next. A more fit individual has a higher probability of reproduction over a less fit one.

### 2.11.2 Reproduction

Every individual is *defined* by its genetic information - stored in nature using a DNA strand. When two individuals reproduce, the resulting individual's DNA reflects some of the information from each of its parents.

After having been selected, two individuals swap genetic material to create 'offspring'. The idea is that, through this swapping of material, even two relatively average individuals can create even more fit offspring.

In the following example, the parents reproduce and swap genetic information to create their offspring:

Offspring 1 oooooooo

|

---

| |

**** ****

Parent A xxxxoooo Parent B ooooxxxx

**** ****

| |

---

|

Offspring 2 xxxxxxxx

### 2.11.3 Mutation

Over time, all of the individuals remaining in the population may have lost a specific attribute. Mutation [10] allows for the reintroduction of attributes, by randomly altering the characteristics of an individual.

The following individual shows the effects of mutation:

Before xxxxxxxx

after xxxxoxxx

, And the implementation of it, have been kept as simple as possible. Although there are many possible ways to improve the *performance* of the algorithm, they have been ignored in favour of *simplicity*. This is a gentle introduction, after all.

### 2.11.4 Individual

The chromosomes of an individual [10], in our implementation (as well as in Nature) contain all of the genetic information for it. You manipulate an individual by altering its chromosomes, whether that be through reproduction or mutation.

In our example, a 32-character string, containing only the characters $0 and $1, represents the chromosome. Our initial population will be seeded with random individuals, created as follows:

GA Individual new chromosome:

((RandomNumberGenerator between: 0 and: (2 raisedTo: 32) - 1) printStringRadix: 2 padTo: 32).

Note: RandomNumberGenerator is a class provided to you if you care to download the 'proof of concept' code. Strictly a 'class of convenience'.

The 'fitness' of an individual represents how well that individual solves the problem at hand. This, along with how the chromosome is built, is the variant for different problems to which the GA can be applied.

In our case, the fitness of the individual is determined by treating the chromosome as a binary string. The value of this binary string is the fitness of the individual.

GAIndividual>>fitness

"Return the fitness of the individual."

| base anInteger |

```
base := 2.
anInteger := 0.
self chromosome
doWithIndex:
[ :eachCharacter :index |
anInteger := anInteger * base + eachCharacter digitValue ].
^anInteger
```

### 2.11.5 Population

A population is a collection of individuals, representing a specific 'generation'. The population determines which individuals are to be selected for reproduction, with an increasing probability of selection going to the more fit individuals.

### 2.11.6 Controller

The controller guides the process - building the next generation, breeding and mutating individuals taken from the previous generation.

## 2.12 Summary

It is through this interaction, between relatively simple objects, that better and better solutions are bred.

- Random individuals (representing solutions to the problem at hand) are created.
- An initial population is created, containing these random individuals.
- The population is added to an instance of the controller, as the initial generation.

The controller repeatedly builds generations, based on the previous, with each new generation (hopefully) containing individuals that are progressively better that the last.

# 3.GENETIC ALGORITHM BASED OPTIMIZATION

## 3.1 Optimisation based on Genetic Algorithms:

Genetic algorithms [11] were formally introduced in the United States in the 1970s by John Holland at University of Michigan. The continuing price/performance improvements of computational systems have made them attractive for some types of optimisation. In particular, genetic algorithms work very well on mixed (continuous *and* discrete), combinatorial problems. They are less susceptible to getting 'stuck' at local optima than gradient search methods. But they tend to be computationally expensive.

To use a genetic algorithm, you must represent a solution to your problem as a genome (or chromosome). The genetic algorithm then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s).

This presentation outlines some of the basics of genetic algorithms. The three most important aspects of using genetic algorithms are:

(1) Definition of the objective function.

(2) Definition and implementation of the genetic representation.

(3) Definition and implementation of the genetic operators.

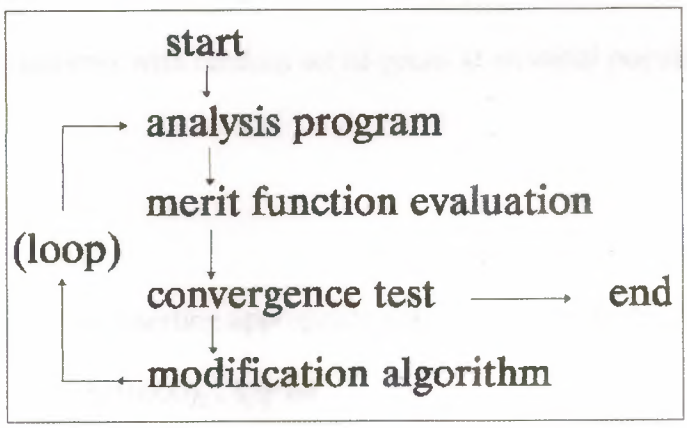
Once these three have been defined, the generic genetic algorithm should work fairly well. Beyond that you can try many different variations to improve performance, find multiple optima (species - if they exist), or parallelism the algorithms.

Genetic algorithm (GA) uses the principles of evolution, natural selection, and genetics from natural biological systems in a computer algorithm to simulate evolution. Essentially, the genetic algorithm is an optimization technique that

performs a parallel, stochastic, but directed search to evolve the most fit population. In this section we will introduce the genetic algorithm and explain how it can be used for design and yuning of fuzzy systems.

The genetic algorithm borrows ideas from and attempts to simulate Darwin's theory on natural selection and Mendel's work in genetics on inheritance. The genetic algorithm is an optimisation technique that evaluates more than one area of the search space and can discover more than one solution to a problem. In particular, it provides a stochastic optimisation method where if it "gets stuck" at a local optimum, it tries to simultaneously find other parts of the search space and 'tjump out" of the local optimum to a global one.

**3 -1 Characteristics common to all optimisers**



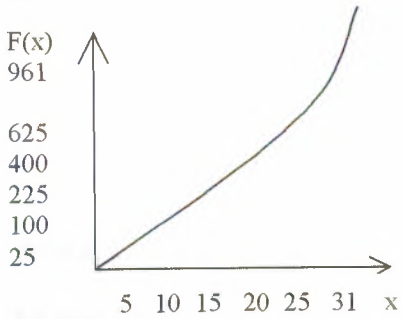

## 3.2 Main Features for Optimisation

- Probabilistic algorithms for searching and optimisation
- Mimic natural evolution process
- Capable of handling non-linear, non-convex problem
- Optimisation procedure does not require differentiation operation

- Capable of locating global and local optima within search domain

Optimisation is based on population instead of a single point.

Let's consider a simple problem of maximization of the function F (x)=$x^2$, where x∈[0,31] (see Fig).




In order to use GA we should first code variables in to bit strings as any integer number between 0 to 31 may be represented in binary number 5 symbols between (00000)=0 and (11111)=31 the length of chromosomes will be five.

Lets take 4 chromosomes with random set of genes as an initial population as:

01101
11000
01000
10011

Then define their fitness inserting appropriate real values into the function as:

$$f(01000)_2 = f(8) = 64$$

Thus the fitness of all individuals in calculated then accordance with the formula

$$P^i_5 = f_i / \Sigma^I_{j=1} F^i_5, \qquad i = 1..4$$

Survival probability for each individual is calculated where as cumulative

$$P^i_{cum} = \Sigma^I_{j=1} P^i_5 \qquad i = 1..4$$

Let's enter all the calculated values in table

| Initial Population | Their integer values | $F(x)=x^2$ | $P_5$ | $P_{cum}$ | Num after Selection |
|---|---|---|---|---|---|
| 01101 | 13 | 169 | 0.14 | 0.14 | 1 |
| 11000 | 24 | 576 | 0.49 | 0.63 | 2 |
| 01000 | 8 | 64 | 0.06 | 0.69 | 0 |
| 10011 | 19 | 361 | 0.31 | 1 | 1 |
| Average | | 293 | 0.25 | | 1 |
| Maximum | | 576 | 0.49 | | 2 |
| Sum | | 1170 | 1 | | 4 |

For the selection process we generate 4 random numbers from the range [0,1]. Suppose, that we generated 0.1; 0.25; 0.5 and 0.8.

Comparing these values with cumulative probabilities, we obtaining the following

$$0.1 < P^i_{cum}$$

$$P^1_{cum} < 0.25 < P^2_{cum}$$

$$P^1_{cum} < 0.5 < P^2_{cum}$$

$$P^3_{cum} < 0.8 < P^4_{cum}$$

Looking at the right sides of these inequalities, one can easily see, that the first and the fourth chromosomes have passed the selection, each four taking a place in the new generation, the second chromosome-the most highly fitted has got 2 copies, while the third one did not survive at all. These indices are written in the right column of table 3-1.

Then crossover operation is applied. If the probability of crossover Pc=1 is given, it means that, 4.1=4 chromosomes will participate in crossover process.

Let's choose them at random. Suppose that the first and the second strings mate from crossover point 4, and the third with the forth-from crossover point 2.

0110|1
1100|0
11|000
00|011

| Population after selection | Crossover | New Population | Value of X | $F(x)=X^2$ |
|---|---|---|---|---|
| 01101 | 4 | 01100 | 12 | 144 |
| 11000 | 4 | 11001 | 25 | 625 |
| 11000 | 2 | 11011 | 27 | 729 |
| 10011 | 2 | 10000 | 16 | 256 |

Table 3-1

| | |
|---|---|
| Average | 439 |
| Maximum | 729 |
| Sum | 1754 |

Having compared both of the tables we see, for ourselves, that the population fitness is improved and we have come close to the solution. The next recombination operator, mutation, is performed on a bit-by-bit basis. If Pm=0.05 is given, it means that only one of twenty bits in population will be changed: 20· 0.05=1. Suppose, that the third bit of the fourth string undergoes mutation. I.e.

$x_4$=10100. Repeating these operations in a finite number of generations we will get the chromosome (11111) corresponding to problem optimal solution. It is necessary to mention, that GA is especially effective for multi-extreme problems in the global solution search process. For example, if the junction is of the type shown. It is rather difficult to find its global maximum by means of traditional methods. Suppose that the junction is defined as [1]

$$f(x)=x\cdot \sin(10\pi\cdot x)+1.$$

The problem is to find x from the range [-1,2], which maximizes the function f, i.e., to find $x_0$ such that

$$f(x_0) \geq f(x), \text{ for all } x\in[-1,2].$$

It is relatively easy to analyze the junction f. The zeros of the first derivative f' should be determined

$$F(x)=\sin(10\pi\cdot x)+ 10\pi\cdot x\cdot \cos(10\pi\cdot x)=0$$

The formula is equivalent to

$$\text{Tan}(10\pi\cdot x)= (10\pi\cdot x)$$

It is clear that the above equation has an infinite number of solutions,

$$x_i=\frac{2i-1}{20}+\xi_i, \qquad \text{for i=1, 2, ...,}$$

$$x_0 = 0,$$

$$x_i=\frac{2i+1}{20}+\xi_i, \qquad \text{for i= 1, -2, ...,}$$

Where terms $\xi_1$ represent decreasing sequences of real numbers (for i= 1, 2, ..., and i=-1, -2, ... ) approaching zero. Note also that the function f reaches its local maximal for x, it i is an odd integer, and its local minima for x, if i is an even

integer. Since the domain of the problem is $x \in [-1,2]$, the function reaches its maximum for

$$X_{19} = \frac{37}{20} + \xi_{19} = 1.85 + \xi_{19},$$

Where $f(x_{19})$ is slightly larger than

$$F(1.85) = 1.85 \cdot \sin(18\pi + \pi/2) = 1.0 = 2.85.$$

Assume that we wish to construct a genetic algorithm to solve the above problem, i.e., to maximize the function f. let us discuss the major components of such a genetic algorithm in turn. [4]

### 3.2.1 Representation.

We use a binary vector as a chromosome to represent real values of the variable x. The length of the vector depends on the required precision, which, in this example, is six places after the decimal point. The domain of the variable x has length 3; the precision requirement implies that the range [-1,2] should be divided into at least $3 \cdot 1000000$ equal size ranges. This means that 22 bits are required as a binary vector (chromosome):

$$2097152 = 2^{31} < 3000000 \leq 2^{33} = 4194304$$

The mapping from a binary ($b_{21}\ b_{20}\ b_0$) string into a real number x from the range [-1,2] is straightforward and is completed in two steps:

- Convert the binary string ($b_{21}\ b_{20}\ b_0$) from the base 2 to base 10:

$$(<b_{21}\ b_{20}\ b_0>)_2 = \left( \sum_{i=0}^{21} b_i \cdot 2^i \right)_{10} = x$$

- Find a corresponding real number x:

$$x = -10 + x \cdot \frac{3}{2^{22} - 1},$$

Where –1,0 is the left boundary of the domain and 3 is the length of the domain.

For example, a chromosome

(10001011101101010001111)

Represents the number 0.637197, since

$x'=(10001011101101010001111)_3 = 2288967$ and

$$x=1.0+2288967 \cdot \frac{3}{4194303} = 0.637197.$$

Of course, the chromosomes

(00000000000000000000) And (11111111111111111111)

Represent boundaries of the domain-1.0 and 2.0, respectively.

Initial population. The initialization process is very simple; we create a population of chromosomes, where each chromosome is a binary vector of 22 bits. All 22 bits for each chromosome are initialized randomly.

Evaluation function. Evaluation function eval for binary vectors v is equivalent to the function f:

Eval (v)=f(x),

Where the chromosome represents the real value x.

As noted earlier, the evaluation junction plays the role of the environment, rating potential solutions in terms of their fitness. For example, three chromosomes:

$V_1$=(10001011101101010001111)

$V_2$=(00000011100000000010000)

$V_3$=(11100000001111111000101)

Correspond to values x=0.637197, x=0.958973 and $x_3$=1.627888, respectively.

Consequently, the evaluation function would rate them as follows:

$$\text{Eval}(v_1)=f(x_1)=1.586345$$

$$\text{Eval}(v_2)=f(x_2)=0.078878$$

$$\text{Eval}(v_3)=f(x_3)=2.250650$$

Clearly, the chromosome $v_3$ is the best of the three chromosomes, since its evaluation returns the highest value.

During the reproduction phase of the genetic algorithm we would use two classical genetic operators; mutation and crossover.

As mentioned earlier, mutation alters one or more genes (positions in a chromosome) with a probability equal to the mutation rate. Assume that the fifth gene from the $v_3$ chromosome was selected for a mutation. Since the fifth gene in this chromosome is 0, it would be flipped into 1. So the chromosome $v_3$ after this mutation would be

$$V_3=(11101000001111111000101)$$

This chromosome represents the value $x_3=1.721638$ and $f(x_3)=-0.082257$. This means that this particular mutation resulted in a significant decrease of the value of the chromosome $v_3$. On the other hand, if the $10^{th}$ gene was selected for mutation in the chromosome $v_3$ them

$$V_3=(11100000011111111000101)$$

The corresponding value $x_3=1.630818$ and $f(x_3)=2.343555$, an improvement over the original value of $f(x_3)=2.250650$.

Let us illustrate the crossover operator on chromosomes $v_2$ and $v_3$. Assume that the crossover point was (randomly) selected after the 5th gene:

$$V_2=(00000 \mid 01110000000010000)$$

$$V_3=(11100 \mid 00000111111000101)$$

The two resulting offspring are

$$V'_2=(00000\,|\,00000111111000101)$$

$$V'_3=(11100\,|\,01110000000010000)$$

These offspring evaluate to

$$F(v'_2)=f(-0.998113)=0.940865.$$

$$F(v'_3)=f(1.666028)=2.459245$$

Note that the second offspring has a better evaluation than both of its parents.

## Parameters.

For this particular problem we have used the following parameters population size ps=50, probability of crossover $p_c$ =0.25, probability of mutation $p_m$ =0.01. The following section presents some experimental results for such a genetic system.

## Experimental results.

we provide the generation number for which we noted an improvement in the evaluation function together with the value of the function. The best chromosome after 150 generations was

$$V_{max}=(11110011010001000001 01),$$

Which corresponds to a value $x_{max}=1.850773$

AS expected, $x_{max}=1.85+\xi$ and $f(x_{max})$ is slightly larger than 2.85.

$\{\} \rightarrow \{f(x)\}\ x_1^t$ [1] | ∈<≥ · ξ π

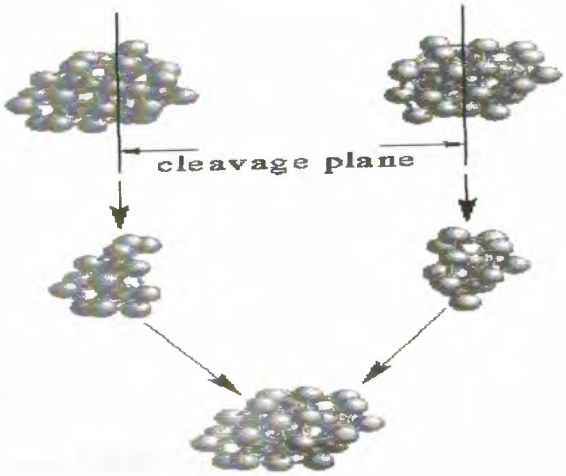
| Generation Number | Fitness Function |
|---|---|
| 1 | 1.441942 |
| 5 | 2.250003 |
| 8 | 2.250283 |
| 9 | 2.250284 |
| 10 | 2.250363 |
| 12 | 2.328077 |
| 36 | 2.344251 |
| 40 | 2.345087 |
| 51 | 2.738930 |
| 99 | 2.849246 |
| 137 | 2.850217 |
| 145 | 2.850227 |

## 3.3 Genetic Algorithm Structural Optimization

Atomistic models of materials can provide accurate total energies. For problems where the structures are not known, however, discovering the lowest energy geometry is difficult. This is particularly true for atomic clusters, whose structure may vary dramatically with a small change in the number of atoms. For this type of problem, the number of possible stable structures increases exponentially fast with the number of atoms. Furthermore, there is considerable experimental difficulty in determining the structure of an atomic cluster. We have been able to address this problem using a novel approach to applying genetic algorithms. The Darwinian evolution process inspires these algorithms. A population of structures is maintained, and "mating" structures and selecting out the lowest energy geometries produce new generations.

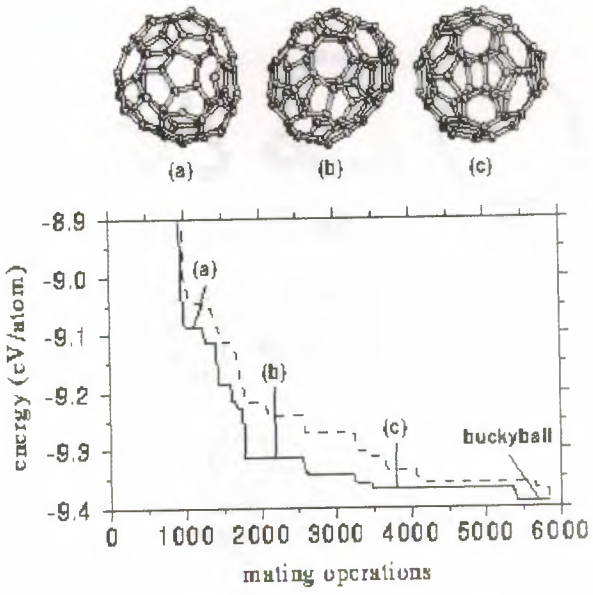
The key to a successful genetic algorithm is to design a mating process that allows for the good parts of the parent structures to be inherited by the next generation. Such a process allows for efficient searching of the possible stable structures. A

poor mating algorithm is no better than a random search. We have designed a new mating process, depicted at left. Two structures are chosen as "parent" structures. Each one is divided into two halves by a cleavage plane. A new structure is generated by connecting half of each parent into a new cluster, followed by atomic relaxation to a local minimum. We have successfully applied our "cut and paste" approach to a number of challenging problems, including: [12]




## 3.4 Genetic Algorithm Optimization of Carbon Clusters

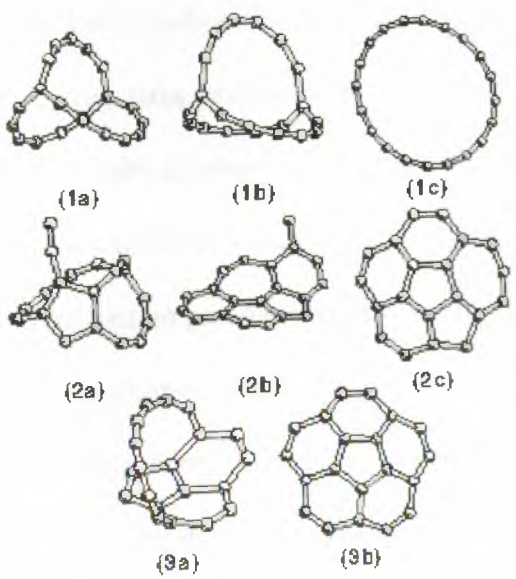
The first application of our genetic algorithm was the test case of carbon clusters,



or "Bucky balls." These structures were quite surprising when originally discovered. Traditional computational approaches are unable to correctly predict these structures without prior knowledge: from a random geometry, simulated annealing is too inefficient to locate the ground state $C_{60}$ structure.

Our first goal, therefore, was to see if the genetic algorithm could generate the correct structure for $C_{60}$ from random initial conditions. The results are shown at left. In this figure, the lowest energy of a structure in a given generation is shown as a solid line, and the highest energy as a dashed line. For approximately the first 1000 generations, the clusters are primarily disordered, high-energy structures. After 1000 generations have passed, cage-like structures with large holes or other defects dominate the population, as seen in structure (a). After 2000 generations, the structure still contains some 7-fold rings (b). Another 2000 generations occur before only 5- and 6-fold rings are found (c). Eventually, near 6000 generations, the ground-state structure is observed.

Optimising the $C_{20}$ structure presented some new problems. The genetic algorithm tended to produce structures such as those shown in fig. 1a-1c at



right. Very quickly, ring-like structures dominated the population. Once this occurred, only ring-like structures would be generated. This lack of diversity was overcome by the use of mutation operators. Essentially, occasionally one member of the population would be "scrambled" into an entirely new structure. This allowed for the formation of the lowest energy, "capped" structure. Typical progressions to this structure are shown in figs. 2 and 3 at right. [12]

## 3.5 Multilevel Fuzzy Process Control Optimised By Genetic Algorithm

The new method for complex processes control with the coordinating control unit based upon a genetic algorithm has been described. The algorithm for control of not well-known complex processes controlled by PID and fuzzy regulators on the first level and coordinating unit on the second level has been theoretically laid out. A genetic algorithm and its application to proposed control method have been described in details. The idea has been verified experimentally and by simulation in two-stage laboratory process. Minimal energy spent criteria restricted by given process response limitations has been applied, and improvement in relation to other known optimising methods has been found. Independent and non-coordinating PID and fuzzy regulator parameter tuning has been performed using a genetic algorithm and the results achieved are the same or better than with traditional optimising methods while at the same time proposed method can be easily automated. Genetic algorithm parameters appropriate for the application described has been defined. Multilevel coordinated control using a genetic algorithm applied to a PID and a fuzzy regulator has been researched. The results of various traditional optimising methods have been compared with an independent non-coordinating control and multilevel coordinating control using

genetic algorithm. The best results have been achieved with the multilevel coordinating fuzzy control optimized by genetic algorithm. Process visualization using control plane is proposed. Inherent adaptability of proposed multilevel coordinating fuzzy control method has been found out as the consequence of evolutionary tuning method [13].

### 3.5.1 Encoding and Decoding

Binary encoding, decoding strategy.

e.g. encoding:

$x_1 = 12(dec) = 1010(bin)$

$x_2 = 5(dec) = 0101(bin)$

after one point cross-over at the middle of strings, and decoding, we get:

$x'_1 = 1001(bin) = 9(dec)$

$x'_2 = 0110(bin) = 6(dec)$

## 3.6 Procedures Involved in Genetic Algorithms Optimization

- Production of initial population.
- 1. Individual fitness evaluation.
- 2. Genetic Operation
    - Reproduction/Selection
    - Crossover/Mating
    - Mutation
- 3. Produce new generation.

- 4. Repeat from step 1. till some stop criteria is met.

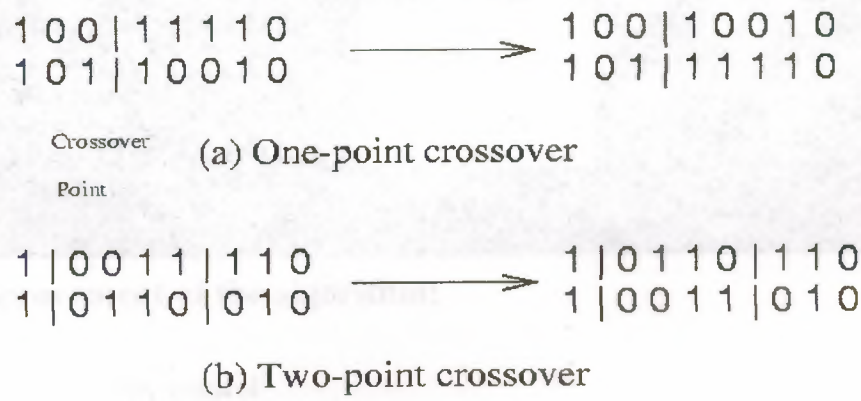
## 3.7 Procedures

### 3.7.1 Selection/Reproduction

Creates new population from old population biased towards the highest fitness.

### 3.7.2 Crossover/Mating

Swaps chromosome parts between individuals.




(a) One-point crossover

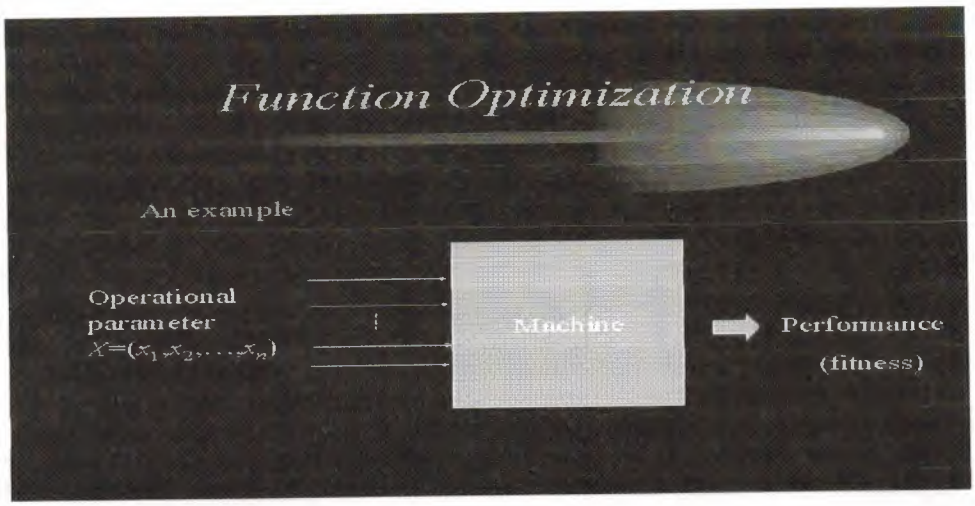



(b) Two-point crossover

## 3.8 Genetic Optimization Method

$$y = Fitness(x) \quad (1)$$

$$x_{optimal} \subset \{x | y = max[Fitness(x)]\} \quad (2)$$




### 3.8.1 Improvement of the algorithm:

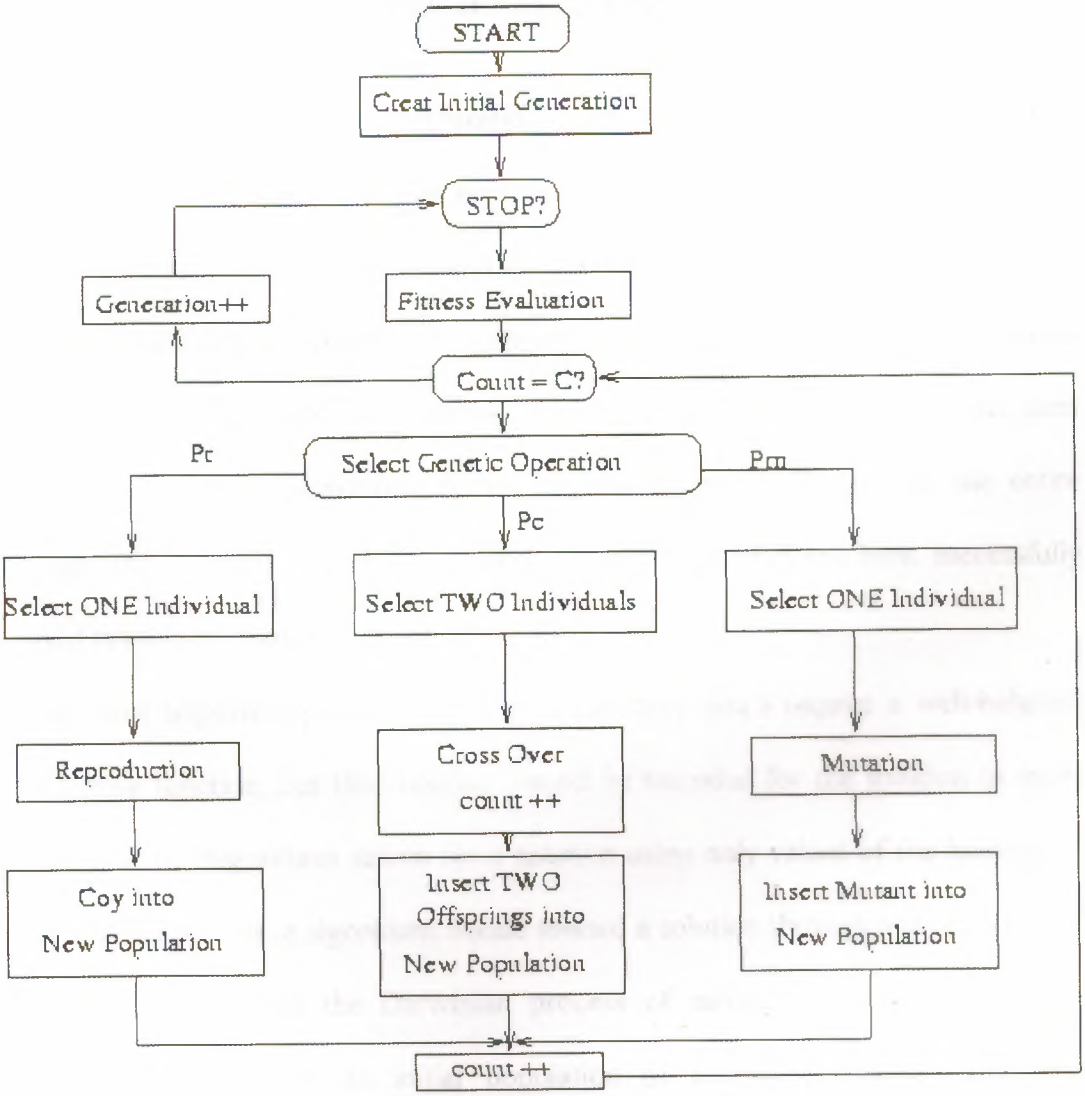
1. Mutation probability control
2. Elitism strategy
3. Termination criteria

### 3.8.2 Mutation probability ($P_m$) control

- $P_m$ is decreased with increasing of generation numbers.
- Concept is from decreased 'temperature' in Simulated Annealing.

**Figure 2:** The decreasing trajectory of mutation probability

| | High $P_m$ | Low $P_m$ |
|---|---|---|
| Stage | Beginning | Close to End |
| Effects | Search diversity | Stable solution |




## 3.9 Decisive factors for Genetic Algorithms optimization:

1. Encoding and decoding methods
2. Crossover probability.
3. Mutation probability.
4. Population control.

[14]

# 4.UTILIZATION AND FUNCTIONAL IMPLIMENTATION OF GENETIC ALGORITHMS

Identification of the global Optimization for the objective function is fundamental to the theoretical support of most econometric estimators. Numerous Optimization techniques have been developed and are represented in literature (non-linear least square, maximum likelihood,). It is always a problem for these methods to locate local optima, especially in situations of multiple local optima, which limits their usefulness. Genetic Algorithm (GAs) has the property of searching the entire space and locates the global optimum. Genetic Algorithm has been successfully used in biology, engineering, and other areas.

The most important property of GAs is that they don't require a well-behaved objective function, but this function should be bounded for the solution to exist. The Genetic Algorithms search for a solution using only values of the function to be maximised. These algorithms iterate toward a solution through a process that in many ways parallels the Darwinian process of natural selection. Simply, the algorithm starts with an initial population of candidate solutions (the first generation), and then selects a subset of the population to contribute off springs to the next generation of candidate solutions. The choice of population number is very important. According to the writers, a population number between 20 and 30 seemed to work best.

The GAs operators like reproduction; crossover and mutation were discussed with simple example to illustrate these three operators. Based on the fitness of every member in the population, only a few selected will contribute to the next

generation and this process will continue and eventually, the initial population evolves to one that contains a solution to the Optimization problem.

The applicability of GAs to econometric Optimization problems was also discussed by applying GAs to six test problems selected from the econometric literature. Global solutions were obtained for each problem, with the accuracy dependent on the number of generations allowed for search.

Concluding, GAs incorporates both a global search process and the ability to focus on the optimal value once found. The algorithm is broadly applicable, requiring not differentiability, global convexity, or even continuity. The GAs used in this paper offer a solution to the common difficulty of obtaining the global optimum when attempting to estimate complex econometric problems. [15]

## 4.1Optimization Stuff (Problems)

Just think of an OPTIMIZATION problem as a black box. A large black box. As large as, for example, a Coca-Cola vending machine. Now, we don't know nothing about the inner workings of this box, but see, that there are some regulators to play with, and of course we know,that we want to have a bottle of the real thing... Putting this everyday problem into a mathematical model, we proceed as follows:

(1) we label all the regulators with x and a number starting from 1; the result is a vector x, i.e. (x_1,...,x_n), where n is the number of visible regulators.

(2) we must find an objective function, in this case it's obvious, we want to get k bottles of the real thing, where k is equal to 1.[some might want a "greater or equal" here, but we restricted ourselves to the visible regulators (we all know that sometimes a "kick in the right place" gets use more than 1, but I have no idea how to put this mathematically...)]

(3) thus, in the language some mathematicians prefer to speak in:

$f(x) = k = 1$. So, what we have here is a maximization problem presented in a form we know from some boring calculus lessons, and we also know that there at least a dozen utterly uninteresting techniques to solve problems presented this way. [ 16 ]

## 4.2 Solution Of The Problem

We can either try to gain more knowledge or exploit what we already know about the interior of the black box. If the objective function turns out to be smooth and differentiable, analytical methods will produce the exact solution.

If this turns out to be impossible, we might resort to the brute force method of enumerating the entire SEARCH SPACE. But with the number of possibilities growing exponentially in n, the number of dimensions (inputs), this method becomes infeasible even for low-dimensional spaces.

Consequently, mathematicians have developed theories for certain kinds of problems leading to specialized OPTIMIZATION procedures.These algorithms perform well if the black box fulfils their respective prerequisites. For example, Dantzig's simplex algorithm(Dantzig 66) probably represents the best known multidimensional method capable of efficiently finding the global optimum of a linear,hence convex, objective function in a SEARCH SPACE limited by linear constraints. Gradient strategies are no longer tied to these linear worlds, but they smooth their world by exploiting the objective function's first partial derivatives one has to supply in advance. Therefore, these algorithms rely on a locally linear internal model of the black box.

Newton strategies additionally require the second partial derivatives, thus building a quadratic internal model.Quasi-Newton,conjugate gradient and variable metric strategies approximate this information during the search.The deterministic strategies mentioned so far cannot cope with deteriorations, so the search will stop if anticipated improvements no longer occur.In a multimodal ENVIRONMENT these algorithms move "uphill" from their respective starting points. Hence, they can only converge to the next local optimum.

Newton-Raphson-methods might even diverge if a discrepancy between their internal assumptions and reality occurs. But of course,these methods turn out to be superior if a given task matches their requirements. Not relying on derivatives, polyeder strategy,pattern search and rotating coordinate search should also be mentioned here because they represent robust non-linear OPTIMIZATION algorithms.

Dealing with technical OPTIMIZATION problems, one will rarely be able to write down the objective function in a closed form. We often need a SIMULATION model in order to grasp reality.In general,one cannot even expect these models to behave smoothly. Consequently,derivatives do not exist.That is why optimization algorithms that can successfully deal with black box-type situations habe been developed. The increasing applicability is of course paid for by a loss of "convergence velocity," compared to algorithms specially designed for the given problem.Furthermore,the guarantee to find the global optimum no longer exists! In the attempt to create tools for various purposes, mankind has copied, more often instinctively than geniously,solutions invented by nature. Nowadays,one can prove in some cases that certain forms or structures are not only well adapted to their ENVIRONMENT but have even reached the optimum.

This is due to the fact that the laws of nature have remained stable during the last 3.5 billion years. For instance, at branching points the measured ratio of the diameters in a system of blood-vessels comes close to the theoretical optimum provided by the laws of fluid dynamics (2^-1/3). This,of course,only represents a limited,engineering point of view on nature. In general, nature performs adaptation, not optimization.

The idea to imitate basic principles of natural processes for optimum seeking procedures emerged more than three decades ago.

Although these algorithms have proven to be robust and direct OPTIMIZATION tools, it is only in the last five years that they have caught the researchers' attention. This is due to the fact that many people still look at organic EVOLUTION as a giantsized game of dice,thus ignoring the fact that this model of evolution cannot have worked:a human germ-cell comprises approximately 50,000 GENEs,each of which consists of about 300 triplets of nucleic bases. Although the four existing bases only encode 20 different amino acids,20^15,000,000,ie circa 10^19,500,000 different GENOTYPEs had to be tested in only circa 10^17 seconds,the age of our planet.So,simply rolling the dice could not have produced the diversity of today's complex living systems.

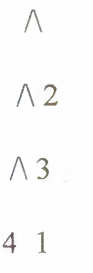
Accordingly, taking random samples from the high-dimensional parameter space of an objective function in order to hit the global optimum must fail (Monte-Carlo search). But by looking at organic.EVOLUTION as a cumulative, highly parallel sieving process, the results of which pass on slightly modified into the next sieve, the amazing diversity and efficiency on earth no longer appears miraculous.When building a model, the point is to isolate the main mechanisms which have led to today's world and which have been subjected to

evolution themselves. Inevitably,nature has come up with a mechanism allowing INDIVIDUALs of one SPECIES to exchange parts of their genetic information (RECOMBINATION or CROSSOVER),thus being able to meet changing environmental conditions in a better way.

## 4.3 Genetic Query Optimization (GEQO) in Postgres

The GEQO [17] module is intended for the solution of the query Optimization problem similar to a travelling salesman problem (TSP). Possible query plans are encoded as integer strings. Each

String represents the join order from one relation of the query to the next. e. g., the query tree



is encoded by the integer string '4-1-3-2', which means, first join relation '4' and '1', then '3', and then '2', where 1, 2, 3, 4 are reloads in Postures.

Parts of the GEQO module are adapted from D. Whitley's Genitor algorithm.

Specific characteristics of the GEQO implementation in Postgres are:

- Usage of a steady state GA (replacement of the least fit individuals in a population, not whole-generational replacement) allows fast convergence towards improved query plans. This is essential for query handling with reasonable time;
- Usage of edge recombination crossover which is especially suited to keep edge losses low for the solution of the TSP by means of a GA;

- Mutation as genetic operator is deprecated so that no repair mechanisms are needed to generate legal TSP tours.

The GEQO module gives the following benefits to the Postgres DBMS compared to the Postgres query epitomiser implementation:

- Handling of large join queries through non-exhaustive search;
- Improved cost size approximation of query plans since no longer plan merging is needed (the GEQO module evaluates the cost for a query plan as an individual).

## 4.4 Functional Implementations

### 4.4.1 GA function for plotting the object function in 3D.

```
function gaPlot3D(fun,low,up,step)
xr = low(1):step(1):up(1);
yr = low(2):step(2):up(2);
[x y] = meshgrid(xr, yr);
                    % Compute the function values for the population elements.
z = eval([fun, '(x,y)']);
surface(xr,yr,z); xlabel('x'); ylabel('y');
hold on;
contour(xr,yr,z,15,'--');
axis([low(1) up(1) low(2) up(2)]);
view(-35,25);
hold off;
```

[18]

### 4.4.2 Implementation of a real-coded Genetic Algorithm

```
Format long;
```

```
format compact;
rand('state',sum(100*clock));
clear
case_nr = 1;
if (case_nr == 1)
  % Griewank
  fun = 'griewank';                    % Name of object function
  low = [-600,-600]; up = [600, 600];  % Box constraints
  nvar = 2;                            % Number of variables
  genNr = 300;                         % Number of generations
  result = [100,100];                  % The optimum point
  llim = -0.01;                        % Lower limit for accepting optimum
elseif (case_nr == 2)

                                       % Peaks
  fun = 'peaks';                       % Name of object function
  low = [-3,-3]; up = [3, 3];          % Box constraints
  nvar = 2;                            % Number of variables
  genNr = 20;                          % Number of generations
  result = [-0.0106,1.5803];           % The optimum point
  llim = 18;                           % Lower limit for accepting optimum
elseif (case_nr == 3)

                                       % Rosenbrock
  fun = 'rosenbrock';                  % Name of object function
  low = [-5.12,-5.12];                 % Box constraints
```

```
up = [5.12, 5.12];
nvar = 2;                          % Number of variables
genNr = 200;                       % Number of generations
result = [1,1];                    % The optimum point
llim = -0.1;                       % Lower limit for accepting optimum
end


                                   % Parameters for genetic algorithm
npop = 30;                         % Size of the population
crossProb = 0.8;                   % Probability of crossover
mutProb = 0.02;                    % Probability of mutation
p_tour = 0.7;                      % Tournament probability
mut_scale = 0.1;                   % Scale for mutations
n = 100;                           % Number of runs
if (1)                             % Start testing the genetic algorithm
mytime = cputime;
low_value = 1000;
up_value = -1e8;
ga_ok = 0;
initpop = zeros(npop,nvar);
for i = 1:n
                                   % Initialize the population
for j = 1:npop
initpop(j,:) = low + (up-low).*rand(1,nvar);
end
```

```
[endpop endvalues stats] = gaSim(fun, low, up, ...
initpop, crossProb, mutProb, p_tour, mut_scale, genNr);
values(i) = max(endvalues);
if (values(i) < low_value)
        w_initpop = initpop;
        w_pop = endpop;
        w_val = endvalues;
        w_stat = stats;
        low_value = values(i);
end
if (values(i) > up_value)
        b_initpop = initpop;
        b_pop = endpop;
        b_val = endvalues;
        b_stat = stats;
        up_value = values(i);
end
tmp = find(values(i) == endvalues);
x = endpop(tmp(1),1);
y = endpop(tmp(1),2);
disp(['case: ', num2str(i), ', best value: ',num2str(values(i)) ...
', coord: ', num2str(x), ' ', num2str(y)]);
tmp = find(endvalues > llim);
x = endpop(tmp,:);
```

```
testres = result(ones(size(x(:,2))),:);
if (any(sqrt(sum((x-testres)'.^2)) < 0.2))
    ga_ok = ga_ok + 1;
end
end
wplot                                          % Plot worst case
bplot                                          % Plot best case
[max(values), min(values), mean(values), median(values), std(values)]
disp(['success rate: ', num2str(100*ga_ok/n), '%']);
disp(['cputime: ', num2str(cputime-mytime)]);
end

                                               % Random sampling:
if (1)                                         % Start testing the random search
mytime = cputime;
rvalues = -500 + zeros(n,1);
r_ok = 0;
for i = 1:n
for j = 1:(npop*genNr)
x = low + (up-low).*rand(1,2);
   z = eval([fun '(x(1),x(2))']);
   if (z > rvalues(i))
     rvalues(i) = z;
          bestx = x;
        end
```

```
 end

 x = bestx;

 disp(['case: ', num2str(i), ', best value: ',num2str(rvalues(i))...

    ', coord: ', num2str(x(1)), ' ', num2str(x(2))]);

 if ((sqrt(sum((x-result).^2)) < 0.2) & rvalues(i) > llim)

    r_ok = r_ok + 1;
 end

end

[max(rvalues), min(rvalues), mean(rvalues), median(rvalues), std(rvalues)]

disp(['success rate: ', num2str(100*r_ok/n), '%']);

disp(['cputime: ', num2str(cputime-mytime)]);

end
```

### 4.4.3 Commands for plotting the best final population of a number of GA runs.

```
figure;
ind = 1:length(b_stat(:,1));

p = plot(ind,b_stat(:,1),'-',ind,b_stat(:,2),'-.');

xlabel('Iteration')

title('Best and average value')

set(gcf, 'Position', [350 500 640 250])

set(gcf,'PaperPosition', [0.2 2.5 8 4])

set(get(gcf,'Children'),'Position', [0.05 0.15 0.9 0.720])

print -deps -loose bplot

figure;
```

```
set(gcf, 'Position', [350 200 640 300])
set(gcf,'PaperPosition', [0.2 2.5 8 4])
set(get(gcf,'Children'),'Position', [0.05 0.11 0.9 0.680])
subplot(1,2,1);
endx = b_initpop(:,1);
endy = b_initpop(:,2);
endvalues = eval([fun '(endx,endy)']);
gaPlot(fun, low, up, endx, endy, endvalues);
title('Initial population');
subplot(1,2,2);
endx = b_pop(:,1);
endy = b_pop(:,2);
endvalues = eval([fun '(endx,endy)']);
gaPlot(fun, [min(endx),min(endy)], [max(endx),max(endy)], ...
  endx, endy, endvalues);
title('Final population');
print -deps -loose bplotcont
```

[ 18 ]

## 4.5 GENETIC Algorithms for Optimisation Technology transfer

### Problem solving strategies

Usual methods for solving problems, replace an infinite procedure with a finite procedure.

- Replace a non-linear problem with a linear one: Taylor series, Newton's method (use

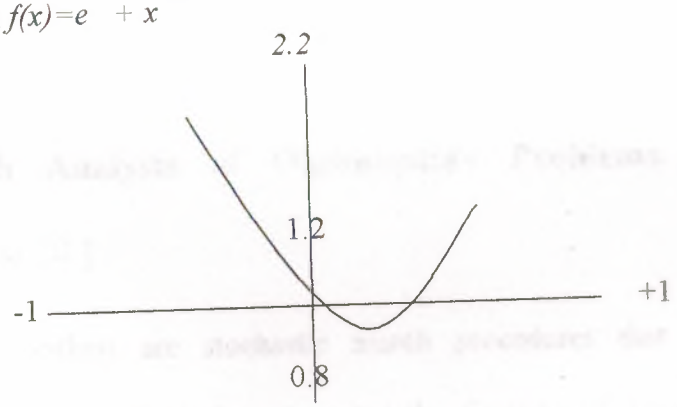
- Of derivatives), etc.
- Use simpler forms of matrices.
- Etc.
- Heuristic methods and frameworks may speed up this process

Thus, we need

- Alternative problems, which can be solved
- Transformations into the solvable problems

4.2 A simple optimisation problem
Problem:

$$\min f(x) = e^{-x} + x^2$$




Solution with Mathematica:

```
In[1]:= D[Exp[-x] + x^2, x]

Out[1]= -E^-x + 2 x

In[2]:= FindRoot[%1, {x,0}]

Out[2]= {x -> 0.351734}

In[3]:= FindMinimum[Exp[-x] + x^2, {x, 0}]

Out[3]= {0. 827184, {x -> 0.351734}}
```

[ 19 ]

### 4.5.1 Types of Optimization problems

- linear programming (LP):

  min x c $^{T}x$, Ax =b or Ax $\leq b$, x $\geq 0$

- integer programming (IP):

  min $x$,y c $^{T}$x + d $^{T}y$; so that Ax + Dy $\leq b$;

  $yi$, $i$ ... $1$; : : :; r, are integer variables.

- Quadratic programming (QP):

$$\min x \; \frac{1}{2} \; {}^{T}Qx + c \; {}^{T}x, \; Ax \leq b$$

- (Unconstrained) non linear optimisation:

  min f(x)

## 4.6 Walsh Analysis of Optimization Problems for Genetic Algorithms[ 20 ]

Genetic Algorithms are stochastic search procedures that have borrowed concepts from natural evolution to control the direction of search in the solution space. GAs have proven useful in providing solutions in an industrial setting for a variety of very difficult problems such as jet engine design, factory floor scheduling, VLSI layout, and production parameter selection.

Bit strings that play the role of chromosomes represent the search space for GAs. Selective pressure is applied to the "breeding" of new chromosomes based on the "fitness" of the chromosome. It is this fitness, which the algorithm tries to optimize. In order to understand the difficulty that various fitness functions pose for GAs we have turned to Walsh analysis. Walsh analysis is an analogue for

discrete Fourier analysis but is designed for functions with a bit space domain. This makes them ideal for studying the properties of the chromosomal search space.

In this talk I will give some background on genetic algorithms. I will introduce Walsh analysis focusing on some of the interesting properties of Walsh functions and their implications. I will then show how we have been using Walsh analysis to analyse problem difficulty for GAs and characterise completeness and difficulty of several classes of problems. Finally, I will show that Walsh analysis is insufficient for estimating problem difficulty.

# 5. MULTIMODAL OPTIMIZATION USING GENETIC ALGORITHMS

## 5.1 Problem statement

Genetic algorithms (GA's) have been applied successfully to optimize several kinds of problems, where many traditional methods (such as the gradient search, the simplex meth) are not applicable because of assumptions or requirements - such as the need for objective function derivatives, convexity, or the linear correlation of variables. One general problem of Optimization techniques is that there is not always a guarantee of convergence to he globally optimal solution. Frequently they converge to local optima. GA's may also have similar problems. One of the major problems is known as *premature convergence*, which means that all individuals in a population become nearly identical before the optima has been located. In the worst case, the GA also becomes trapped in a suboptimal solution, even though GA's are actually considered to be very robust. In many Optimization problems (i.e. design problems) a technique capable of locating multiple solutions is often desired because humans will make the final judgment, decision or selection between alternatives. A better understanding of the search space structure, again revealed by finding multiple solutions can be quite helpful. If the search space has multiple solutions, a traditional GA is only able to locate one of them, even though the multiple solutions might be of equal quality. Additionally, niching methods might overcome problems such as premature convergence and being trapped in local optima. This paper explores the application of niching methods to multimode domains, where a desire for locating multiple solutions exists. [21]

## 5.2 Multi-modal Optimization (Problem solving by Niching Method)

The extension of genetic algorithms to multimodal Optimization (niching methods) is investigated. After reviewing current niching methods, a new variation of a crowding technique, named the struggle genetic algorithm, is introduced. Replacement occurs only between similar individuals based upon a similarity measure if an offspring wins the competition. Using a suite of test problems, the performance of the struggle genetic algorithm and three other niching methods – deterministic crowding restricted tournament selection, and fitness sharing - is empirically examined. For each test problem the struggle GA consistently located a more complete set of optimal solutions. The struggle GA has also performed well when compared to global methods (simple and steady state GA). Additionally, crossover's adaptive mechanism based upon the similarity measurement of parents is investigated for commonly used representations. Empirical investigations suggest that real-coded parameters are superior to the traditional binary or Gary coding of continuous variables. For real-coded variables a newly designed crossover operator, the sphere-crossover, is introduced and tested. Preliminary test results illustrate its adaptive Power [21]

### 5.2.1 Using GAMS

Using GAMS (General Algebraic Modelling System) to minimize the function

$$F(x1,x2)=100(x2-x1)^2+(1-x1)^2$$

```
VARIABLE  F object function

POSITIVE VARIABLES X1, X2;

EQUATION FUNC define the object function;
```

```
FUNC.. F =E= 100*SQR(X2 - SQR(X1))+ SQR(1 - X1);      * Feasible region

X1.LO = -10; X1.UP = 5;

X2.LO = -10; X2.UP = 5;                                * Initial guess

X1.L = -1.2; X2.L = 1.0;

MODEL ROSE / ALL /;

SOLVE ROSE MINIMIZING F USING NLP;
```

## 5.3 Algorithms For Multi –Modal Optimisation

Select an initial guess x 1 and set $k$ =1.

**Repeat**

Solve the search direction p $k$ from equation (1)–(3) below.

Find the next point using Eq. (4):

$$xk + 1 = x\,k + \lambda k\ p\,k\ .$$

Set $k=k+1$.

**until** $||x\,k + xk\text{-}1|| < E$

Conjugate gradient methods:

$$p\,k = -\nabla f(xk) + \beta k\ p\,k - 1$$

Secant methods:

$$Bk\ p\,k = -\nabla f\,(xk)$$

Newton's method:

$$H(x\,k)\ p\,k = -\nabla f(x\,k)$$

Line search:

$$\lambda k = \arg\min f(x\,k + \lambda p\,k)$$

## 5.4 About computational science:

- Cross-disciplinary co-operation is increasingly important

- Discipline-specific knowledge
- Mathematical modelling skills
- Knowledge of numerical solution methods
- Programming skills (including code optimisation and parallelization)
- Skills for visualization and analysis of data
- Phenomenon! Model! Simplification! Solution algorithm! Implementation! Simulation! Optimization! Analysis of results
- Bottleneck: human effort vs. computer power [21]

### 5.4.1 Approaches to computational science:

"Experimental approach"

"Model building"

"Algorithmic thinking"

"Theorem-based thinking"

Targets for GA research"

- Using GAs as general problem-solving tools
- Finding the "perfect GA" for a given class of problems
- Understanding the behaviour of GAs
- Using GAs for teaching/learning
- Using GAs for increasing co-operation between disciplines [21]

## 5.5 Methods for global optimisation:

- There are no general-purpose algorithms
- Methods based on random sampling
- Local optimisation with different starting points
- Clustering methods
- Transformation methods

- Simulated annealing
- Genetic algorithms
- Tabu search
- Etc.

## 5.6 Developing scientific software

Finnish industry is starting to invest in mathematical modelling and software development. Industry is interested in co-operation with universities (technology transfer works well) Education is a bottleneck: numerical methods, mathematical modelling, software development etc. are not taught widely enough

Modelling and algorithm development are very important skills, but difficult to teach Many engineering laboratories have developed their own software and have extensive industrial contacts. [ 21 ]

## 5.7 Review of niching methods

Evolutionary strategies like genetic algorithms are a general optimisation/search technique, which imitate the principles of natural evolution and molecular genetics. Unlike other methods, genetic algorithms operate upon several solutions (a population).

GA's to allocate and maintain multiple different optimal / suboptimal solutions in a population are called niching methods or multimodal GA's. The inspiration for niching again stems from nature, where different species coexist through adaptation to different niches. [22]

### 5.7.1 Niching methods

*Crowding* [and *fitness sharing*] are the two primary approaches for preserving diversity and maintaining multiple solutions in a population. A distance metric defined over the search space (which can be either *genotypic* or *phenotypic*) is used to distinguish the similarity of individuals in all niching methods. Crowding techniques use this measurement to replace favourably similar individuals, whereas fitness sharing uses the metric to debate an individuals' fitness by an amount according to the number of similar individuals in the population. [22]

### 5.7.2 De Jong's crowding

De Jong introduced an algorithm, which he called the *crowding factor model*. Specified by the generation gap G, $G \cdot Pop\ Size$ individuals are chosen via fitness proportionate selection to create an equal number of offspring. For each of the offspring a random sample of *CF* (crowding factor) individuals is scanned from the current population. The offspring then replaces the most similar individual of this sample. De Jong used the Hamming-distance between two individuals as the measurement for similarity. Typical values for the crowding factor *CF* are 2 or 3 and 10% for the generation gap G. However, for the goal of maintaining multiple solutions in a population, it has been shown that De Jong's crowding is only of limited use Mash Niching. [22]

### 5.7.3 Deterministic crowding

S. Mahfoud proposed this variation of De Jong's crowding. Deterministic crowding first randomly groups all individuals in a population into parent pairs. Each pair generates two offspring by application of the genetic operators. Every offspring then competes against one of its parents. There are two possible parent-

child tournaments, decided by the sum of the distances between the parents and the children of both possible combinations. The winner of the competitions moves on to the next generation. The pseud code for the two possible parent-child tournaments is given below:

IF [d($P1$, $C'1$) + d($P2$, $C'2$) = d($P1$, $C'2$) + d($P2$, $C'1$)]

IF f($C'1$) _ f($P1$) replace *P1* with *C'1*

IF f($C'2$) _ f($P2$) replace *P2* with *C'2*

ELSE

IF f($C'1$) _ f($P2$) replace *P2* with *C'1*

IF f($C'2$) _ f($P1$) replace *P1* with *C'2*

### 5.7.4 Fitness Sharing

Goldberg and Richardson used Holland's sharing concept for niching. Every individual in a niche shares its fitness with all others in that niche. Niches with higher fitness values are able to support more individuals. In contrast with crowding techniques (where the replacement strategy influences diversity by replacing similar individuals), sharing uses fitness durations and the parental selection mechanism to affect population diversity. In addition to requiring a distance or similarity measure, a niche radius (threshold distance) is needed to define the niches for individuals. The altered fitness value (shared fitness) of an individual is computed from its individual fitness divided by its niche count. The niche count of an individual is the sum of the sharing function values between itself and all individuals in the population (including itself). If two elements of a population are identical, the sharing function returns a value of 1. If two elements in a population exceed a certain threshold distance (also called niche radius)

sharing function returns a value of 0, implying that the individuals are in different niches and do not share their fitness values.

The parameter alpha controls the shape of the sharing function (the sharing function is linear for i =1). The derated fitness *f* of an individual *i* is given by:

$$f^*(I) = \frac{f(I)}{\sum_{j=1} sh[d(I)j)]}$$

Along with the use of simple GAs (generational replacement) for sharing, overlapping populations may be used. In this scheme, new offspring are first added to the current population. Shared fitness values are computed and a number of individuals (equal to the number of children) are then eliminated from the population. This is accomplished by removing those individuals with the worst shared fitness values. For fitness proportionate selection to work properly on the next generation, the shared fitness values are then recalculated. This technique carries a higher computational overhead (distance comparisons) than fitness sharing with non-overlapping populations. [22]

### 5.7.5 Sequential Niching

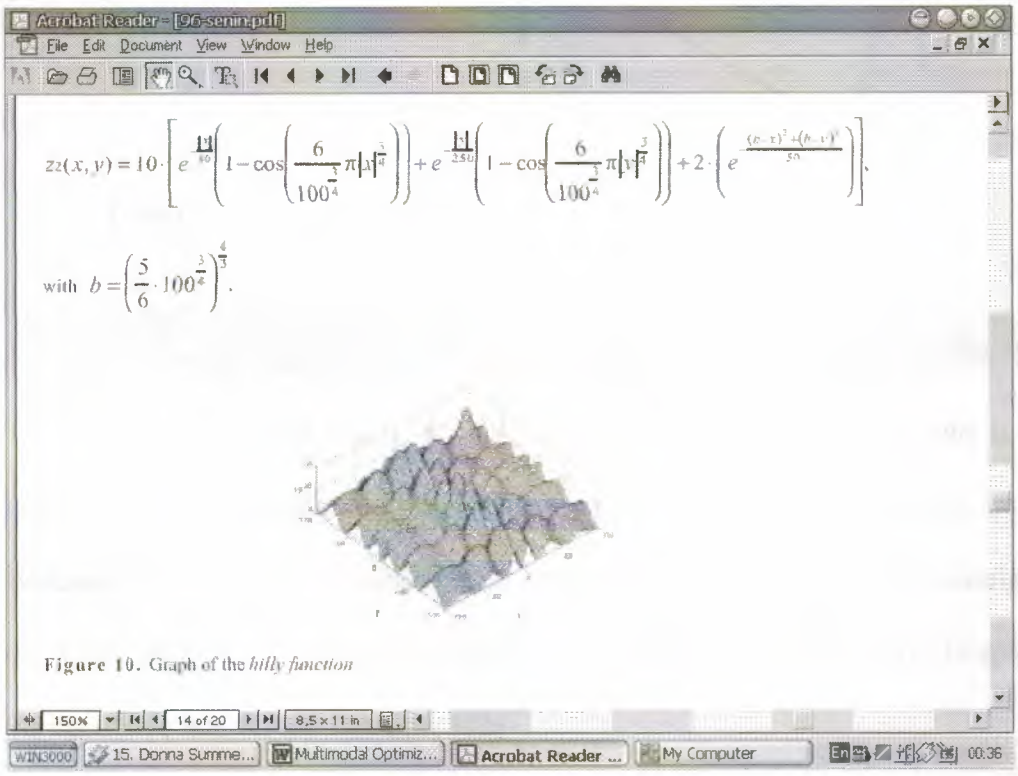
This variation on fitness sharing was proposed by Beasley, Bull and Martin Multiple solutions are found serially with sequential runs of a simple GA. The best solution of each

run is stored. In order to prevent convergence to a previously located optima, the fitness values of solutions near previously found solutions (within a niche radius) are derated. [ 22 ]

## 5.8 Hilly Function

Frequently, a wide variety of one-dimensional sinusoidal test functions with different properties are used to test Niching methods. Here, we designed a two-dimensional function with unequally spaced peaks of non-uniform height. In the x-direction the height decreases much more than in the y-direction (from the origin). Additionally, global optima are added far from the generally promising area near the origin. For

$x,y \in [-100;100]$, this function (as defined in equation 13 and shown in figure 10) has 36 peaks and its global optima is located




**5-1.** Graph of the *hilly function* [ **22** ]

## 5.9 Order-five deceptive problem

Another class of test problem was an order-five deceptive problem [GuD] with ten concatenated fully deceptive five-bit sub functions, altogether forming a 50-bit problem. Each sub function has two complementary attractors (a deceptive one at

00000} and a global one at 11111. Sub function and function values related to the number of 1's (unitation) are shown in figure 5-2

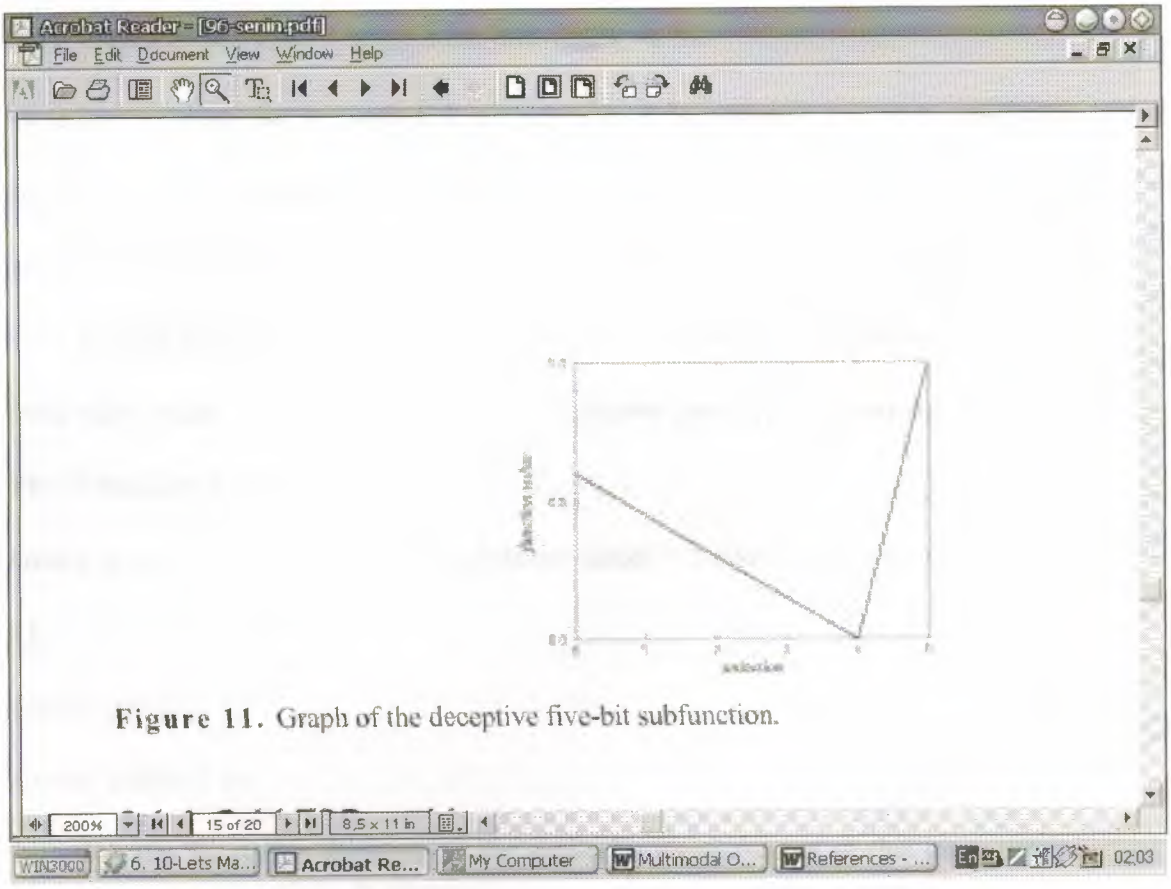



**Figure 11.** Graph of the deceptive five-bit subfunction.

This 50-bit problem has a total of 2 10 = 1024 optima, of which only one is the global. Goldberg investigated this problem with a fast messy GA , where in the evaluation only the global optima was considered. Likewise we will investigate the ability of Niching methods to locate the global maxima..**Figure 11.** Graph of the deceptive five-bit sub function. **[22]**

## 5.10 Grid-TSP Problem

The traveling salesman problem (TSP) is representative for a class of combinatorial optimization problems. Given *n* cities the task for the salesman is to visit all cities only once so that the overall length of the tour is minimal. In order to build a problem with multiple global optima, twenty cities were arranged on an

evenly spaced rectangular (5x4) grid. Since we chose the grid-distance to be 1.0 so the 14 global optima (shown in figure 12) have all tour lengths of 20.0. [22]

## 5.11 Guidelines For Using The Struggle GA.

Experiences using the struggle GA suggest that it is desirable to use a similarity measure with a meaning that matches the similarity in the phenotypic search space. Consider the binary encoding of real values. Two nearby real values could have a quite big genotypic Hamming-distance, whereas two real values far away from each other could have an almost negligible genotypic Hamming-distance. This is illustrated below:

*binary strings s1 and s2 Hamming distance dmax* = 5 *Euclidean distance dmax* = 31

10000, 01111 5 1

10000, 00000 1 16

By using the Hamming-distance as the similarity measurement in this case, many replacement errors will occur and therefore a Niching algorithm will perform poorly. Additionally, the crossover operator should adapt its scatter according to the phenotypic similarity of the mated parents. Even though it is not clear which distribution form is the best for a particular problem, a discontinuous and inconsistent form (for parents of instinct distances) appears to be disadvantageous. Interestingly enough, this observation about crossover seems to strongly affect traditional genetic algorithms (simple and steady state). Domain knowledge should be incorporated into a meaningful distance metric and the design of a crossover operator, which has the identified properties of adapting its scatter to the parents similarity. [ 22 ]

## Summary

In this paper we have investigated and tested Niching methods for both global optimization and the location of multiple solutions in several domains. A new crowding variation, the struggle GA, was proposed for multimodal optimization. Newly generated offspring must be better than the most similar individual in a population in order to survive. This mechanism minimizes replacement errors, thereby reducing the algorithm's chance of being misled. Additionally, no problem-specific parameters for the algorithm are required. Empirical test results show that, on average, a more complete set of optimal solutions is located and maintained compared to other investigated Niching methods. Further improvements might be achieved if parents were also selected based upon similarity. This is a subject for future work. A close relationship between crossover's scatter and the similarity of parents was shown. Every crossover operator examined scattered with an operator-specific distribution around the similarity average of the parents and adapted its sampling area according to the distance (similarity) of the parents. This adaptive mechanism of crossover seems to support its unique capability of exploring and exploiting a search space. A binomial-like and normal distributions of the offspring from the parental similarity average seems to be advantageous. The traditional crossover operators for binary and Gray-coded continuous variables exhibit characteristics quite different to the other investigated cases. The scatter distributions of these operators are inconsistent and discontinuous. These anomalies may contribute to the observed superiority of real-coded chromosomes over the binary and Gray-coding of real-value parameters. An understanding of crossover's properties can be used to design new operators. Building on observations of crossover operators

(in relation to the similarity measure) a new crossover operator - the sphere-crossover- was designed for real-coded variables. It adapts the scatter of children according to the Euclidean distance of the parents. Test results demonstrate its capabilities relative to BLX.

# CONCLUSION

The graduation project is devoted to one of actual problem, to use of Genetic Algorithms for solving these optimization problems to solve these problems.

To solve the problem the state of understanding optimization problems are considered.

Specific character of GA, main genetic operations such as Selection, Mutation, and reproduction are described. Effectiveness of the application of GA for solving global optimization problem is shown. Main features of Genetic Algorithm based optimization and its representation are given as an example the optimization of Quadratic functions is discussed. Operational principles of GA methods such as Selection, reproduction, crossover, and mutation are widely discussed.

After the fragments of utilization and functional implementation of GA for optimization problems in MATLAB Programming. Language are given, also the application of GA for multi-modal optimization their description methods for global optimization and the application of Niching method are widely described.

# REFERENCES


[1] Wolfgang Banzhaf , Colin Reeves , col Reeves (1999) Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, New York.

[2] David E. Goldberg Addison,(January 1989), Optimization and Machine Learning Wesley Pub Co; ISBN: 0201157675

[3] Randy L. Haupt, Sue Ellen Haupt (January 1998) John Wiley & Sons; ISBN: 0471188735

[4] Lawrence Davis, , (1991),Handbook of Genetic Algorithms, Van Nostrand Reinhold New York.

[5] Mitsuo Gen, Runwei Cheng (2000) Genetic Algorithms and Engineering Optimization John Wiley & Sons, New York: ISBN:0471315311

[6] Nirwan Ansari, Edwin Hou (April 1997) Computational Intelligence for Optimization Kluwer Academic Publishers; ISBN: 0792398386

[7] David E. Goldberg and J. Richardso*n*, (1987), Genetic Algorithms with sharing formulation modal function optimization In J. J. Hillsdale, NJ.

[8] David E. Goldberg, K. Deb, H. Kargupta, and G. Harik, (1993),Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms, In J. D. Morgan Kaufmann Publishers.

[4] H. J. Antonisse and K. S. Keller, "1987",,Genetic operators for high-level knowledge representations,"Lawrence Erlbaum Associates, Publishers.

[21] CSC Report on Scientific Computing 1997–1998, Cray T3E User's Guide, CSC User's Guide:

[19] Textbook: Fortran 90/95 Solving optimization problems

[05] Thomas Grüninger , and David Wallace, Department of Mechanical Engineering, Stuttgart University Massachusetts Avenue, Massachusetts Institute of Technology.

[22] Juha Haataja, May 30 – June 3, 1999 , Using Genetic Algorithms for Optimization ,Center for Scientific Computing Finland

[7] David H. Ackley, 1985, Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates.NJ.

[14] Thomas and Hans-Paul Schwefel ,1993",An Overview of Evolutionary Algorithms for Parameter Optimization," NY.

**WEB REFRENCES:**

[17] http://www.ieee.org

[3] http://www.computer.org/abstracts

[10] http://www.csc.fi/math_topics/opt/

[11] http://www.csc.fi/math_topics/opt/ohj/

[12] http://www.csc.fi/oppaat/gams/

[13] http://www.csc.fi/oppaat/symb.html

[14] http://www.csc.fi/reports/cr97-98/

[15] http://www.mcs.anl.gov/home/otc/Guide/

http://www.netlib.org

http://lancet.mit.edu/~mbwall/presentations/IntroToGAs/P001.html

[9] http://cadlab.mit.edu/

http://www.computer.org/proceedings

[8] http://cne.gmu.edu/modules/GA/

http://www.aridolan.com/ga/gaa/SphereModel.html

# INDEX