Optimization of Oxygen in Ventilators Using Genetic Algorithm

Project Report Submitted

by

GAYATHRI GANGADHARAN

V. JAISHREE

M. V. KAVITHA

P. V. PRABHA

Under the guidance of Prof. K. RAMPRAKASH, M.E., MISTE.

Submitted in partial fulfilment of the requirements for the Degree of

BACHELOR OF ENGINEERING

in Electronics and Communication Engineering of

Bharathiar University, Coimbatore.

1996 - 97

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Kumaraguru College of Technology

COIMBATORE 641 006

Department of Electronics & Communication Engineering

Kumaraguru College of Technology

Coimbatore - 641 006.

CERTIFICATE

This is to certify that the Report entitled

OPTIMIZATION OF OXYGEN IN VENTILATORS USING GENETIC ALGORITHM

has been submitted by

Ms. KAVITHA, GAYATHRI, PRABHA & JAISHREE.

In partial fulfillment of the requirements for the award of Bachelor of Engineering in Electronics and Communication Engineering Branch of Bharathiar University, Coimbatore - 641046 during the academic year 1996 - '97.

GUIDE

9/4

HEAD OF THE DEPARTMENT

Certified that the candidate was examined by us in the project work Viva - Voce examination held on _____ and the University Register Number was _____

10/4

14/97

INTERNAL EXAMINER

EXTERNAL EXAMINER

CONTENTS

ACKNOWLEDGEMENT

We are grateful to Dr.S.SUBRAMANIAN M.Sc (Engg.), Ph.D. , SMIEEE , Principal, Kumaraguru College of Technology , for the ample facilities made available to accomplish this project.

We are immensely thankful to our beloved Prof . M.RAMASAMY M.E., MISTE , MIEEE , Head of Electronics and Communication Engineering Department , Kumaraguru College of Technology for his help to complete this project .

We express our deep sense of gratitude to Prof .K .RAMPRAKASH , Asst.Professor,Electronics and Communication Engineering Department , Kumaraguru College of Technology ,for his able guidance during the project work .His constant encouragement and support placed us in good stead , right through this work .

We are sincerely acknowledge our external guide Mr. . K .SUKESH KUMAR, Research Scholar, PSG Tech , for his valuable suggestion to select this project and for having put in his effort , time and cooperation in completion of this project work..

We are extremely thankful to Dr .M . SARAVANAN M .D .S ., KMCH , Mr. . M . SATHISH KUMAR , Bio - Medical Engineer, KMCH, Dr. SELVARAJ , Chief Anaesthesiologist , KMCH, Dr . PREMA , ICU Incharge , KMCH , Dr .MANONMANI , KMCH , Dr . P. CHANDRA MOHAN, Dr. NEELAVATHI ,Cardiothoracic Department , GKNM Hospital , Dr.RAJ PAUL , GKNM Hospital, Dr. VASISTA , Chief Anaesthesiologist , RAMAKRISHNA Hospital , for their invaluable help in the completion of this project.

We also express our indebtedness to the teaching and non - teaching staff of Electronics and Communication Engineering Department who helped us during the course of the project.

We wish to express our sincere and heartfelt thanks to our friends and well - wishers for their valuable suggestions and enthusiastic help towards the successful completion of this project .

SYNOPSIS

The use of Bio - Medical instruments has been increased in recent times in he medical field for clinical and research purposes . The advent of modern computing facilities has further enhanced the utility of the same with higher degree of reliability nd accuracy . An extensive use of computers in the Bio -Medical instruments designed to perform routine clinical measurements .

This project work involves optimization of oxygen in ventilators using genetic algorithm . Analysing the problems undergone by the doctors during the fixation of oxygen to the patient , it was found that they fix the oxygen using trial and error method . This leads to lot of complications to the patient . In order to optimize the oxygen level in the very first delivery we are using one of the optimization technique called Genetic algorithm , which focus on robustness .

In respiratory system the following 5 parameters are playing a major role to fix the oxygen level ,

(1) Total Lung Capacity (TLC)

(2) Tidal Volume (TV)

(3) Vital Capacity (VC)

(4) Forced Expiratory Volume (FEV)

(5) Respiratory Rate (RR)

Using standard biological equations involving age , height , weight and sex the above 5 parameters are calculated and from these parameters the required amount of oxygen is fixed by performing iterations over the fitness function , till the convergence is achieved .

This project is working satisfactorily for the tests conducted under different conditions .The results obtained are found to be correct and informative . This project has been implemented on Turbo C++ .

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The function of lungs is to promote the exchange of blood gases , so that oxygen is taken up from the ambient air and carbon di oxide is given off , thus keeping the threshold level of gases in the blood . This is carried out by the process of ventilation by which the lung is aerated . From this we can infer that the lungs are playing a major role in all functions of the human beings .

For patient needing ventilation , an external support to the respiratory system is provided with the help of a VENTILATOR . It gives required oxygen to the patient after the process of humidification . Fixing the amount of oxygen is the major problem faced by anaestheseoligists . Let us try to solve it .

Analyzing the recent developments in medical science have resulted in a situation where most physicians find it increasingly

difficult in making optimal clinical judgements even after specialization . Physicians usually fix the amount of oxygen in ventilators by trial and error method . It leads to various complications in the patients who are already in a critical condition .

This project is aimed at over coming such a difficulty . The amount of oxygen given to the patient is optimized at the very first delivery . This is achieved using a randomized technique , yes it is GENETIC ALGORITHM . It is an optimization technique , which focuses solely on convergence . It's goal is improvement and is concerned with doing better relative to others .

Chapter 2 deals with the functions of the respiratory system , Chapter 3 deals with the programming language C++ , Chapter 4 deals with the Genetic algorithm and Chapter 5 deals with the Design procedure of the project .

CHAPTER -2

THE FUNCTIONS OF RESPIRATORY SYSTEM

2.1 INTRODUCTION :

The exchange of gases in any biological process is termed respiration . To sustain life , the human must take in oxygen , which combines with carbon , hydrogen and various nutrients to produce heat and energy for the performance of work . As a result of this process of metabolism , which takes place in the cells ,a certain amount of water is produced along with the principal waste product carbon dioxide .The entire process of taking in oxygen from the environment , transporting the oxygen to the cells , removing the carbon dioxide from the cells and exhausting this waste product into the atmosphere must be considered within the definition of respiration. .

In the human body , the tissue cells are generally not in direct contact with their external environment . Instead , the cells are batched in fluid . This tissue fluid can be considered as the internal environment of

the body . The cells absorb oxygen from this fluid .The circulating blood is the medium by which oxygen is brought to the internal environment .

Carbon dioxide is carried from the tissue fluids by the same mechanism . The exchange of gases between the blood and the external environment takes place in the lungs is external respiration .

The function of the lungs is to oxygenate the blood in a controlled manner . During inspiration fresh air enters the respiratory tract , becomes humidified and heated to body temperature , and is mixed with the gases already present in the region comprising the trachea and bronchi (see figure 2.1) . This gas is then mixed further with the gas residing in the alveoli as it enters these small sacs in the walls of the lungs . Oxygen diffuses from the alveoli to the pulmonary capillary blood supply , whereas carbon dioxide diffuses from the blood to the alveoli . The oxygen is carried from the lungs and distributed among the various cells of the body by the blood circulation system , which also returns the carbon dioxide to the lungs . The entire process of inspiring and expiring air , exchange of gases , distribution of oxygen to the cells , and collection of carbon dioxide from the cells forms what is known as the pulmonary function .

2.2 MECHANICS OF RESPIRATION

Air enters the lungs through the air passages, which include the nasal cavities , pharynx , larynx , trachea , bronchi and bronchioles as shown in figure 2.1 .

The trachea is about 1.5 to 2.5 cm in diameter and approximately 11 cm long , extending from larynx to the upper boundary of the chest . Here it bifurcates (forks) into the right and left main stem bronchi . Each bronchus enters into the corresponding lung and divides like the limbs of a tree into smaller branches . The branches are unequal length and at different angles , with over 20 of these non-symmetrical bi-furcations normally present in the human body Further along with these branching , where the diameter is reduced to about 0.1 cm , the air - conducting tubes are called bronchioles . As they continue to decrease in size to about 0 .05 cm in diameter ,they form the terminal bronchioles , which branch again into the respiratory branchioles , here some alveoli are attached as small air sacs increase in number , becoming the pulmonary alveoli . The alveoli are each about 0.02 cm in diameter . It is estimated that , all told , some 300 million alveoli are found in the lungs .

Beyond about the tenth stage of branching , the bronchioles are embedded within alveolar lung tissue ;and with the expansion and

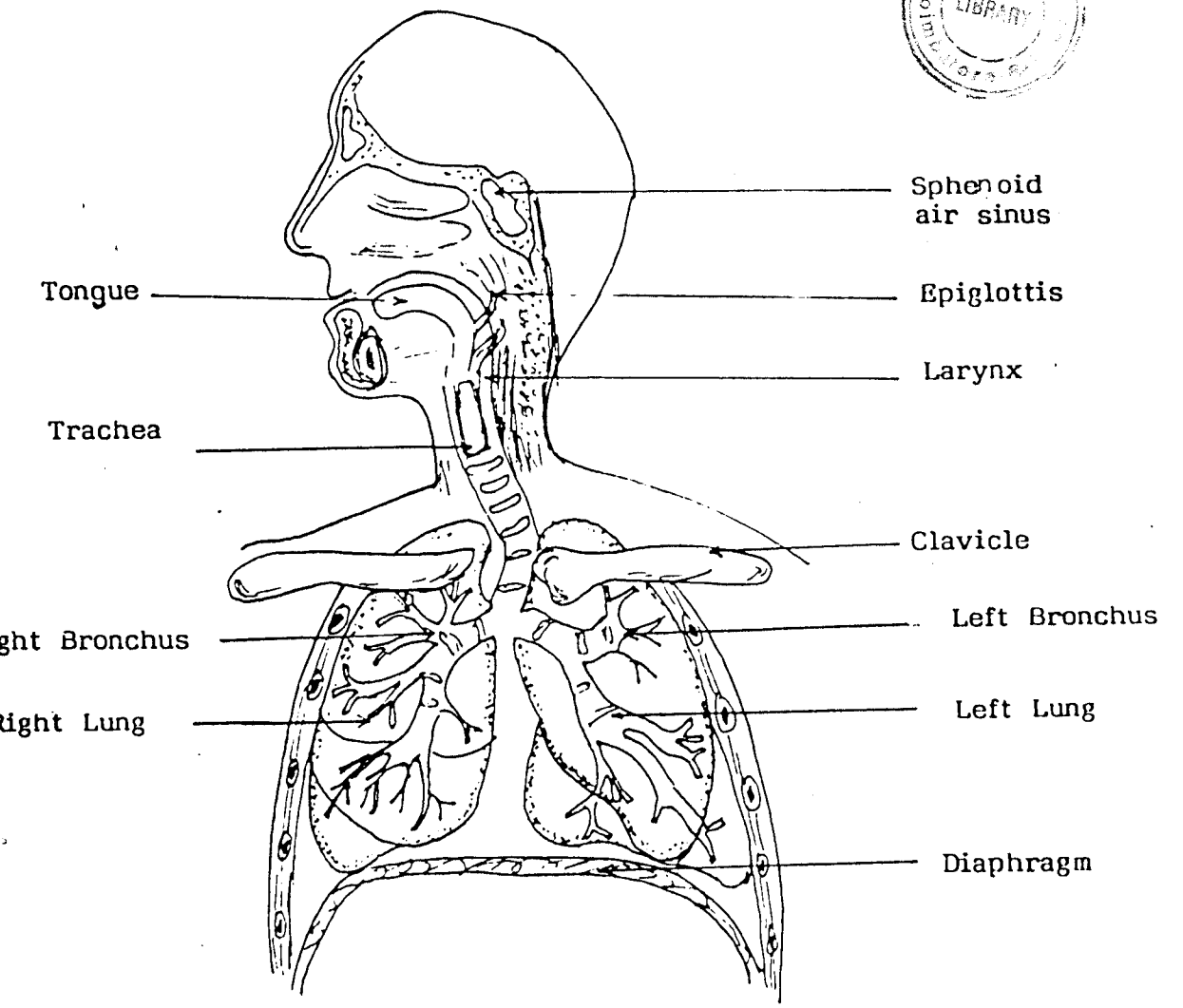


FIGURE 2.1

THE RESPIRATORY SYSTEM

relaxation of the lung , their diameters are greatly affected by the lung size or lung volume . Up to this point , the diameter of the air sacs are greatly affected by the pleural pressure , the pressure inside the thorax .

2.3. THE MECHANICS OF VENTILATION :

The function of the respiratory system is to secure gas exchange between the blood and ambient air so that the arterial blood gas pressures are kept within certain limits . Any departures from these limits implies that there is a breakdown in the system at some level .It is a function of the clinician to determine at what level this has occured and what is the cause of the condition .

In order to understand how the respiratory system secures this gas exchange , the various processes involved in respiration have to be considered .

The first VENTILATION , the process by which the lung is aerated .Ventilation is a cyclic activity with an inward flow of air called inspiration and an outward flow called expiration . The forces that cause the flow of gas are called the MECHANICS OF VENTILATION

Ventilation is the most easily measured of the processes of respiration . It is also the most important in that it is most frequently involved in the breakdown of the respiratory process .

2.4 VENTILATORS AND ITS TYPES :

Ventilators are equipments that gives an external support to the respiratory system . We have the types of ventilators based on the controlling operation . The first one is the a PRESSURE controlled ventilator while the other one is called as the VOLUME controlled ventilator and offcourse they are classified based on the manufacturer . Ventilators may be classified according to their functional components (table 1) and the control of minute ventilation , tidal volume and respiratory timing (table 2)

TABLE 1 CLASSIFICATION OF VENTILATORS : FUNCTIONAL COMPONENTS

1. POWER SOURCE ;

   Pneumatic (compressed gas)

   Electronic

   Combination

2 . POWER MECHANISM ;

Piston - driver ventilator

Rotary drive piston.

Linear drive piston (gear or spring)

Pneumatically driven ventilator.

High-pressure drive with high internal resistance

Low-pressure -driven bellows

Compressor- drive bellows.

3 . TRANSMISSON OF POWER ;

Direct or single circuit

Indirect or double circuit .

4. CONTROL MECHANISMS ;

Pneumatic

Fluidic

Electronic

Combination

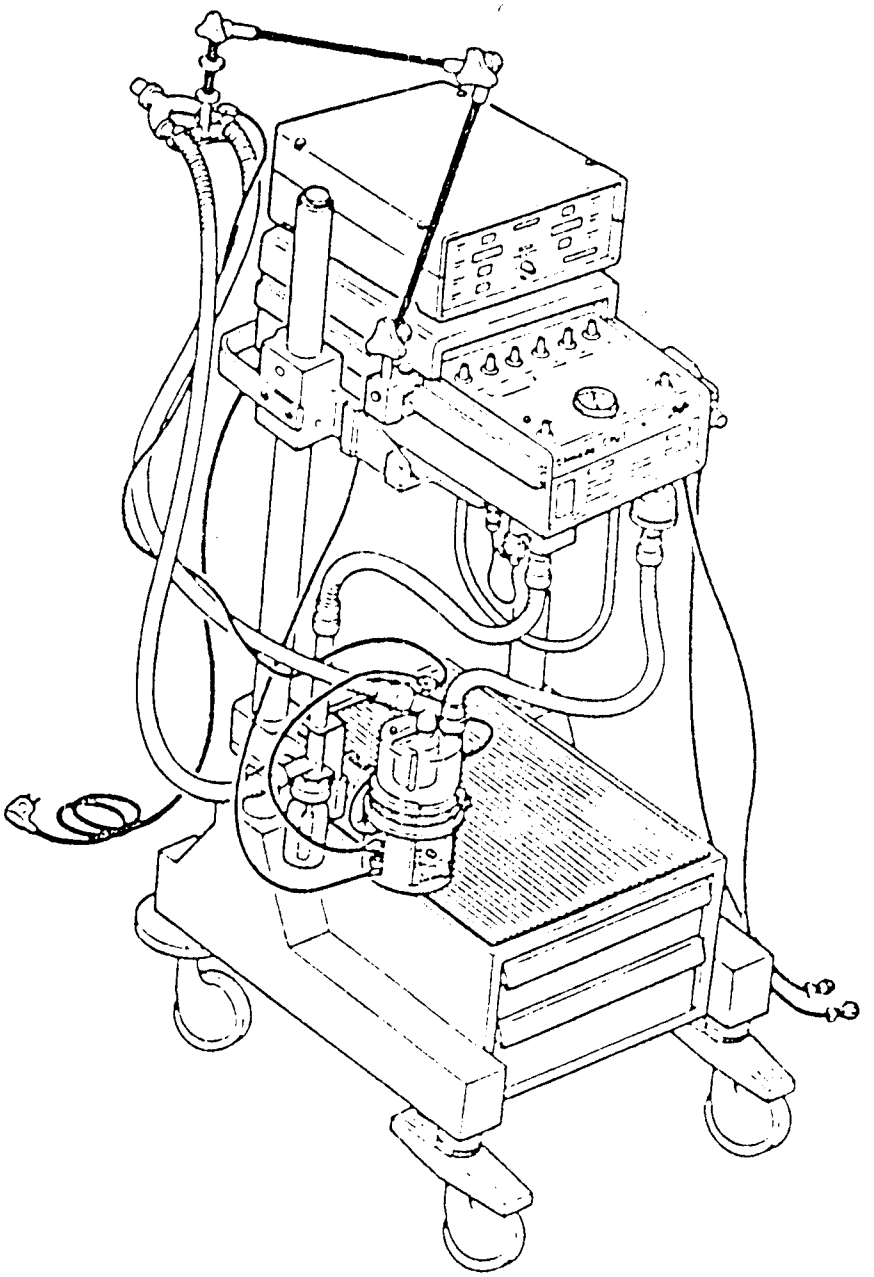


FIG. 2.4

VENTILATOR

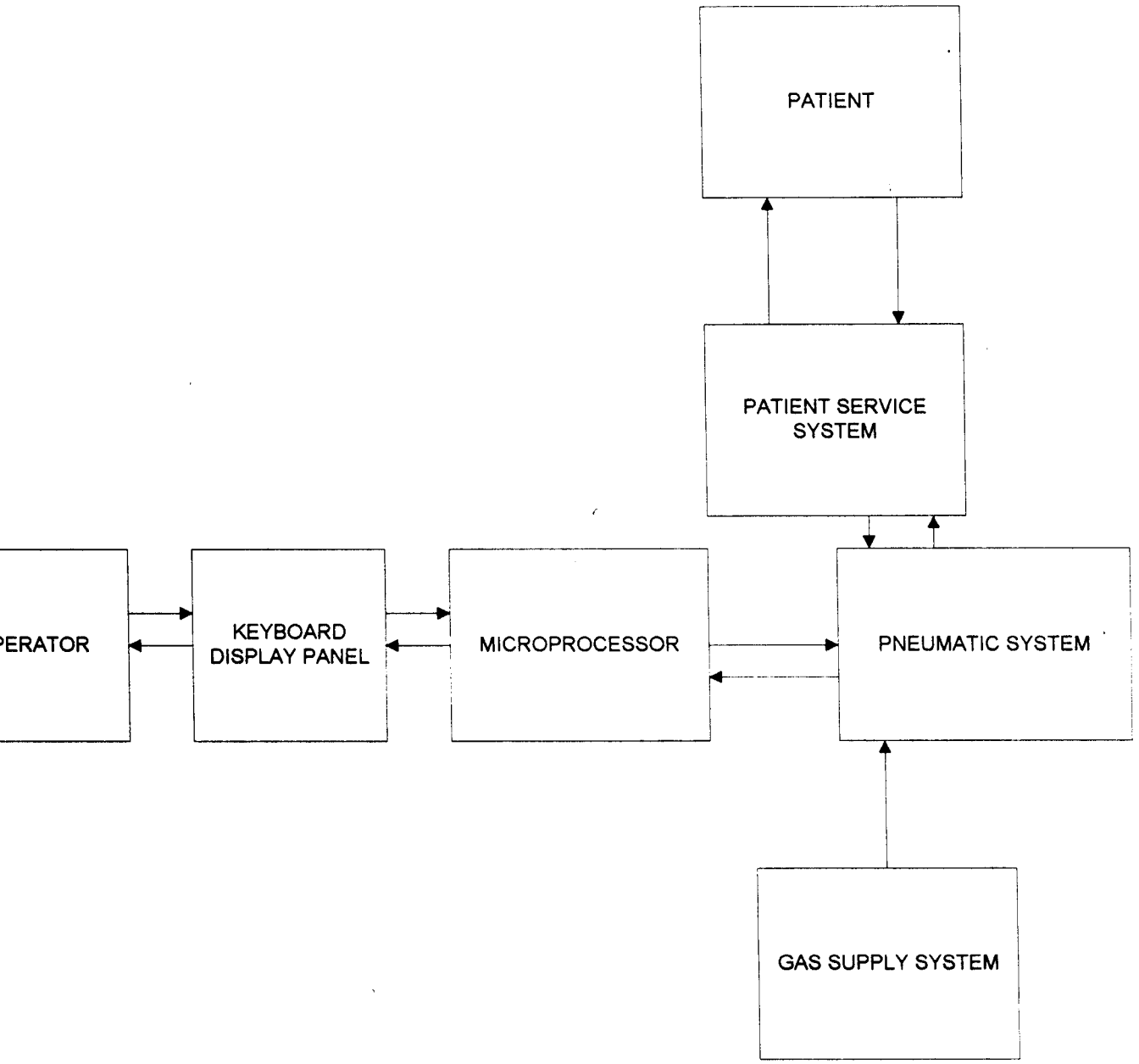


FIG 2.3 BLOCK DIAGRAM OF VENTILATOR INTERFACE

TABLE 2. CLASSIFICATION OF VENTILATORS : MECHANISM OF CONTROL, PRESSURE , AND FLOW PATTERNS;

1. INITIATION OF INSPIRATORY PHASE;

Controlled

Assisted

Controlled / assisted

2. PRESSURE PATTERN: Inspiration- Expiration

Positive- atmosphere

Positive- positive

Positive - negative

3. CONTROL OF MINUTE VENTILATION

Preset (a) minute ventilation

Preset (a) and frequency

Preset (a) and tidal volume

Minute volume divider

Preset tidal volume

Tidal volume (volume cycled)

Pressure limit (pressure cycled)

Time limit (time cycled)

Inspiratory flow

2.5 PARAMETERS INVOLVING RESPIRATION :

We first determine a reference level , chosen to be at the end of a passive expiration . Here the respiratory muscle activity is at a minimum . The volume of the lungs at this time is primarily determined by the elastic forces in the lung tissue and the thorax . This lung volume , the resting expirating level , is a convenient reference point from which changes in lung volume can be measured .(See figure 2.5.1)

2.5.1 LUNG VOLUMES

1. Tidal volume (TV) :

The volume of air inspired and expired with each normal breath (600 ml - 10 %) .

2. Inspiratory reserve volume (IRV) :

The maximal volume of gas that can be inspired beyond the end of a normal tidal volume (3000 ml - 50 %).

3. Expiratory reserve volume (ERV) :

The volume of gas expired by forceful expiration after the end of a normal tidal volume (1200ml -20 %).

4. Residual volume :

The volume of air remaining in the lungs after a maximal expiration (1200 ml -20 %).

2.5.2 LUNG CAPACITIES :

There are four capacities each of which includes two or more of the primary volumes .These lung capacities are given as follows :

1. Vital capacity (VC) :

Maximum amount of that can be expelled from the lungs by forceful effort after inspiration (6000 ml - 80 %)

VC = IRV + ERV + TV .

2 . Total lung capacity (TLC) :

The amount of air contained in the lungs after maximal inspiration (6000 ml - 100 %)

TLC = VC + RV .

3 . Inspiration capacity (IC) :

The maximum amount of air that can be inspired after reaching the end expiratory level (3600 ml -60 %) .

IC = TV + IRV .

4 Functional residual capacity (FRC) :

The volume of air remaining after a normal expiration (24000 ml - 40 %)

FRC = ERV + RV .

All pulmonary volumes and capacities are about 20 to 25 % less in females than in males .

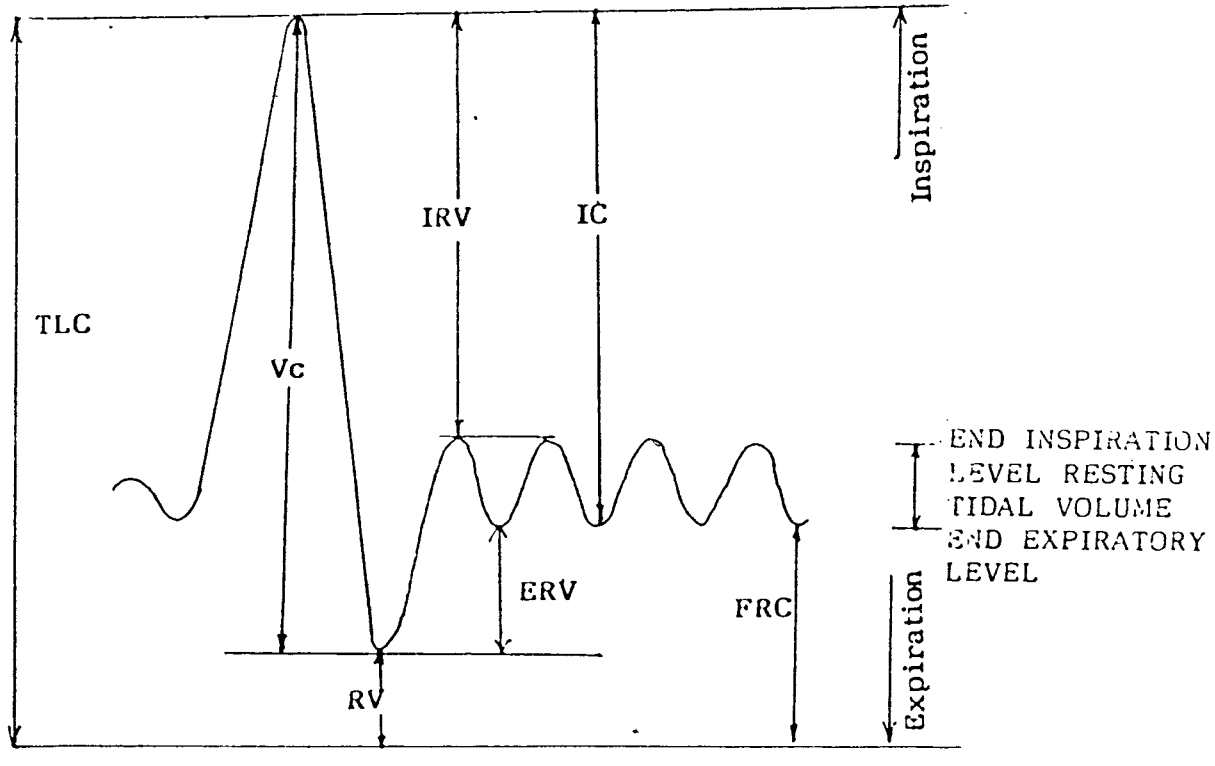


FIGURE 2.2

Lung Volumes and Capacites

2.5.3 *DYNAMIC VOLUMES AND CAPACITIES :*

In addition to the static volumes and capacities already given , several dynamic measures are used to assess pulmonary mechanics . These measures are important because breathing is a dynamic process and the rate at which gases can be exchanged within the lungs is a direct function of the rate at which air can be transported .

A measure of the overall output of the respiratory system is the respiratory minute volume (RNV) . This is the measure of the amount of air inspired during 1 minute at rest . It is calculated by taking the product of the tidal volume and the respiratory frequency .

A number of forced breathing tests are also used to assess the muscle power associated with he breathing and the resistance of the airway . Among them is the forced vital capacity (FVC), which is nothing more than a vital capacity maneur performed as quickly as possible. By definition , the FVC is the total amount of air that can be forcibly expired gas quickly as possible after taking the deepest possible breath .If the measurement is made

with respect to the time required for the maneuer , it is called a timed vital capacity . A measure of the maximum amount of gas that can be expelled in a given number of seconds is called the forced expiratory volume .

CHAPTER 3

PROGRAMMING LANGUAGE C ++

3 . 1 INTRODUCTION :

C ++ was developed by BJARNE STROUSTRUP at the Bell laboratories in 1983 . Originally it was called as " C with class " . C ++ as an enhancement to the C language was developed primarily to facilitate managing programming and maintaining large software projects . There are several C ++ products available TURBO C ++ , BORLAND C ++ ,ZORTECH C ++ , AT & T C ++ , SUN C ++ PREPROCESSOR . Turbo C ++ and Borland C ++ are the most widely used among them . C ++ also uses arrays and pointers in order to be advanced over the other high level languages.

3 .2 OBJECT ORIENTED PROGRAMMING :

OOPS is the most dramatic innovation in software development in the last decade . It ranks in importance with the development of the first higher level languages act the dawn of the computer age . OOPS can be implemented in many languages , it

supports object oriented programming . The most important features of OOPS are DATD HIDING , ENCAPSULATION , POLYMORPHISM INHERITANCE and REUSABILITY .

3.3 CLASSES AND OBJECTS :

Class is a new data type very similar to structures . A structure is a user defined data type . It is a conglomerate of logically related data items . Unlike arrays , they can be made of members of different data types . A class contains data members as well as the function members . They can be defined either in the public section for access to any function of the program or in the private section , where access is allowed to member functions of the class only . A class is a user defined data type and an object is an instance of a class . Data members of different objects of the same class occupy different memory area , but function members of different objects of the same class share the same set of functions . While designing a class , the designer can include within the class , a special member function that is executed when ever an object of the class is created . Such functions are called as

CONSTRUCTORS . A member function can be specified as a constructor by assigning to it the same name as the name as the name of the class . And in C ++ we have the DESTRUCTORS , they are the functions that are complementary to constructors . It cannot take arguments or specify a return value , or explicitly return a value . This is invoked either when an object of the class goes out of scope , or when the memory occupied by it is deallocated using the delete operator .

3.4 FEATURES OF OOPS :

3.4.1 DATA ABSTRACTION :

Structures as seen are used to represent the data in a program like a list , of values or the description of the employee . This logical picture hides the details of how the data is eventually stored in the memory . Because of this only it is also called as DATA HIDING. The computer stores the data in the form of binary

bits and bytes but humans can think about the information's in the form of characters .Data abstraction helps in separating the computers view of data from that of human . It also represents the information in terms of its interface with the user .It protects the integrity of the data when putting the data in the private section .

3.4.2 DATA ENCAPSULATION :

Class definition is used to specify a class design . The class definition is modeled after a structure template and can include data members and function members . This definition has a private section , and members declared in their section can be accessed only through the member functions . The definition has also a public section , and members declared there can be accessed by an outside program . Typically , data members go into the private section and the member functions go into the public section . Bundling the data together with the member functions describing how the data can be used is called as ENCAPSULATION .

3.4.3 POLYMORPHISM :

Polymorphism lets us create multiple definitions for operators and functions , with the programming context which definition is used . The term polymorphism means having many forms . We have two types of polymorphism's they are

(1) FUNCTIONAL POLYMORPHISM :

Functional polymorphism is also called as FUNCTIONAL OVERLOADING because we can attach more function in the same name , thus over loading the name . So function polymorphism lets a function has many forms . We can use this function over loading to design a family of functions that to essentially the same thing , but using different argument lists . The functional approach , how ever, is more powerful , for it enables us to use different data types of arguments as well as different numbers of arguments .

(2) OPERATOR OVERLOADING :

Operator over loading is another example of C ++ polymorphism . It extends the overloading concepts to operators , to assign multiple meanings to C ++ operators . Actually , many C ++ operators are already over loaded . C ++ extends the over loading to user defined types . For example + symbol is used to add two objects . Again , the compiler will use the number and type of operands to determine which definition of addition to use . Over loaded operators often can make code look more natural .

3.4.4. INHERITANCE :

A new classes are derived from the old class , with the derived class inheriting the property of old ones , including the methods of the old class , called a base class . It is easier than

designing a new class . We can add new properties and even we can redefine the member function as per our requirement .

3.4.5. REUSABILITY :

C ++ classes brings us a higher level reusability . Many vendors now offer class libraries , which consists of class definitions and implementations . Because a class encapsulates data representation with class methods , it provides a more integrated packages than does a class library . It is one of the main goals of OOPS . The property of using the code which is already used in the project is called as reusability . Employing old code saves time and , because it has been used and tested , can help suppress the introduction of bugs into program . With all the above we have a special function called as friend function , it is a nonmember function that 's granted access to a class's private members similarly , a friend class is a class whose member functions can access another class's private members .

The optimization of oxygen using genetic algorithm can be implemented on Turbo C++ which is the most advanced over other higher level languages.

CHAPTER 4

GENETIC ALGORITHM

4.1 INTRODUCTION :

The rapid growth in the field of computer has lead to the development of the number of fields . Genetic algorithms was developed in late 1960 's . Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics . . GA 's have been developed by JOHN HOLLAND . Though it was efficient in producing the required result , it was in decline because of the numerous computations involved . Now this job is taken up by the a computers and hence genetic algorithms are in the upcoming trend . This uses the DARWINIAN PRINCIPLE of " survival to the fittest " . The theme of GA ' s is the ROBUSTNESS , a balance between efficiency and efficacy necessary for survival in many different environments. Search methods like calculus based , enumerative and random method are conventional methods which does not meet our robustness requirement . Thus we go for GA's which has theoretically and empirically proven to provide robust search in complex spaces .

Features of self repair , self guidance and reproduction are the rule in biological systems . These are the features which make GA's as a sophisticated optimization technique.

EVOLUTIONARY COMPUTATION

Evolutionary computations can be classified as based on the evolution models they use as follows :

GENETIC ALGORITHMS (GA)

Genetics algorithms models evolution at the level of gene propagation and this widely used in U.S.

EVOLUTION STRATERGIES (ES)

Evolution strategy models how evolution optimizes individuals to exploit their environment . This has been developed and studied much in West Germany.

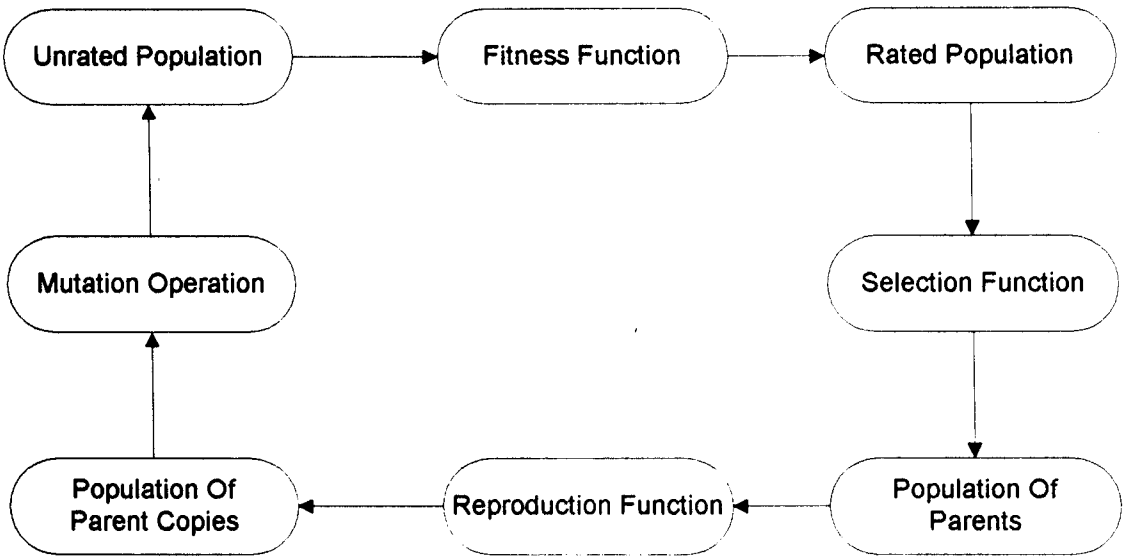


FIG 4.1

BLOCK DIAGRAM- EVOLUTIONARY TECHINIQUE

EVOLUTIONARY PROGRAMMING (EP)

This is one of the earliest methods and is still popular .It models evolution based on multiple species competing for shared resources .

All these evolutionary computational techniques used the general procedure shown in Fig 4.1.First , every member of the given set of candidates is rated as how good it is using a fitness function . Evolutionary computation attempt to maximize this function gradually to improve the solution . A subset of this set is selected to act as parents and the choice of the method depend on many factors . One simple method is to select the best half of the members as parents . 'Roulette wheel selection ' method chooses a members with a probability equal to fitness divided by the total fitness of all the members and in ' tournament ' method one selects at random one subset from which a member with largest fitness function is selected to be parent .

Next , a set of operations are performed over the parents to modify their contents and this is the most important features of evolutionary computations. These operations are generally called as mutations which may include inversion , point mutation or cross over

The resulting children become a part of subsequent population and the cycle is repeated again and again till the value of fitness function reaches a desired. One cycle of the process, called generation , is shown .Generally the initial population is chosen at random and the problem is solved by several cycle of selection of the parents and subsequent motivation . A typical algorithm for such computational scheme is given below:

1. Choose a population based on the information the problem domain or at random (If such information is available) such that each individual many represent a possible solution to the problem .
2. Evaluate the fitness of each individual .
3. Select the parents from the population .
4. Perform cross over and mutations with the parents to produce children and add to the 'population .
5. Evaluate the children's fitness .
6. Output the solution if one is formed with the desired fitnesselse go to step 3.

4.2 OPTIMIZATION :

Optimization seeks to improve performance towards some optimal point or points . The definition has two parts (1) we seek

improvement to approach an (2) optimal point . In judging optimization procedures we focus on convergence .

Genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite length string over some finite alphabet . GA's use random choice as a tool to guide a search towards the optimal point .

4.3 HOW DO GA'S DIFFER FROM TRADITIONAL METHODS :

GA ' s differ from more normal optimization and search procedures in four ways . They are ,

1 . GA 's work with a coding of parameter set and not the parameter themselves .

2 . GA's search from a population of points and not on a single point .

3 . GA's use objective function , not derivatives or other auxillary knowledge .

4 . GA's use probabilistic transition rules not deterministic rules.

4.4 FUNDAMENTAL CONCEPTS

GAs directly manipulate a population of strings. Let's see what a string is and a lot more about it. Let us consider a seven bit string A= 0 1 1 0 1 1 0 which can be represented symbolically as A = a1 a2 a3 a4 a5 a6 a7

Here ai represents a single binary features or detector. A number '0' or '1' taken by a gene is called as the feature value. A number of strings together constitute a population.

4.5 COMPARISION OF NATURAL SYSTEM AND GAs TERMINOLOGY

Chromosomes are long stretches of DNA that carry the genetic information needed build an organism. The strings of artificial genetic systems are analogous to genes. Each gene is a unit of information. Genes are located at positions called as LOCI. At loci genes takes up different values and each is called an ALLELES. In artificial systems, strings are composed of features or detectors, that assume different values located at different values located at different positions of a string.

The total genetic packages is called the GENOTYPE where as it is called a structure in artificial genetic systems. In natural systems, the organism formed by the interaction of the total genetic package with its is called the PHENOTYPE. In artificial genetic systems the structures decode to form a particular parameter set or a possible solution.

Biological system	Artificial genetic system
Chromosome	String
Gene	Features of detector
Allele	Feature value
Locus	String position
Genotype	Structure
Phenotype	Parameter set

A population can be represented with ease with the help of schemata . Schemata is a similarity template . It describes a subset of strings with similarities at certain string positions .Usually strings are binary alphabets (0 ,1) . Concept of schemata is brought in by appending a ' * ' symbol which is a wild card . It can assume the value of either ' 0 ' or ' 1 ' . Consider a schemata * 110 * , this describes a subset with four members (01100 , 011001 , 11100 , 11101) for an alphabet with carnality K and strings with length l there are (K + 1) schemata . For a population with ' n ' strings there are n (K + 1) schemata .

A schemata is described by two properties ,

(i) order

(ii) defining length

ORDER (O (H))

It is the number of fixed positions in the template (in binary ' 0 ' or '1 ')

example : 1 * * 1 0 * *

order is 3

DEFINING LENGTH (&(H))

It is the distance between the first and last specific string position .

example : 1 * * 1 0 * *

&(H) = 4

4.6 MECHANICS OF GENETIC ALGORITHMS :

The mechanics of a simple GA's are surprisingly simple , involving nothing more complex than copying strings and swapping partial strings .Simplicity of operation power of effect are two of the main attraction of the GA's approach.

A simple GA that yields good result in many practical is composed of three operators,

1. Reproduction
2. Cross over
3. Mutation

4.6.1. REPRODUCTION :

Reproduction is a process in which individual strings are copied according to their objective function values , f (fitness function) . Copying strings according to their fitness function values means that strings with a higher value have a higher probability of contributing one or more offspring in the next string . The reproduction operator may beimplemented in algorthermic form in a number of ways . The easiest is to create a biased ROULETTE WHEEL where each current string in the population has a roulette wheel slot sized in proportion to its fitness .(Fig 4 .2) .

Figure 4.2 shows roulette wheel with percentages indicating the fittness . As an example , string 1 is given 14.4% of the biased roulette wheel , and each spin turns up string 1 with probability 0.144 . Each time we require another string has been selected for reproduction, an exact replica of the string is made . This string is then entered into a mating pool , a tentative new population for further genetic operator action .

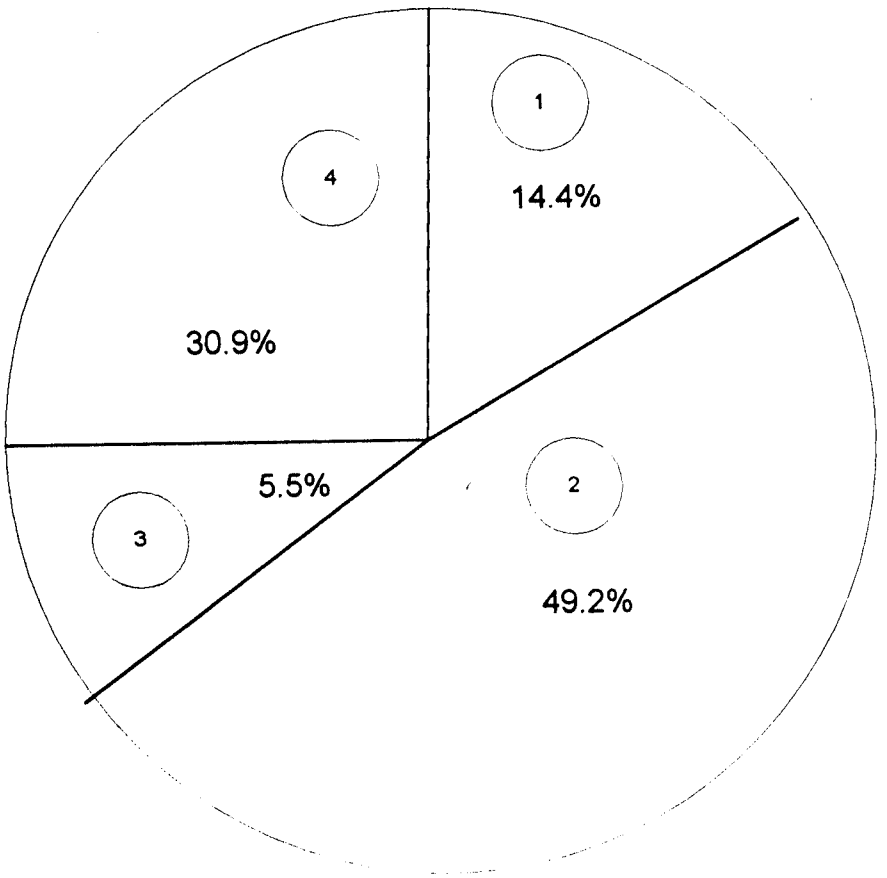


FIG. 4.2

A BIASED ROULETTE WHEEL

4.6.2.CROSS OVER :

After reproduction , simple crossover may proceed in two steps . First , members of the newly reproduced strings in the mating pool are muted at random. Second , each pair of string undergoes crossing over as follows ,an integer position K along the string is selected uniformly at random between I and the string length less one [I , I-1] .Two new strings are created by swapping all characters between positions K+1 and I inclusively . For example , consider strings A1 and A2 from our example initial population .

A1=0 1 1 0 | 1

A2=1 1 0 0 | 0

Suppose in choosing a random number between 1 and 4 . We obtain a K=4 (as indicated by the seperator symbol |) . The resulting crossover yields two new strings where the prime (') means the strings are part of the new generation .[Fig 4.3]

A'1 = 0 1 1 0 0

A'2 = 1 1 0 0 1

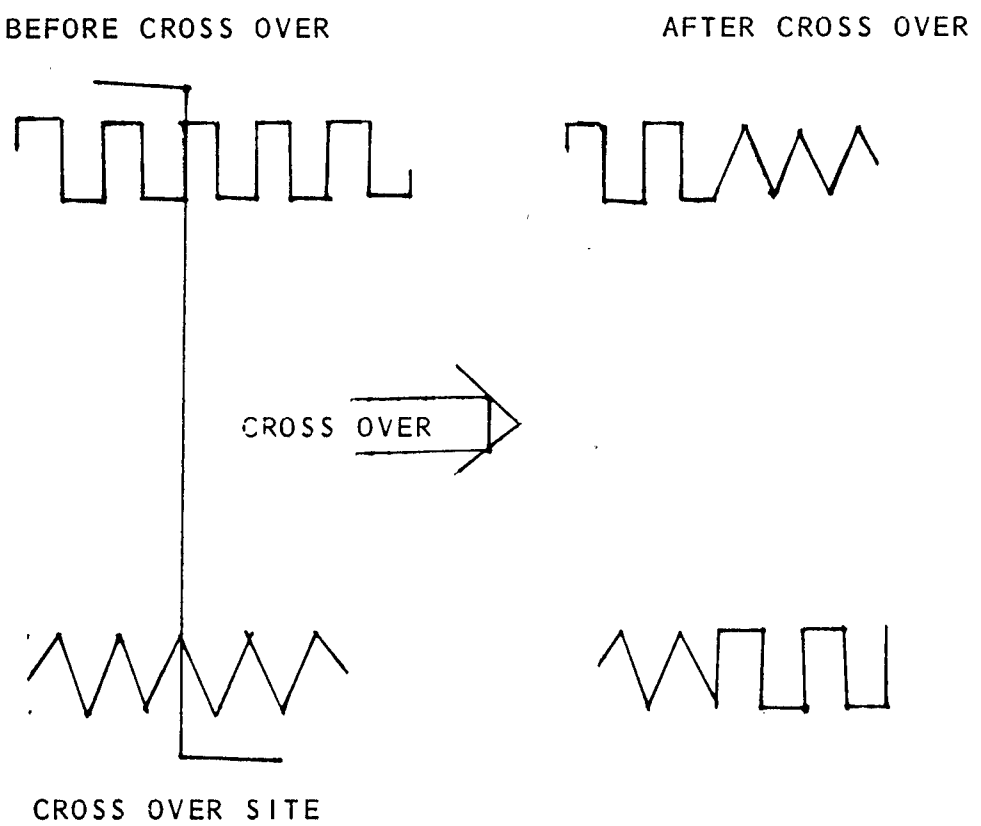


FIG. 4.3

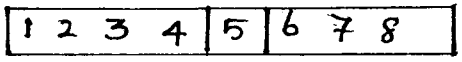
CROSS OVER - DIAGRAMMATIC REPRESENTATION

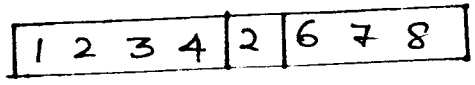
The mechanics of reproduction and crossover both gives GA's much of their power.

4.6.3. MUTATION

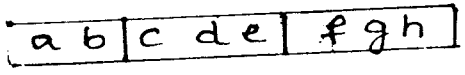
A mutation operator is included in most GA's . In simple GA s a mutation is a random alteration of a value of the string position . This operator helps to gain information that is not available to the rest of the population .The purpose of the mutation operator is to prevent loss of important information by effectively increasing the population diversity. The frequency of mutation to obtain good results in empirical GA studies is on the order of one mutation per thousand bit (position) transfer.

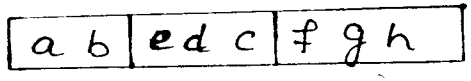
GA also uses 'Point Mutation ' by altering a single feature to some other value at random and 'Inversions ' wherein a randomly selecting portions of feature vector is reversed . Binary string , called ' chromosome ' , is commonly used for feature vector , though string in any other format is permissible. (FIG 4.4)





POINT MUTATION





INVERSION

FIG. 4.4

MUTATION - DIAGRAMMATIC REPRESENTATION

SIMULATION OF A GA :

Consider the problem of maximizing the function

$$f(x) = x - 64x + 100,$$

where x varies from 0 to 63.

This function has a global maximum value of 100 at $x = 0$, as can easily computed. To use a genetic algorithm, decision variables of the problem have to be coded in strings of finite length. For this problem, we can encode the variables as a binary string of length 6. We create an initial population with 4 samples by randomly selecting them from the interval 0 to 63 and encode each sample. A binary string of length 6 can represent any value from 0 to 63 ; hence string length is chosen as 6 for the example . Four encoded samples in
the initial population are :

5 = 0 0 0 1 0 1

60 = 1 1 1 1 0 0

33 = 1 0 0 0 0 1

8 = 0 0 1 0 0 0

These individuals are sorted according to their fitness values . (They are arranged in the descending order of their fitness values). In this case , the fitness value is the same as the cost function value . These sorted individuals are given as:

NUMBER	X	STRING	FITNESS VALUE
1	60	111100	-140
2	5	000101	-195
3	8	001000	-348
4	33	100001	-9

In the 1 st generation , the 1 st and 2nd strings are crossed over at site 3 , (cross over site is randomly selected) to get two new strings :

CROSS OVER SITE	NEW STRING	FITNESS VALUE
111 / 100	111101	-83
000 / 101	000100	-140

Similarly, the third and fourth strings are crossed over at cross over site 2 , to get :

CROSS OVER SITE	NEW STRING	FITNESS VALUE
00 / 1000	000001	37
10 / 0001	101000	-860

Sorting these new individuals we get

NUMBER	X	STRING	FITNESS VALUE
1	1	000001	37
2	61	111101	-83
3	4	000100	-140
4	40	101000	-860

We see that in one generation fitness is improved from -140 to 37 (f(1) > f(60)) . Before proceeding to the next step , the weakest member of the population is replaced by the fittest member of previous population , i .e. , string 1 0 1 0 0 0 that has fitness -860 is replaced by the string 1 1 1 1 0 0 , whose fitness is -140 .

In the 2nd generation , the 1st and the 2nd strings are crossed over at site 1 , we get :

CROSS OVER SITE	NEW STRINGS	FITNESS VALUE
0 / 00001	011101	-915
1 / 11101	100001	-923

Similarly , the 3rd and 4th strings are crossed over at crossover site 3 to get :

CROSS OVER SITE	NEW STRING	FITNESS VALUE
000 / 100	000100	-140
111 / 100	111100	-140

Replacing the weakest member by the fittest member of the previous population and sorting them according to the fitness values , we get :

NUMBER	X	STRING	FITNESS VALUE
1	1	000001	37
2	4	000100	-140
3	60	111100	-140
4	29	011101	-915

In the 3rd generation , the above process of crossover is repeated (not shown) . The new set of strings in the population , after replacement of the weakest by the fittest member of the previous population is given by :

NUMBER	X	STRING	FITNESS VALUE
1	0	000000	100
2	1	000001	37
3	61	111101	-83
4	5	000101	-195

For simplicity , the probability of the mutation is chosen as 0 . We see that as genetic operations continue from one generation to the next , improved solutions evolve . At $x = 0$, $f(x) = 100$, which is the desired result . Here , the problem which could have been solved using conventional optimization algorithms ,is used to illustrate GA operations for the sake of simplicity

Nature has been using GAs for millions of years and along the way it has produced complex , intelligent living organisms capable of reproduction , self - guidance and repair . Although GAs are computationally simple , they have demonstrated their power and capability in optimizing multi- modal , multi - dimensional and multi - objective problems . Hence they should find widespread applications in business , science and engineering .The price paid for global optimization and robustness is the amount of computation required , which is not a problem in the age of fast / parallel computers . GAs use simple computational operations and yet are powerful tools for optimization . Several versions / improvements in GAs are now emerging .

For its power and capability in optimizing objective problems GAs is the best method for optimizing the oxygen level given to the patient.

CHAPTER 5

DESIGN PROCEDURE

This chapter deals with the programming steps involved in the design of the software . Object Oriented Programming concept of Turbo C++ has been used to optimise the oxygen in ventilators . A random search tool " Genetic Algorithm " has been used for optimísation .

The lung parameters which mainly contribute to the requirement of oxygen are

1. Total lung Capacity (TLC)
2. Tidal Volume (TV)
3. Vital Capacity (VC)
4. Forced Expiratory Volume (FEV)
5. Respiratory Rate (RR)

The formulae for these parameters are

1. TLC

For Males:

TLC = 0.78 (height in cms) -- 7.3

For Females:

TLC = 0.079 (height in cms) -- (0.008 * age) -- 7.47

2. VC

For Males:

VC = 27.63 -- (0.112 * age) height

For Females:

VC = 21.78 -- (0.101 * age) height

3. FEV

For Males:

FEV = (0.092 * height) -- (0.032 * age) -- 1.26

For Females:

FEV = (0.089 * height) -- (0.025 * age) -- 1.93

4. RR -- 12 / min

5. TV -- (8 - 10) % of TLC

The algorithm of the program developed is as follows

1. The physical parameters of the patient i. e. sex , age , , height , weight are obtained .

2. The lung parameters are calculated using the pre-existing standard formulae .

3. The population of individuals is initialised .

4. The weightages of the five parameters are calculated .

(i) The first eight bits of an individual of a population gives the weight of TLC .

(ii) The second eight bits of an individual of a population gives the weight of TV .

(iii) The third eight bits of an individual of a population gives the weight of VC .

(iv) The fourth eight bits of an individual of a population gives the weight of FEV .

(v) The last eight bits of an individual of a population gives the weight of RR .

5. The fitness function is used to calculate the oxygen value pertaining to each individual using the formula

Oxygen = TLC * TLC weight + TV * TV weight + FEV * FEV weight + RR * RR weight + VC * VC weight

6. Convergence checking is performed over the population i.e. the difference between the maximum and minimum fitness values should be negligible. If the above condition is true stop generating .

7. The individuals of the populations are selected two at a time for reproduction at random.

8. Mutation is performed on the population of individuals . The probability of mutation is 0.0001

9. The crossover site is selected at random . The crossover operation is performed on the old population and the new population individuals are obtained. The probability of crossover is 0.3. Steps 4 , 5 & 6 are then carried out.

10. The optimal value of oxygen is displayed on the screen .

For carrying out the above steps two header files are defined with required classes and functions.

The two header files used are

1. CLASS . H
2. RANDOM . H

CLASS . H

This header file has definitions of the classes used . The classes used are

1. PATIENT CLASS
2. CHROMOSOME CLASS
3. INDIVIDUAL CLASS
4. POPULATION CLASS

PATIENT CLASS

The member variables of this class are

1 . NAME

2 . SEX

3 . AGE

4 . HEIGHT

5 . WEIGHT

The member functions of patient class include the below functions along with constructors and destructors.

1 . get_details ( )

Gets details of the patient . It uses graphics header file .

2. tlc ( )

Returns total lung capacity of the patient using the formula .

3. tv ( )

Returns tidal volume .

4. vc ( )

Returns vital capacity .

5. fev ( )

Returns forced expiratory volume .

6. operator << (ostream & , patient &)

'<<' operator overloading to print out the details of patient

CHROMOSOME CLASS

An array of ALLELE ' S make up a chromosome . Character is type defined as ALLELE .An ALLELE takes up a value of ' 0 ' or ' 1 ' as the coding used for the programming is binary coding . The coefficients of the fitness equation are coded into a chromosome . The functions are

1 . tlc_wt ( )

Returns the value of the coefficient of tlc .

2 . vc_wt ( )

Returns the coefficient of vc .

3 . tv_wt ( )

Returns the coefficient of tv .

4 . fev_wt ( )

Returns the coefficient of fev .

5. rr_wt ( )

Returns the coefficient of respiratory rate .

6 . ret_chrom ( )

Returns the string of chromosome .

In this class ' << ' operator and ' = ' operator are overloaded and a friend function (friend of patient class and chromosome class) oxy ( ) is used which uses the parameter values obtained in patient class , the coefficient values obtained in chromosome class and returns the value of oxygen .

INDIVIDUAL CLASS

The members of this class include variables

1 . Chromosome chrom

2 . Fitness

3 . Parent 1

4 . Parent 2

5 . Crossover site

The functions used are

1 . get_oxy (patient & , chromosome &)

Returns the value of oxygen for an individual . This uses the friend function oxy ( ).

2 . get_p1 ( )

Returns the index of the parent 1 .

3 . get_p2 ( )

Returns the index of the parent 2 .

4. get_xsite ( )

Returns the value of crossover site .

' << ' operator is overloaded to display the values of individual attributes .

POPULATION CLASS

The population class consists of an array of individuals of size MAX_POP .

1 . scene_fit

2 . max_fit

3 . min_fit

4. max_fit_index

5. min_fit_index

6. converge

7 . static patient p

The functions include

1 . init_pop ( )

Initialises all the attributes of population . Individuals are initialised using flip ( ) function.

2 . max_min_sum_fit ( )

It is the values of max_fit , min_fit and sum_fit .

3 . get_sum_fit ( )

Returns the value of sum_fit .

4 . get_max_fit ( )

Returns the value of max_fit .

5 . get_min_fit ( )

Returns the value of min_fit .

6 . get_max_fit_index ( )

Returns the index of individual with max_fitness .

The functions include

1 . init_pop ( )

Initialises all the attributes of population . Individuals are initialised using flip ( ) function.

2 . max_min_sum_fit ( )

It is the values of max_fit , min_fit and sum_fit .

3 . get_sum_fit ( )

Returns the value of sum_fit .

4 . get_max_fit ( )

Returns the value of max_fit .

5 . get_min_fit ( )

Returns the value of min_fit .

6 . get_max_fit_index ( )

Returns the index of individual with max_fitness .

Returns an integer value between the ulimit and llimit .

4 . flip (double probability)

Returns ' 0 ' or ' 1 ' representing the ' tail ' or ' head ' of a coin toss .

5. advance_random ( )

Alters the double values in random array .

HARDWARE & SOFTWARE CONFIGURATION

The hardware configuration required to run this software is

processor : 80486

CPU clock : 66 Mhz

Base memory : 640Kb

Extended memory : 4Mb

Monitor Type : monochrome

The software requirement to run this software is

OS : MS DOS Version 6 . 22

Language : TURBO C++ Ver 2 . 0

OPTIMISATION OF OXYGEN

PATIENT DETAILS

NAME:

AGE: SEX: M/F

WEIGHT: HEIGHT:

OK: [y/n]

MESS:

CONCLUSION

A ventilator is a support system for a respiratory stem. Anaesthesiologists use it to fix the amount of ygen to be given to the patient. The previous methods ing a trial and error was ineffective and error prone. ere the genetic algorithm which is randomised optimisation echnique is incorporated. It comprehensively excels the xisting system by providing accurate guidelines to naesthesiologists to help them fix the optimum amount of xygen. The doctors can easily detect the parameters which s varying and also the amount of variation. Having this ecord anaesthesiologists can give treatment accordingly.

In future this project can be implemented by having nore number of parameters which involves in the respiratory system.

REFERENCES

1. Genetic Algorithm in search, optimisation & machine learning

-David E. Goldberg.

2. Genetic search : Analysis using fitness moments

-M. Srinivas & L. M. Patnaik
IEEE transaction on knowledge & Data Engineering Volume No. 1 February 1996.

3. Resonance - Aug. 1996.

4. Biomedical Instrumentation & Measurements
-Cromurell

5. C++ Primer Plus
-Stephen Prata

6. Respiratory System diagonosis through Windows
- A .Sukesh Kumar & Dr. A. Kandasamy.
Technology - Journal of PSG College of Technology Mar. 1996.

7. B. R.Garbe and T. T. Chapman, M .D.,
" The simple measurements of Lung Ventilation"
Vitalograph Ltd., 10th Edition, Copyright, 1986.

8. Leslie Cromwell, Fred J. Weibell & Erich A Pfeiffer,
"Biomedical Instrumentation & Measurements",2nd edition,
Prentice Hall Of India.

9. Barry N. Feinberg, " Applied Clinical Engineering".

CHAPTER 9

SOURCE CODE

```
//front screen design//
#include <graphics.h>
#include<fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include<dos.h>

int main(void)
{
   struct date d;
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int left, top, right, bottom,left1,top1,right1,bottom1;
   char colstr[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "c:\\tcplus\\bin" );

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   left = getmaxx() / 2+300;
   top = getmaxy() / 2 +200;
   right = getmaxx() / 2 -300;
   bottom = getmaxy() / 2 -200;


   left1=640/2 +275;
   top1=150/2 +50;
   right1=640/2 -275;
   bottom1=150/2 -30;
   /* draw a rectangle */
  rectangle(left,top,right,bottom);
  rectangle(left1,top1,right1,bottom1);

    /* select a text style */
 settextstyle(GOTHIC_FONT, HORIZ_DIR, 4);
```

```
/* move to the text starting position */
moveto(55, 50);

/* output some normal text */
outtext("OPTIMISATION ");
outtext("  OF ");
outtext("  OXYGEN ");

settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
moveto(210,95);
outtext("PATIENT");
outtext(" DETAILS");

   left1=550/2 +280;
  top1=200/2 +80;
  right1=550/2 -225;
  bottom1=375/2 -30;
  /* draw a rectangle */
 rectangle(left1,top1,right1,bottom1);
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(63,160,"PATIENT NO:");

  left1=550/2 +280;
  top1=300/2 +80;
  right1=550/2 -225;
  bottom1=475/2 -30;
  /* draw a rectangle */
 rectangle(left1,top1,right1,bottom1);
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(63,210,"NAME:");

  left1=550/2 +280;
  top1=400/2 +80;
  right1=550/2 -225;
  bottom1=575/2 -30;
  /* draw a rectangle */
 rectangle(left1,top1,right1,bottom1);
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(63,260,"AGE:");
outtextxy(300,260,"SEX:    M/F");

  left1=550/2 +280;
  top1=500/2 +80;
  right1=550/2 -225;
  bottom1=675/2 -30;
  /* draw a rectangle */
 rectangle(left1,top1,right1,bottom1);
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(63,310,"WEIGHT:");
outtextxy(300,310,"HEIGHT:");

 left1=550/2 +150;
  top1=600/2 +80;
  right1=550/2 -150;
  bottom1=775/2 -30;
```

```
#include<iostream.h>
#include<bios.h>
#include<graphics.h>
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>

main()
{
   char *str,*str2;
   char *str1='\0';
   int i;

   int gdriver = DETECT, gmode, errorcode;
    /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "c:\\tcplus\\bin" );

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)  /* an error occurred */
  {
     printf("Graphics error: %s\n", grapherrormsg(errorcode));
     printf("Press any key to halt:");
     getch();
     exit(1); /* terminate with an error code */
  }
   /* select a text style */
//settextstyle(GOTHIC_FONT, HORIZ_DIR, 4);
moveto(10,200);

 for(i=0 ; str[i]!=13 ; i++)
   str[i]=bioskey(0);

  //  printf("%d", str[i]);
    strcat(str,str1);
    printf("%s",*str);
    str2=str1;
   // cout<< *str2;
    outtext(str);

 //   char destination[25];
//char *blank = " ", *c = "C++", *turbo = "Turbo";
  getch();
  closegraph();
  }
```

```
#include<iostream.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>

#define MAX_GEN 50
#define MAX_POP 240
#define MAX_STRING 40
#define TOO_SMALL 0.005

double p_cross=0.3;          // * * probability of cross over
double p_mut=0.0001;         // * * probability of mutation
int n_cross,n_mut;           // * * no. of cross over &mutation

#include "class.h"

void main(){

int i,j,k,mate1,mate2,j_cross;
int select(double sum_fit,population &p);
ALLELE mutation(ALLELE a);
double get_fit(patient &pt,chromosome &chrom);
population old_pop,new_pop;

        // * * initialise the population * *

new_pop.init_pop();

// ** Perform Genetic  Operations on the Individuals of the population ***

for(i=0; i<MAX_GEN,new_pop.get_converge() > TOO_SMALL ;i++){
        old_pop=new_pop;        // '=' operator overload


        for(j=0;j<MAX_POP;j+=2){

        //* * reproduction using roulette wheel

                mate1=select(old_pop.get_sum_fit(),old_pop);
                mate2=select(old_pop.get_sum_fit(),old_pop);

        //* * cross over && mutation simultaneously

        if(flip(p_cross)==1){
                j_cross=i_rand(MAX_STRING-2,0);
                n_cross++;
        }
        else
                j_cross=MAX_STRING-1;
        for(k=0;k<=j_cross;k++){
          new_pop.indi[j].chrom.allel[k]=mutation(old_pop[mate1].chrom[k]);
          new_pop.indi[j+1].chrom.allel[k]=mutation(old_pop[mate2].chrom[k]);
        }

        if(j_cross!=MAX_STRING-1){
          for(k=j_cross+1;k<MAX_STRING;k++){
```

```
                new_pop.indi[j].chrom.allel[k]=mutation(old_pop[mate2].chrom[k]);
                new_pop.indi[j+1].chrom.allel[k]=mutation(old_pop[mate1].chrom[k]);
            }
        }

                // * * plenish the population members

                new_pop.max_min_sum_fit();
                new_pop.indi[j].p1=mate1;
                new_pop.indi[j].p2=mate2;
                new_pop.indi[j+1].p1=mate1;
                new_pop.indi[j+1].p2=mate2;
                new_pop.indi[j].xsite=j_cross;
                new_pop.indi[j+1].xsite=j_cross;
    }//end of j loop
} // end of main()


// * * Roulette wheel selection

int select(double sum_fit,population &p){
        double rand,sum=0.0;
        int j;
        rand=f_rand()*sum_fit;
        for(j=0;j<MAX_POP,sum<rand;j++)
                sum+=p[j].get_fit();
        return (j-1);
}

// * * Mutation

ALLELE mutation(ALLELE a){
        short mutate;
        mutate=flip(p_mut);
        if(mutate){
                n_mut++ ;
                if(a=='1')
                        return '0';
                else
                        return '1';
        }
        else
                return a;
}

double get_fit(patient &pt,chromosome &chrom) {
        double oxy(patient &, chromosome &);
        return oxy(pt,chrom );
}
```

```
        cout<<"WRONG ENTRY,ENTER AGAIN";
        flag=0;
      }

  }while(flag==0);
  getch();
   exit(0);
}
```

```
#include"random.h"
#include<iostream.h>
#include<stdlib.h>
#include<string.h>

#define MAX_POP 240
#define MAX_STRING 40

class chromosome;       //forward declaration

//PATIENT CLASS

class patient{
        int pno;
        char *pna;
        char sex;
        double ht;
        double wt;
        int rr;
        int age;

 public:
        patient()       {pna=new char[30];}
        ~patient()      { delete pna; }

        void getdetails();
        char* name()    {return pna;}
        int no() {return pno;}
        char p_sex()    {return sex;}
        double p_ht()   {return ht;}
        double p_wt()   {return wt;}
        int p_rr()      {return rr;}
        int p_age()     {return age;}
        double tlc();
        double vc();
        double tv();
        double fev();
//      friend chromosome;
        friend ostream & operator<<(ostream &,patient &);
        friend double oxy(patient &,chromosome &);
        };

 void patient::getdetails(){
        cout<<"enter no,name,sex,ht,wt,age,hb,rr"<<endl;

        cin>>pno;
        cout<<"--------\n";
        cin>>pna;
        cout<<"--------\n";
        cin>>sex;
        cout<<"--------\n";
        cin>>ht;
        cout<<"--------\n";
        cin>>wt;
        cout<<"--------\n";
```

```
        cin>>age;
        cout<<"--------\n";
//      cin>>hb;
//      cin>>rr;
//      fstream finout("patient.dat",ios::app);
//      finout.write((char*) &p1,sizeof p1);
//      finout.close();
        }

double patient ::tlc(){
        if(sex=='f')
                return (0.079*p_ht()-0.008*p_age()-7.49);
        else
                return (0.078*p_ht()-7.3);
        }

double patient::tv(){
                return ((0.08*tlc()+0.1*tlc())/2);
        }

double patient::vc(){
        if(sex=='f')
                return (4.664*p_ht()-0.024*p_age()-3.284);
        else
                return(6.103*p_ht()-0.028*p_age()-4.654);
        }

double patient::fev(){
        if(sex=='f')
                return (0.089*p_ht()-0.025*p_age()-1.93);
        else
                return(0.092*p_ht()-0.032*p_age()-1.26);
        }

//      a[i]=pow(fh[i],0.725)*pow(fw[i],0.425)*0.007184;    //



ostream & operator<<(ostream &os,patient &p){
        os<<"name    :"<<p.name()<<endl;
        os<<"no      :"<<p.no()<<endl;
        os<<"ht      :"<<p.p_ht()<<endl;
        os<<"wt      :"<<p.p_wt()<<endl;
        os<<"sex     :"<<p.p_sex()<<endl;
        os<<"age     :"<<p.p_age()<<endl;
        return os;
}


//CHROMOSOME CLASS


typedef char ALLELE;

class chromosome{
 public:
```

```
        ALLELE *allel;

        chromosome();
        ~chromosome();

        chromosome ret_chrom();
        void disp_chrom();
//      void get_chrom();
        int tlc_wt();
        int tv_wt();
        int fev_wt();
        int vc_wt();
        int rr_wt();

        ALLELE operator[](int i) {return allel[i];}
        void operator=(chromosome &);
//      friend ostream & operator<<(ostream &,chromosome &);
//      friend patient;
        friend double oxy(patient &,chromosome &);
        };

chromosome::chromosome(){
        allel=new ALLELE[MAX_STRING];
        int i;
        for(i=0;i<MAX_STRING;i++)
                allel[i]='0';
        }

chromosome::~chromosome(){
        delete allel;
        }

int chromosome::tlc_wt(){
        int i,acc=0;
        int pow_2=1;
        for(i=7;i>=0;i--){
                if(allel[i]=='1'){
                        acc+=pow_2;
                        pow_2*=2;
                        }
                else
                        pow_2*=2;
                }
        return acc;
        }
int chromosome::tv_wt(){
        int i,acc=0;
        int pow_2=1;
        for(i=15;i>=8;i--){
                if(allel[i]=='1'){
                        acc+=pow_2;
                        pow_2*=2;
                        }
                else
                        pow_2*=2;
                }
```

```
        return acc;
        }

int chromosome::vc_wt(){
        int i,acc=0;
        int pow_2=1;
        for(i=23;i>=16;i--){
                if(allel[i]!='0'){
                        acc+=pow_2;
                        pow_2*=2;
                        }
                else
                        pow_2*=2;
                }
        return acc;
        }

int chromosome::fev_wt(){
        int i,acc=0;
        int pow_2=1;
        for(i=31;i>=24;i--){
                if(allel[i]!='0'){
                        acc+=pow_2;
                        pow_2*=2;
                        }
                else
                        pow_2*=2;
                }
        return acc;
        }

int chromosome::rr_wt(){
        int i,acc=0;
        int pow_2=1;
        for(i=39;i>=32;i--){
                if(allel[i]=='1'){
                        acc+=pow_2;
                        pow_2*=2;
                        }
                else
                        pow_2*=2;
                }
        return acc;
        }

ostream &operator<<(ostream &os,chromosome &c){
//void chromosome::disp_chrom(){
//      cout<<"chrosome is\n";
        for(int i=0;i<40;i++)
                os<<c[i];       //overloaded '[]'
        return os;

}

void chromosome::operator=(chromosome &c){
        int i;
```

```
        for(i=0;i<MAX_STRING;i++)
                allel[i]=c[i];      //"[]"operator overload
}

chromosome chromosome::ret_chrom(){
        int i;
        chromosome c;
        for(i=0;i<MAX_STRING;i++)
                c.allel[i]=allel[i];
        return c;
}
 double oxy(patient &p,chromosome &c){
        double fit;
        fit=p.tlc()*c.tlc_wt()+p.tv()*c.tv_wt()+p.vc()*c.vc_wt()+p.fev()*c.fev_wt()+p.p_rr()*c.rr_wt()
;
        return fit;
        }

 // double get_oxy(patient &,chromosome &);
 //INDIVIDUAL CLASS

 class individual{
   public:
          chromosome chrom;
 //       double oxy;     //from this fitness is to be calculated
          double fit;
          int p1,p2;
          int xsite;

          individual()      { }
          ~individual()     { }
          void get_indi();
          void disp_indi();

          chromosome get_chrom()        {return chrom;}
 //       double ret_oxy()          {return oxy;}
          double ret_fit()  {return fit;}
          int get_p1()              {return p1;}
          int get_p2()              {return p2;}
          int get_xsite()           {return xsite;}

          friend ostream &operator<<(ostream &,individual &);
          double get_fit(patient &pt,chromosome &chrom) {
                  double oxy(patient &, chromosome &);
                  return oxy(pt,chrom );
          }
   };

   void individual::get_indi(){          //plenishes values of individual
          int acc=0;                             //'pt' & 'chrom' ,the rest plenished
          int pow_2=1;                  //in population class
          int i;
   //     pt.getdetails();
   //     chrom.get_chrom();
          chromosome c1;
          c1=chrom.ret_chrom();             //operator'=' overload
```

```
        for(i=39;i>=0;i--){
                if(c1[i]=='1')      //operator'[]' overload
                        acc+=pow_2;
                pow_2*=2;
                }
        }

ostream &operator<<(ostream &os,individual &i){
        chromosome c;
        c=i.get_chrom() ;
        os<<"chromosome is"<<c;
//      c.disp_chrom();
//      os<<"oxy for that weightage: "<<i.ret_oxy()<<endl;
        os<<"fitness is          : "<<i.ret_fit()<<endl;
        os<<"Parent 1 is         : "<<i.get_p1()<<endl;
        os<<"Parent 2 is         : "<<i.get_p2()<<endl;
        os<<"cross over site is  : "<<i.get_xsite()<<endl;
        return os;
}


 //POPULATION CLASS

 class population{
        individual *indi;
 //     double avg_oxy;
        double converge;
        double sum_fit;
        double max_fit;
        int max_fit_index;
        double min_fit;
        int min_fit_index;
        static patient p;
   public:
        void init_pop();
        population()            {indi=new individual[MAX_POP];}
        ~population()           {delete indi;}

        void max_min_sum_fit();
        void disp_pop();
        void disp_fit();

        double get_converge()  {return converge;}
        individual *get_indi()  {return indi;}
        double get_sum_fit()    {return sum_fit;}
        double get_max_fit()    {return max_fit;}
        int get_max_fit_index() {return max_fit_index;}
        double get_min_fit()    {return min_fit;}
        int get_min_fit_index()  {return min_fit_index;}

        individual operator[](int i)        {return indi[i];}
        void operator=(population &p);
        double get_fit(patient &p,chromosome &c);

  };
```

```
void population::init_pop(){
        int i,j,k;
        double get_oxy(patient &, chromosome &);
//      double max_fit,min_fit;
        char s;
        chromosome c;
//      patient p;
        p.getdetails();
        double d;
        init_rand();
        cout<<"initialise the individual pointer\n";
        for(j=0;j<MAX_POP;j++){
                cout<<"INDI  "<<j<<endl;
                for(k=0;k<MAX_STRING;k++) {
                        indi[j].chrom.allel[k]=flip(0.5)+'0';
                        cout<<"flip...allel["<<k<<"]="<<indi[j].chrom[k]<<"\n";
                        cin>>s;
                }
            cout<<"chrom "<<j;
            cout<<indi[j].get_chrom()<<"\n";  //operator '<<' overload
//          c=indi[j].chrom.ret_chrom();
//          c.disp_chrom();
//          cout<<"oxy."<<j;
//          cin>>s;
            indi[j].fit=get_fit(p,c);     //write to class
            cout<<"ret_fit after get_fit"<<indi[j].ret_fit()<<endl;
            cin>>s;
            indi[j].p1=0;
            indi[j].p2=0;           //initialise members of indi
            indi[j].xsite=0;        //except for fitness
            cout<<"indi["<<j<<"] initialised\n";
            cin>>s;
        }

//********* fitness of individuals ********

        this->max_min_sum_fit();

        cout<<"max fit"<<this->get_max_fit()<<endl;
        cout<<"min fit"<<this->get_min_fit()<<endl;
        cout<<"sum fit"<<this->get_sum_fit()<<endl;   //this pointer
        cin>>s;

}

void population::max_min_sum_fit(){
        int i;
        double d;
        char s;
        cout<<"chromosome 0 "<<indi[0].get_chrom()<<endl;
//      c.disp_chrom();                 //operator '<<' overload
        cin>>s;
        max_fit=min_fit=indi[0].ret_fit();  //getting from class
        sum_fit=0;
        for(i=1;i<MAX_POP;i++){
                cout<<"chromosome  "<<i<<"   "<<indi[i].get_chrom()<<endl;
```

```
#include<iostream.h>
#include<string.h>
#include<math.h>
double old_rand[55], new_rand;
int init;


void init_rand(){
        double f;
        f=(double) 1/55;
        cout<<"f= "<<f<<endl;
        int i,j;
        init=0;
        old_rand[0]=f;
        cout<<"old_rand[0]="<<old_rand[0]<<endl;
        cin>>j;
        for(i=1;i<55;i++)
                old_rand[i]=old_rand[i-1]+f;
//      advance_rand();
}

void advance_rand(){
        int i;
        for(i=0;i<24;i++){
                new_rand=old_rand[i]-old_rand[31+i];
                if(new_rand<0.0)
                        new_rand++;
                old_rand[i]=new_rand;
        }
        for(i=24;i<55;i++){
                new_rand=old_rand[i]-old_rand[i-24];
                if(new_rand<0.0)
                        new_rand++;
                old_rand[i]=new_rand;
        }
}


double f_rand(){
        init++;
        if(init>54)
                init=1;
        advance_rand();
        return old_rand[init];
}


int i_rand(int u_limit,int l_limit){
        int i;
        if(l_limit>=u_limit)
                i=l_limit;
        else
                i=int(f_rand()*(u_limit-l_limit+1)) + l_limit;
        if(i>u_limit)
                i=u_limit;
```