AUTOMATIC ACADEMIC TIME TABLE GENERATOR
USING
GENETIC ALGORITHM

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

B.TECH
IN
INFORMATION TECHNOLOGY

Submitted By:
KULDEEP SINGH (09060103038)
PRASHANT RUWALI (09060103054)
GAURAV TIWARI (09060106013)
VARUN KUMAR (09060108111)

Under the Guidance of

Mr. Kaushik R Roy (Project Guide)
(CSE DEPARTMENT)

DEPARTMENT OF INFORMATION TECHNOLOGY
COLLEGE OF ENGINEERING ROORKEE (INDIA)
Affiliated to Uttarakhand Technical University

CERTIFICATE

We hereby declare that the work which is being presented in this dissertation which is titled as **'Automatic Timetable Generator Using Genetic Algorithm'** in the partial fulfillment of the award of the degree of BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY, submitted in the Department of **INFORMATON TECHNOLOGY, COLLEGE OF ENGINEERING ROORKEE** is an authentic record of our work carried out during a period of August 2012 to May 2013 under the guidance of **Mr. KAUSHIK R ROY (Asst.Professor**) Department of COMPUTER SCIENCE ENGINEERING, COLLEGE OF ENGINEERING ROORKEE, ROORKEE. This project report is a submission as a part of the final evaluation of B.Tech Final Year Project (PIT-852) Course.

The matter embodied in this dissertation has not been submitted by me for award of any other degree.

SUBMITTED BY:

Kuldeep Singh
(09060103038)

Gaurav Tiwari
(09060106013)

Prashant Ruwali
(09060103054)

Varun Kumar
(09060108111)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Dr.Anurag Raii	**Mr. Kaushik R Roy**	**Mr.Taresh Singh**
(Head of Department)	**(Assistant Professor)**	**(Assistant Professor)**
IT- Department	**CSE Department**	**IT Department**
	(Project Guide)	**(Project Coordinator)**

External Examiner

DATE:

ACKNOWLEDGEMENT

We take this opportunity to acknowledge our sincere gratitude to our project guide **Mr. Kaushik R Roy** for his valuable guidance and directions that helped us in making this project. We are grateful for his valuable help in implementing and testing the software and providing guidance at crucial juncture.

We are also thankful to our project coordinator **Mr. Taresh Singh** for kind and sincere efforts. We are equally thankful to staff members, both teaching and non-teaching, for their extended support and advice. Without their technical expertise it would have been difficult to complete the project.

We also sincerely admit that our work would have not crystallized into fruits of reality without the firm cooperation and encouragement of the coordinator and all our colleagues.

Lastly, in spite of meticulous care that has been taken in preparing this project report, some errors might have crept in, but we all are aware of the fact "TO ERR IS HUMAN".

KULDEEP SINGH

PRASHANT RUWALI

GAURAV TIWARI

VARUN KUMAR

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Planning timetables is one of the most complex and error-prone applications. There are still serious problems like generation of high cost time tables are occurring while scheduling and these problems are repeating frequently. Therefore there is a great requirement for an application distributing the course evenly and without collisions.

The Time Table problem can be categorized into various types (Ex: - rail timetable, examination timetable, etc) by considering different specific constraints, processes, resources. The particular instance chosen for implementation is the academic timetable.

The academic timetable problem is to organize subjects, teachers, rooms and periods in order to satisfy the set of constraints. In most of the cases faculty, subject tuple is predefined. So, the problem comes down to allocation of Faculty-Subject tuple to a specific location at a specific time following various constraints.

The scheduling problems are essentially the problem that deals with effective distribution of resources. During the scheduling process many constraints have to be considered. Resources are usually limited and no two tasks should occupy one particular resource at the same time.

The college lecture-timetabling problem asks us to find some time slots and classrooms which satisfy the constraints imposed on offered courses, lecturers, classrooms and so on.

Since the problem is a combinatorial optimization problem belonging to NP-hard class, the computation time for timetabling tends to grow exponentially as the number of variables increase. There have been a number of approaches made in the past decades to the problem of constructing timetables for colleges and schools.

Timetabling problems may be solved by different methods inherited from operations research such as graph coloring and mathematical programming, from local search procedures such as simulated annealing, from genetic algorithms or from backtracking-based constraint satisfaction manipulation.

In our project, timetabling problem is formulated as a constraint satisfaction problem and we proposed a practical timetabling algorithm which is capable of taking care of both strong and weak constraints and finding variables instantiation.

Automatic Academic Time Table Generator helps in dealing with difficulties arising while scheduling lectures of a class keeping in view all the required constraints within the domain of College Of Engineering Roorkee. The results are obtained in very less time compared to manual generation of time table.

We have proposed Genetic Algorithm for solving the time table scheduling problem. This approach well suited to time table problem since there exists an easy way to access a good

time table, but not a well structured technique for constructing it. So a population of time table is generated to make an optimal time table through genetic algorithms.

1.2 SCOPE OF PROJECT

Automatic Academic Time Table Generator generates the time table for each class within the CS/IT Department of College of Engineering Roorkee. The application creates 16 Timetables each for different sections of CS/ IT batches from 1st to 4th year. The time tables keep in view the lectures taught by different faculties, availability of resources such as lecture rooms and labs. The number of rooms for lectures and labs are considered in accordance with College of Engineering Roorkee

This application aims at providing best possible resource utilization while respecting all the constraints and generating time table in lesser time compared to manual generation.

1.3 ABSTRACT

The manual system of preparing time table in colleges with large number of students is very time consuming and usually ends up with various classes clashing either at same room or with same teachers having more than one class at a time. These are just due to common human errors which are very difficult to prevent in processes such as these. To overcome these problems people usually takes the previous years' timetable and modifies it but still it is a tedious job to incorporate changes.

To overcome all these problems we propose to make an automated system. The system will take various inputs like details of students, subjects and class rooms and teachers available, depending upon these inputs it will generate a possible time table, making optimal utilization of all resources in a way that will best suit any of constraints or college rules of CS/IT Department of College of Engineering Roorkee.

List of subjects may include electives as well as core subjects. The labs, lecture and tutorial room are considered as different resources. The application takes as input from the user the name of faculties the subject they are teaching and the batch to which these subjects are taught while outputs the scheduled time table of week which has 6 days with Saturday as half day and a lunch every day.

Efforts have been made to minimize possibility of classes during the last slot i.e 4-5, maximize the chances of lectures during morning shift and tutorial and labs during evening shifts.

The programming language used is C++ and compiler used is Devcpp. The stored and retrieved from normal text file created in MS Notepad.

1.4 TECHNOLOGY USED

- **C++ PROGRAMMING LANGUAGE**

C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises both high-level and low-level language features. Developed by Bjarne Stroustrup starting in 1979 at Bell Labs, C++ was originally named C with Classes, adding object oriented features, such as classes, and other enhancements to the C programming language. The language was renamed C++ in 1983, as a pun involving the increment operator.

C++ is one of the most popular programming languages and is implemented on a wide variety of hardware and operating system platforms. As an efficient compiler to native code, its application domains include systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games.

Several groups provide both free and proprietary C++ compiler software, including the GNU Project, Microsoft, Intel and Embarcadero Technologies. C++ has greatly influenced many other popular programming languages, most notably and Java. Other successful languages such as Objective-C use a very different syntax and approach to adding classes to C.

C++ is also used for hardware design, where the design is initially described in C++, then analyzed, architecturally constrained, and scheduled to create a register-transfer level hardware description language via high-level synthesis.

- **DEV C++**

Dev-C++ is a free integrated development environment (IDE) distributed under the GNU General Public License for programming in C and C++. MinGW, a free compiler, is bundled with it. The IDE is written in Delphi.

The project is hosted by Source Forge. Dev-C++ was originally developed by programmer Colin Laplace. Dev-C++ runs exclusively on Microsoft Windows.

Bloodshed Dev-C++ is a full-featured Integrated Development Environment (IDE) for the C and C++ programming languages. It uses the MinGWor TDM-GCC 64bit port of the GCC as its compiler. Dev-C++ can also be used in combination with Cygwin or any other GCC-based compiler.[1]

One additional aspect of Dev-C++ is its use of DevPaks, packaged extensions on the programming environment with additional libraries, templates, and utilities. DevPaks often contain, but are not limited to, GUI utilities, including popular toolkits such as GTK+, wxWidgets, and FLTK. Other DevPaks include libraries for more advanced function use.

Dev-C++ is generally considered a Windows-only program, but there are attempts to create a Linux version: header files and path delimiters are switchable between platforms.

CHAPTER 2

TIMETABLE SCHEDULING

A timetable construction is an NP-Hard scheduling problem. It is not a standard job-shop problem because of the additional classroom allocation. It is large and highly constrained, but above all the problem differs greatly for different colleges and educational institutions. It is difficult to write a universal program, suitable for all imaginable timetabling problems. Although manual construction of timetables is time-consuming, it is still widespread, because of the lack of appropriate computer programs.

2.2 TYPES OF TIMETABLING PROBLEMS

There exist many different timetabling problems such as :

- University Timetabling
- Examination Timetabling
- School Timetabling
- Sports Timetabling
- Employee Timetabling

2.3 APPROACHES

Furthermore, there exist many problem solving methods, which usually use
the concepts of standard optimization algorithms such as Backtracking, Evolutionary Algorithms or Constraint Logic Programming.

In recent years two main approaches seem to have been successful.

- The first approach is based on Local Search Procedures
- The second approach is based on Constraint Programming (CP)

2.3.1 THE LOCAL SEARCH PROCEDURE

The local search procedures such as Simulated Annealing, Tabu Search and Genetic Algorithms. These methods express constraints as some cost functions, which are minimized by a Heuristic Search of better solutions in a neighborhood of some initial feasible solution. Their greatest disadvantages are:

1. The difficulty of taking into account hard constraints.
2. The need to determine their parameters through experimentation.

Although they are good for optimizing the initial feasible solution, they have problems with finding it.

2.3.2 THE CONSTRAINT PROGRAMMING (CP)

Its main advantage is declaratively: a straightforward statement of the constraints serves as part of the program. This makes the program easy to modify, which is crucial in timetabling problems. The constraints are handled through a system of constraint propagation, which reduces domains of variables, coupled with backtracking search. In modern CP languages, both features do not need to be programmed explicitly. The main disadvantages of this approach are :

1. The difficulties with expressing soft constraints.

2. The possible problems with improving the initial feasible solution, which as a rule may be determined without difficulties.

The ability to express complex constraints in a simple, declarative way is crucial for introducing the requirements of the colleges and university timetabling problem into the program and is crucial for their successful solution, a custom-tailored distribution strategy is able to introduce soft constraints during a search, leading quickly to a ”Good” timetable, incorporation of local search into CP gives the ability to optimize effectively the timetable.

2.4 THE GENERAL VIEW

As mentioned above, many types of timetabling problems exist. But all these problems have several properties in common. One of these similarities is that certain entities have to be scheduled. e.g., the college timetabling problem has several entities such as students, lecturers, courses, practical, tutorials, classes and labs. All these entities have properties e.g. classes are linked to the course and the students of this class are taught. Constraints assignments usually cannot be done arbitrarily, but many constraints have to be considered.

We distinguish two different types, namely hard and soft constraints. A solution is feasible if no hard constraints are violated. A feasible solution is better than another if fewer soft constraints are violated. A timetabling algorithm can use different strategies to get a solution without violations of hard constraints. Violations can either be avoided from the outset or penalized to lead the algorithm towards better solutions and introduce repair mechanisms.

2.4.1 CONSTRAINTS

There are various constraints to be satisfied at the time to instantiate variables about time slots and classrooms. The constraints can be categorized into Hard and Soft constraints.

2.4.1.1 HARD CONSTRAINTS

A timetable which breaks a hard constraint is not a feasible solution, and must be repaired or rejected by the timetabling algorithm. Hard constraints include "First Order Conflicts".

Various Hard Constraints in our time table scheduling problem are:

1. A classroom is not assigned to more than one lecture at the same time.
2. A lecturer cannot teach more than one class at the same time.
3. Courses for the same year session students of a department cannot take place at the same time.
4. The classroom for a course should have enough capacity to take students registered in the course.
5. The classroom should be well equipped with required facilities for the classes.
6. The number of lectures taken by a faculty in a day should not be more than four.
7. The number of lectures of students in a batch should not exceed five.
8. Their shouldn't be more than one lecture of a subject on a given day.
9. There should be a compulsory lunch within the time slot of either 12 to 1 or 1 to 2 for each batch and each faculty.
10. Saturday is considered half working day so no allotment of lectures after 1 PM.

2.4.1.2 SOFT CONSTRAINTS

Soft constraints are less important than hard constraints, and it is usually impossible to avoid breaking at least some of them. Whichever timetabling method is applied, timetables are usually rated by a penalty function, which calculates the extent to which a timetable has violated its soft constraints. Some soft constraints are more important than others, and this is often specified with a priority value.

1. The lectures should preferably be in the morning shift.

2. The tutorial and labs should preferably be in afternoon shift.

3. The morning time slots i.e. 9:00 AM to 1:00 PM should not remain empty.

4. If possible minimize the chances for a class to occur during 4:00 PM to 5:00 PM slot.

It is desirable for timetables to satisfy all hard and soft constraints. However, it is usually difficult to meet all these constraints. Any hard constraint must not be violated in any case, but some soft constraints can be sacrificed to find feasible timetables.
The timetabling process is made more difficult by the fact that so many people are affected by its outcome.

CHAPTER 3

METHODOLOGY

3.1 REVIEW OF LITERATURE

This chapter provides an analysis of the automated timetable literature broadly organized by algorithmic technique. It begins with a presentation of the major timetable solution generation algorithms that have persisted in the literature. A detailed examination of the academic literature is provided within the context of these fundamental solution generation algorithms. An analysis of the literature, grouped by the solution generation technique used, is then presented.

3.1.1 LINEAR PROGRAMMING/INTEGER PROGRAMMING

The Linear and Integer Programming techniques, the first applied to timetabling were developed from the broader area of mathematical programming. Mathematical programming is applicable to the class of problems characterised by a large number of variables that intersect within boundaries imposed by a set of restraining conditions. The word "programming" means planning in this context and is related to the type of application. This scheme of programming was developed during World War II in connection with finding optimal strategies for conducting the war effort and used afterwards in the fields of industry, commerce and government services.

Linear Programming (LP) is that subset of mathematical programming concerned with the efficient allocation of limited resources to known activities with the objective of meeting a desired goal such as maximising profits or minimising costs.

Integer Programming (IP) deals with the solution of mathematical programming problems in which some or all of the variables can assume non-negative integer values only. Although LP methods are very valuable in formulating and solving problems related to the efficient use of limited resources they are not restricted to only these problems. Linear programming problems are generally acknowledged to be efficiently solved by just three methods, namely the graphical method, the simplest method, and the transportation method. The construction of a linear programming model involves three successive problem-solving steps.

1. The first step identifies the unknown or independent decision variables.
2. Step two requires the identification of the constraints and the formulation of these constraints as linear equations.
3. Finally, in step three, the objective function is identified and written as a linear function of the decision variables.

3.1.2 EVOLUTIONARY AND GENETIC ALGORITHMS

Evolutionary Algorithms (EAs) are a class of direct, probabilistic search and optimization algorithms gleaned from the model of organic evolution. A Genetic Algorithm (GA) is a type of EA and is regarded as being the most widely known EA in recent times.

A GA differs from other search techniques in the following ways:

- GAs optimises the trade-off between exploring new points in the search space and exploiting the information discovered thus far.

- GAs has the property of implicit parallelism. Implicit parallelism means that the GAs effects are equivalent to an extensive search of hyper planes of the given space, without directly testing all hyper plane values. Each schema denotes a hyper plane.

- GAs is randomised algorithms, in that they use operators whose results are governed by probability. The results for such operations are based on the value of a random number. This means GAs use probabilistic transition rules, not deterministic rules.

- GAs operates on several solutions simultaneously, gathering information from current search points to a direct subsequent search. Their ability to maintain multiple solutions concurrently makes them less susceptible to the convergence problem of local maxima and noise.

- GAs work with a coding of the parameter set, not the parameters themselves.

- GAs search from a population of points, not a single point.

- GAs use payoff (objective function) information, not derivatives or other auxiliary knowledge.

It is demonstrated that the literature is currently converging on the use of constraint based solution algorithms and implementations. It is also noted that the next most commonly reported implementation involves the use of hybrid algorithms.

3.2 GENETIC ALGORITHMS

Genetic algorithms are an approach to optimization and learning loosely based on principles of biological evolution. They use a population of possible solution, which are concurrently subject to modifications aimed at the determination of the optimal solution. In the GA every possible solution is represented by a “digital individual” and after the generation of an initial

set set of feasible solution (a population), individuals are randomly mated allowing the recombination of genetic material.

The resulting individuals can then be mutated with a specific mutation probability. The new population so obtained undergoes a process of natural selection which favours the survival of the fittest individuals (the best solutions), and provides and provides the basis for the new evolutionary cycle. In case of timetabling problem, the consistence of individual's genes is extremely important. The consistence means that each type of information has to be presented just once in the genotype so, instead of using destructive operators, just the mixing operators are usable.

Many approaches exist to solve the scheduling problem using genetic algorithms. All of these differ in the way how the basic operators of genetic algorithms namely, selection, cross over and mutation, are applied on a population. This project tries to find an efficient approach depending upon the requirements of the college and implement it to generate the time table schedule.

3.2.1 GENETIC PROGRAMMING

This is an offshoot of genetic algorithms in which the computer structures which undergo adaptation are themselves computer programs. Specialized genetic operators are used which generalize recombination and mutation for the tree structure computer undergoing adaptation.

Genetic algorithms simulate the evolutionary process and its origin lies more in biology rather than Computer Science, GA can be defined as follows:

"Genetic algorithms are search algorithms based on the natural selection and natural genetics"

The genetic algorithm is a highly parallel mathematical algorithm that transfers a set of individual mathematical objects, each with an associated fitness value into a new population using operations patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally occurring genetic operations.

To make things more clear let us take rabbit as an example. At any given time there is a population of rabbits. Some of them are faster and smarter than others. These faster rabbits are less likely to be eaten by foxes and therefore more of them survive. Of course some of them dumber and slower rabbits also survive because they are lucky.

Some slow rabbits breed with fast rabbits, some fast with fast, some smart rabbits with dumb rabbits and so on. And on an average when a fast and smart rabbit will breed with another fast rabbit, the resulting rabbit would be faster and smarter than the parents. This phenomenon of the survival of the fittest is used to make a solution for any problem through Genetic algorithm.

The primary parameters involved in genetic algorithms processes are as follows:

1. **Population:** The algorithm starts with a set of solutions represented by chromosomes and is called population. Solution from one population is used to form new

population. This is motivated by a fact that a new population will be better than the old one.

2. **Population Size:** Population size indicates that how many chromosomes are there in population or in one generation. If there are small no. of chromosomes so there are few possibilities to perform crossover. But if there are large no. of chromosomes then genetic algorithm slows down, and it is not practical to increase the population size after some limits.
3. **Cross over probability:** Crossover probability indicates that how often the cross over will perform. If there are no crossovers ten offsprings are exact copies of their parents.
4. **Mutation Probability:** Mutation Probability indicates that how often will the parts of chromosomes be mutated. If there is no mutation, offspring is taken after the crossover without any change. If mutation is performed, he parts of the chromosomes will be changed.
5. **Fitness:** The fitness value of chromosome indicates the probabilities of selection of chromosomes. The chromosomes having more fitness produce more offspring. Here the offspring indicates the child chromosome.
6. **Crossover:** Cross over is a recombination technique which is used to form new off springs by getting copies of high individuals. In most cross over two strings are taken from the mating pool at random and some portion of the string is exchanged in between. A cross over point usually selected randomly is used to determine the point within the string for exchanging information from selected individual.
7. **Mutation:** Mutation is the process of randomly changing encoded information for a newly created population individual. Mutation makes a random small change to one or more of the solutions. The inclusion of mutation introduced some probability.
8. **Chromosome:** All living organisms consist of cells. In each cell there is some set of chromosomes. Chromosomes are strings of DNA and serves as a model for whole organisms. A chromosome consists of genes, block of DNA. Each gene encodes a particular protein. Basically it can be said that each gene encodes a trait e.g. Colour of eyes, height, colour of skin etc.
9. **Reproduction:** During reproduction first occur recombination or cross overs. Gens from parents' forms in the similar way the whole new chromosomes, the new created off spring can then are mutated. Mutation means, the elements of DNA are a bit changed.

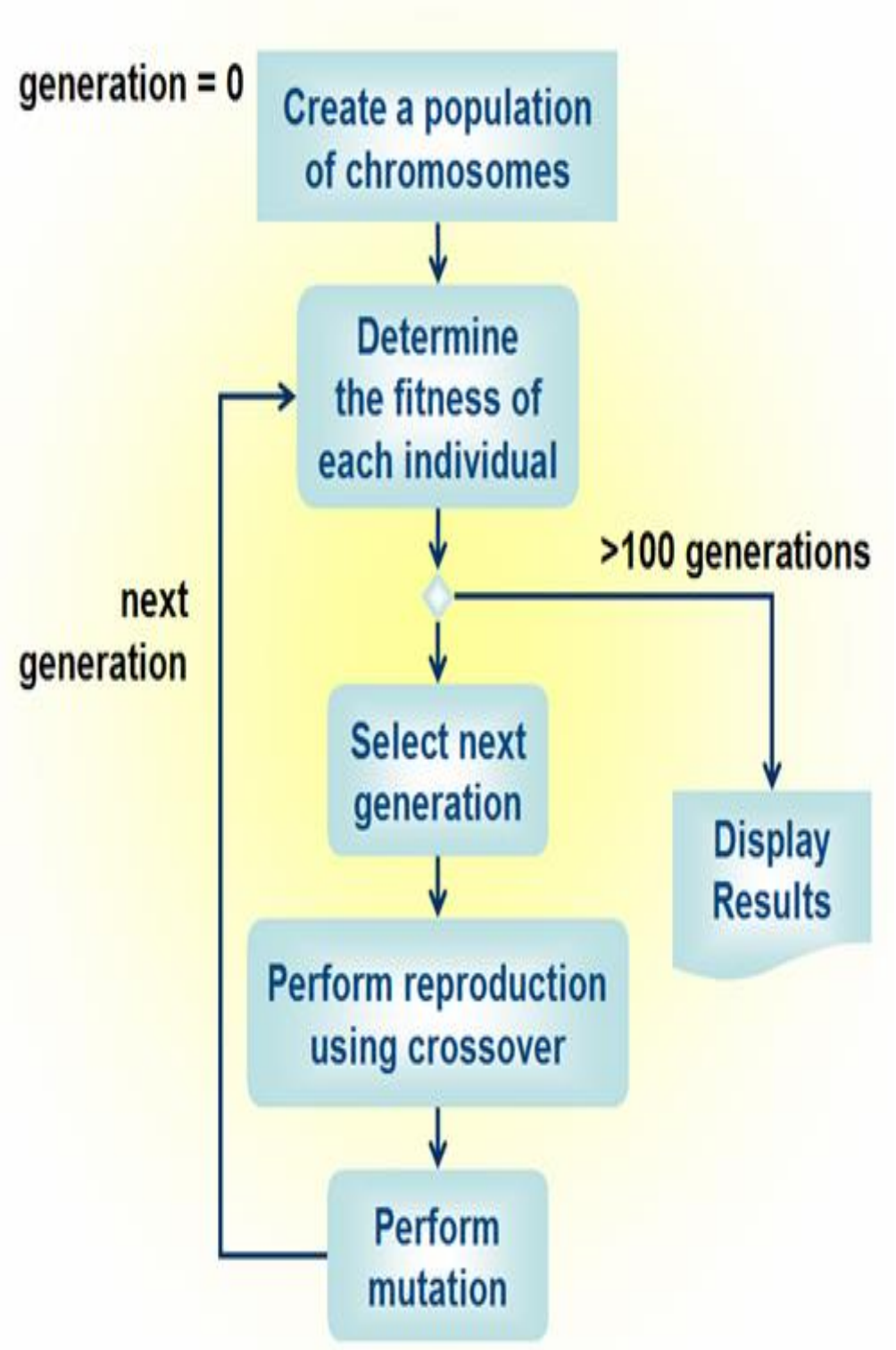

Fig : 3.1 : Diagram representing the working of genetic algorithm in our problem context

CHAPTER 4

DATA REPRESENTATION

4.1.1 DATA SET REPRESENTATION

- A list of faculties (F1,F2,F3……Fn)
- A list of Batches (B1,B2,B3……Bn)
- A list of Subjects (S1,S2,S3…….Sn)

Each of the data sets is represented in the form of linked list data structure. As soon as the input is taken from the text file we create a linked list of the faculties, subjects and batches.

The linked list has three cells with two data part representing name and code assigned whereas linked part holds the address of the next subsequent node in the list

The linked representation of faculty is:

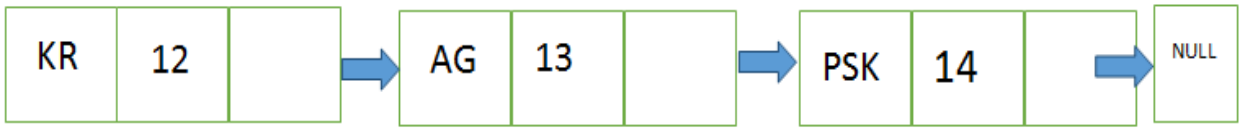

Fig:4.1

The linked representation of subject is:

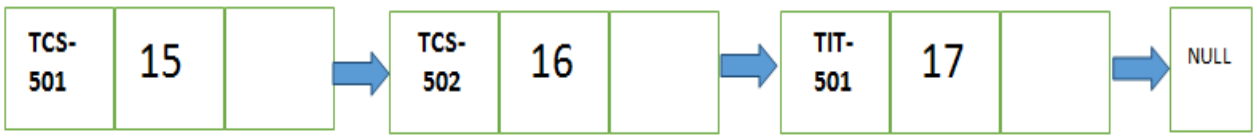

Fig: 4.2

The linked representation of batch is:

Fig : 4.3

4.1.2 DATA STRUCTURE REPRESENTATION OF ACTIVITY

The data structure prepared for each of the faculty, subject and batch are given as an input to the genetic algorithm.

These linked list are used to create a hybrid linked list which is used to create the activity. Activity is the smallest fundamental unit in the genetic algorithm. These activity later on are helpful for the creation of chromosome and to perform the mutation operation.

A diagram representing the modified linked list forming an activity can be shown in the figure as follows:

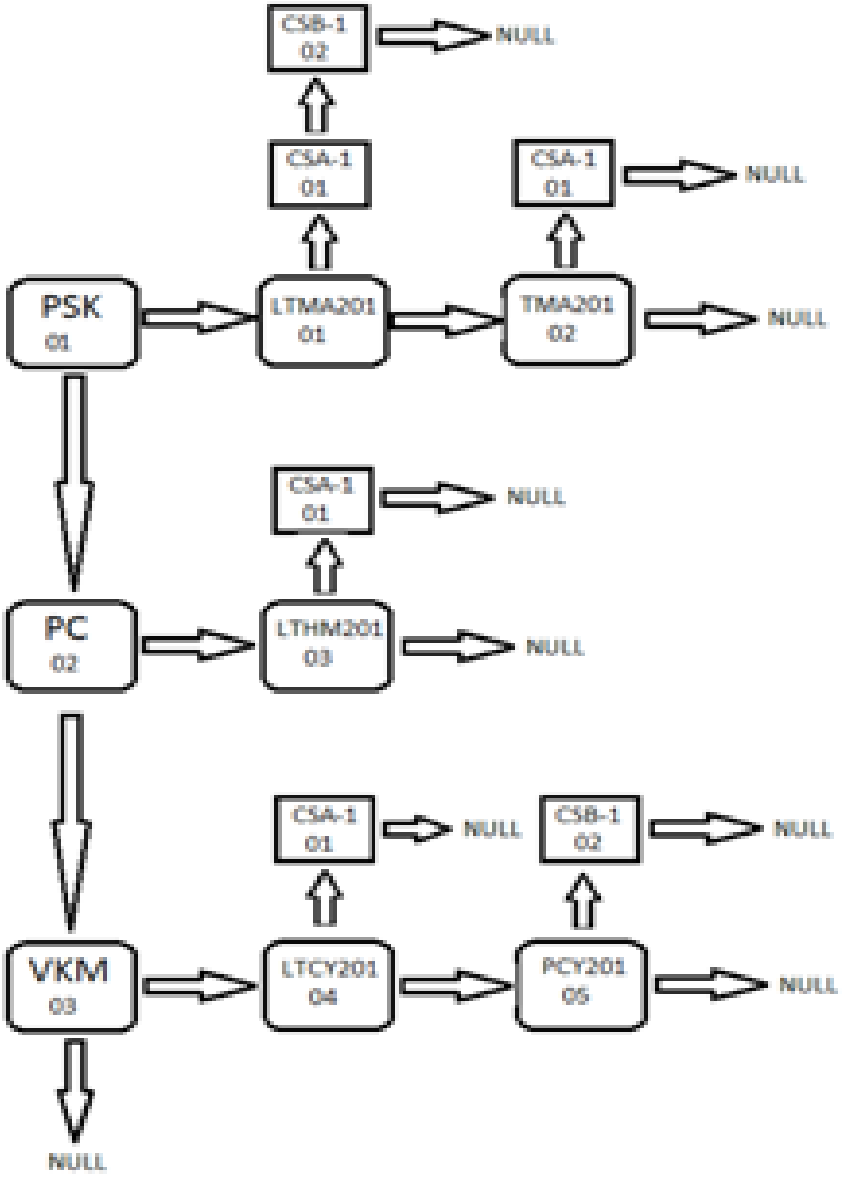

Fig: 4.4

An activity basically represents a relationship in between the faculties, subjects and classroom. Each of the faculty subject and batches are been provided with a particular code such as

- Each of the faculties provided with a code ranging from 01 to 99.
- Each of the subjects provided with a code ranging from 01 to 99.
- Each of the 16 classes provided with a code ranging from 01 to 99.

It means that time tabling problem can schedule maximum 99 faculties teaching maximum 99 subjects to at max 99 batches which is more than enough to schedule all the time table in a particular department of college.

The code for the activity is a 6 digit number which can be represented as 341207.
Here, 34 would represent the code assigned to the particular faculty.
12 represent the code assigned to the particular subject
07 represent the code assigned to the particular batch.

4.2 CODING ACTIVITY

The code assigned initially in the linked list of faculty subject and batch are used for the creation of activity by using the following formula.

act = faculty->code*10000 + subject->code*100 +batch->code;

4.3 DECODING ACTIVITY

1. In order to access the particular batch from the activity

Where, act = 123406

We would perform the operation t=123406%100

This operation will give the value of t=6 thus we would access the batch with the code value 6.

2. Similarly to access the particular subject encoded within the activity act

We would perform operation t=123406/100
Thus, t=1234
Now, t=t%100 will give t=34

This operation will give the value of t=34 thus we would access the subject with the code value 34.

3. Similarly to access the particular faculty encoded within the activity act

We would perform operation t=123406/10000
Thus, t=12

This operation will give the value of t=12 thus we would access the faculty with the code value 12.

4.4 ENCODING OF CHROMOSOME

The greatest power of the genetic algorithm lies in the formation of the chromosome and to perform the mutation and cross over operation amongst them. This operation leads to the development of newer chromosome which are much better, satisfying larger constraints and thus moving more towards the optimized results.

Here, a 3 dimensional data structure is taken by us as the chromosomes and the activities are allocated to this 3 dimensional data structure based on the soft and hard constraints satisfied by them.

Different chromosomes taken are as follows

1. fact_t [99][6][8] = It represents faculty chromosome indexing [99] represents number of faculties, [6] represents different working days in a week, and [8] represents the different slots in a day.

2. room_t [11][6][8] = It represents the room in which the lecture is to be organized. Indexing [11] represents the total number of rooms available for the tutorials and lectures, [6] and [8] represents days and slots in particular day similar to the faculty chromosome.

3. batch_t [16][6][9] = It represents batch chromosome with [16] representing each of the 16 batches in CS/IT department of the college, whereas [6] & [8] are similar to faculty and room chromosomes.

4. lroom_t [6][6][8] = It represents rooms available for lab as labs cannot be scheduled in lecture and tutorial rooms. Thus, separate rooms have to be allocated for labs which are [6] in our college scenario whereas [6] & [8] are same as above.

The chromosomes are allocated with the activity randomly. A random function generator is used to generate a random time slot on a random day within a chromosome. Once the activity is allocated within a given time slot it is checked whether they satisfy all the hard constraints imposed by the algorithm if any one of the hard constraint is violated that generation is dropped. This operation is termed as mutation.

Thus, mutation helps to produce better chromosomes making them fit for the further computation once these activities are allocated and chromosomes are prepared these chromosomes are checked for different soft constraints. The soft constraint evaluation helps to provide a fitness value. A fitness value helps to identify the best possible outcome satisfying all the hard constraints and maximum possible soft constraints.

Greater the fitness value of a chromosome better the time table scheduling has been performed in terms constraints context.

4.5 FITNESS EVALUATION

Fitness is one of the most important parameter for the development of the time table. The fitness describes how much suitable a time table is within the college scenario.

We have certain parameters on the basis of which the fitness of the time table is calculated. Fitness basically represents the satisfaction of the maximum soft constraints. The greater the soft constrains are satisfied the greater would be the fitness and thereby greater will be the feasibility of that time table.

Initially the value of the fitness variable is equal to 0.

The various parameters used for the evaluation of the fitness are:

1. If lecture is in the morning shift then fitness = fitness+10.
2. If labs and tutorials in the afternoon shift then fitness = fitness+5.
3. If lecture in the 2 to 3 shift then fitness = fitness-1.
4. If lecture in the 3 to 4 shift then fitness = fitness-2.
5. If lecture in the 4 to 5 shift then fitness = fitness-2.
6. If labs and tutorial in the 2 to 4 shift then fitness = fitness+3
7. If labs and tutorial in the 4 to 5 shift then fitness = fitness-1.
8. If labs and tutorial in the morning shift then fitness = fitness-10.
9. If morning time slots 9 to 11 are empty then fitness = fitness-10.
10. If time slot 2 to 3 is empty then fitness = fitness-3.

The total fitness associated with the time table is computed by taking into account all these constraints. Each of the time table found worthy of selection are compared on the basis of their fitness value. The time table with the highest fitness will be displayed and is considered as the final time table.

4.6 RANDOM NUMBER GENERATION

Random number generation is an important part of the genetic algorithm as it helps to provide an initial framework to chromosome and then perform mutation over the chromosome.

The use of a random function involves passing a seed value on the basis of which the numbers are generated randomly. In our case system clock is given as the seed value to the random function. This helps us to provide a large variety of random numbers at different instances of time.

It avoid the generation of same number multiple number of times thus giving us a variety of random numbers which could be used for the generation of any of the 8 time slots and any of the six days to schedule an activity.

In order to use this methodology for generation of random number we include <windows.h> header file.

```
time_t seconds;  // time_t is a datatype which stores the value of system clock in seconds
time(&seconds);
srand((unsigned int) seconds) ; // seconds is passed as seed value to the srand function.
```

Function for the generation of random time slot and random day to place an activity.

CHAPTER 5

IMPLEMENTATION

5.1 BASIC LAYOUT

- Platform used at present: GNU C/C++.
- Compiler used: Dev C++
- Input/output format: File based. Input is taken from a text file Database.txt created in MS Notepad. The end results of the application are again redirected to another text file Timetable.txt.
- Timetablegeneration.cpp: It contains the main source code of the application.

5.1.1 HEADER FILES

- #include<iostream.h>
- #include<conio.h>
- #include<string.h>
- #include<stdlib.h>
- #include<fstream.h>
- #include<time.h>
- #include<windows.h>

5.1.2 BASIC METHODS FOR THE timetablegeneration.cpp APPLICATION

- void fac1 : it neither take any parameters nor it returns anything. It is used to take the faculty name as an input to the program.

- void sub1: similar to void fac1. It is used to take subject name as input from the text file.

- void cls1 : Same as above. It is used to take the batch name as an input from the text file.

- void listcls(char *name) : It takes the name of faculty as an input and adds the faculty into a linked list of faculties. It also assigns a code to each of the faculties present within the linked list.

- void listsub(char *name) : It takes the name of subject s input and adds the subject into a linked list of subjects. It also assigns a code to each of the subject present within the linked list.

- void listcls(char *name) : It takes the name of batch as an input and adds the batch into a linked list of batches. It also assigns a code to each of the batches present within the linked list.

- void addfac(char *name) : It takes the name of faculty as the input and is used to create a data structure with each faculty linked to the subject it is teaching. The faculty itself forms the linked list. Also it gives code to the faculty generated while creating faculty linked list.

- void addsub(char *name) : It takes the name of subject as the input and is used to create a data structure with subjects linked to the class to which it is teaching. Moreover ant another subject taught by the same faculty which is teaching current subject is also linked to the current subject. Also it gives code to the subjects generated while creating subject linked list. The data structure named as subnode.

- void addcls(char *name) :It takes the name of class as the input and is used to give code to the batch. There can be a situation in which a subject is taught by a certain faculty to multiple batches in this situation each class is linked together within its data structure. The data structure named as clsnode.

- int searchfac(fac *d,char *name) : It takes the pointer to the faculty linked list and the name of faculty as input parameter and is used to search the name of the faculty within the linked list. It returns the corresponding code of faculty. The data structure named as clsnode.

- int searchsub(sub *d,char *name) : It takes the pointer to the subject linked list and the name of subject as input parameter and is used to search the name of the subject within the linked list. It returns the corresponding code of faculty.

- int searchcls(cls *d,char *name) : It takes the pointer to the subject linked list and the name of subject as input parameter and is used to search the name of the subject within the linked list. It returns the corresponding code of faculty.

- void addact(facnode *l,subnode *m,clsnode *n) : Each of the data structure i.e. facnode, subnode, clsnode created are the passed into this function for the creation of the activity containing codes of the faculty, subject and class.

- void Allocation::create() : It is used for the creation of chromosome. Initially each of the chromosome fact_t, lroom_t, room_t, batch_t are assigned -1 value indicating empty time slots.

- int Allocation::algo(int cond) : It used for the random allocation of activities within a given time slot on a given day. cond passed as an input parameter is a flag. Here there are two objects of allocation class. If value of cond==0 we are dealing with first object otherwise another object. It returns a

- int Allocation::algo1(int t) : This part of genetic algorithms concern with the satisfaction of all the hard constraints. If any of hard constraints is violated then that generation is dropped and regeneration of time table takes place. It returns whether all the constraints are satisfied by the timetable or not.

- int Allocation::fitness() : This part of genetic algorithm is concern with the computation of fitness by checking all the soft constraints associated with the time table. It returns the fitness associated with a particular time table.

- int Allocation::print() : This method help us to print the time table of the batch with maximum fitness thus most suitable time for the department.

5.2 FLOW CHART

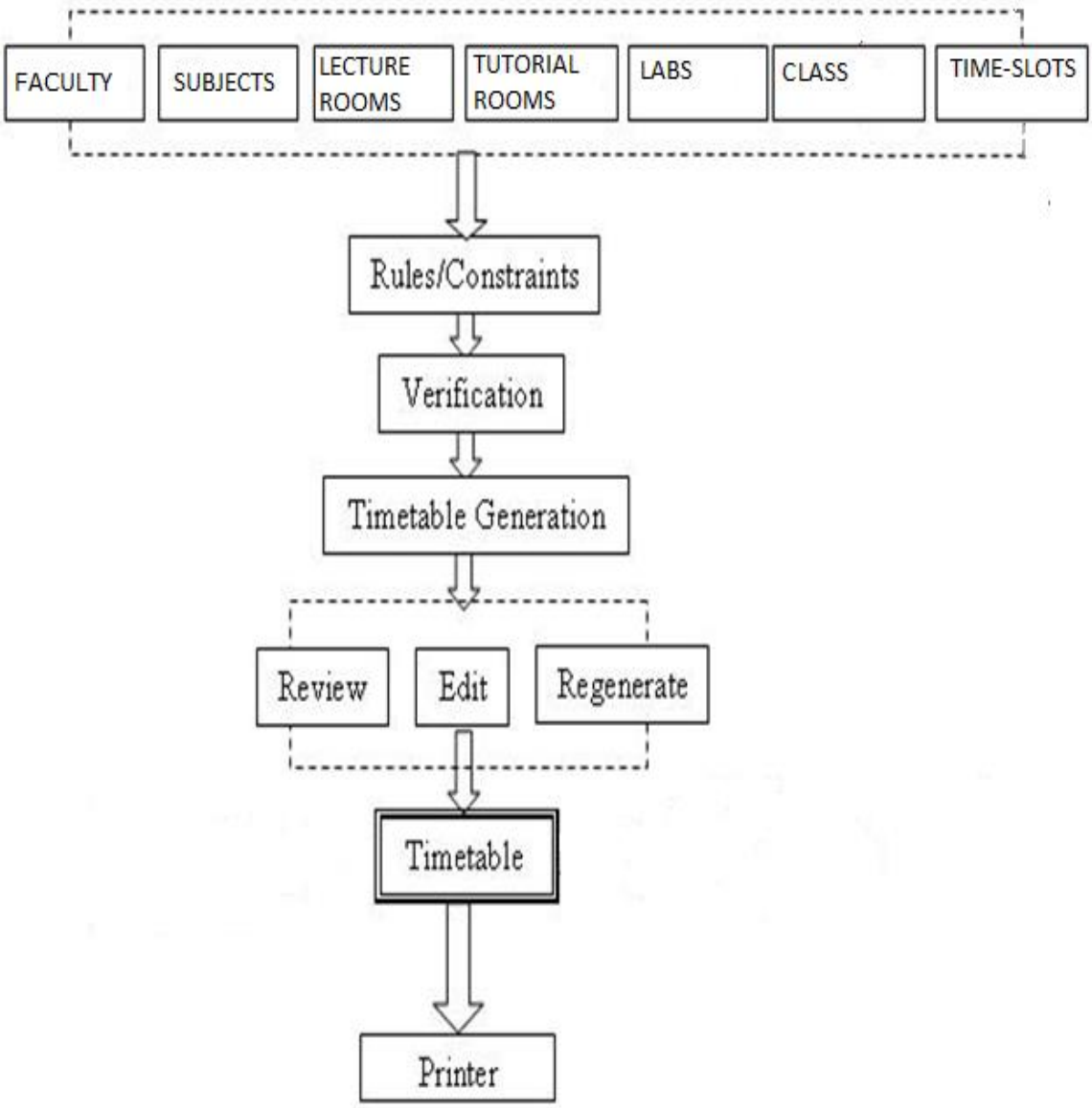

Fig: 5.1

- Get the input
- Apply the constraints
- Verify all entries
- Generate timetable
- Finalize timetable

CHAPTER 6

RESULTS

Aim of the project is to produce about 10,000 (Ten Thousand) time table and then select the time table with the maximum fitness. In order to do a comparative analysis the time tables are considered as the objects of classes. These objects are compared two at a time and the timetable with greater fitness is retained while the one with less fitness is replaced by the next timetable. This process continues for 10,000 times after which finally time table best suited for the college scenario is obtained.

The output of the timetable is fetched into a text file named dtimetable.txt. The time table for each of the 16 batches from 1st to 4th year of CS/IT department in College of Engineering Roorkee can therefore be retrieved from the text file timetable.txt.

The total time taken for the generation of time table is less than 1 minute. Thus, the time table generation problem is efficiently reduced to a considerable amount of time.

The output generated by the application is shown in the figure in next page. The diagram shows the time table for the CS-A branch. Similarly 15 more time tables are also generated for different branch and sections belonging to the CS/IT Department.

CSA-1

9 - 10	10 - 11	11 - 12	12 - 1	1 - 2	2 - 3	3 - 4	4 - 5
PSK LTMA201 302	*PC LTHM201 304	ANK PPED201 102	ANK PPED201 102	FREE	FREE	FREE	FREE
AMG TTME201 302	VKM PPCY201 105	VKM PPCY201 102	FREE	FREE	FREE	FREE	FREE
*PC LTHM201 301	AVB LTEC201 305	VKM TTCY201 302	PSK LTMA201 301	FREE	AMG LTME201 301	SRG PPEC201 102	SRG PPEC201 102
FREE	VKM LTCY201 305	AMG PPME201 102	AMG PPME201 102	FREE	AVB LTEC201 307	FREE	FREE
VKM LTCY201 301	AVB LTEC201 301	*PC LTHM201 303	AMG LTME201 305	FREE	PSK TTMA201 301	FREE	FREE
AVB TTEC201 305	AMG LTME201 302	PSK LTMA201 302	VKM LTCY201 302	FREE	FREE	FREE	FREE

Fig: 6.1

CHAPTER 7

CONCLUSION AND FUTURE WORKS

7.1 CONCLUSION

- Therefore we may conclude that Genetic Algorithms turn out to be an effective solution for optimization problems such as Time Table scheduling

7.2 FUTURE WORKS

- The time table generation problem implemented in College of Engineering Roorkee Scenario is developed for a particular department aspect. The application generates the time table associated with a certain department of college. So, in the future the automation of the timetable of the entire college can be performed. As the constraints associated with the college would be similar to that of a particular department.

- Moreover the database generation is a little bit tedious job and also the output generated is stored within the text file. Instead of using text file for input a suitable database can be taken to perform the operation.

- A suitable Graphical user Interface would also be helpful in creating a user friendly environment. The GUI would help the user to perform the timetabling generation activity a simpler task to be carried out.

- The genetic algorithms perform the selection and rejection of the timetable produced on the basis of the applied constraints, both hard and soft constraints. These constraints therefore may be specific to a certain domain instead of dealing with the problem universally. Suppose a constraint which is good for one of the department may not be worthy for the other. In these kind of situation the genetic algorithms can be provided up with the capability of choosing the constraints depanding on the domain it is working. This enhancement will result in formation of time table in accordance with the need of the organization.

CHAPTER 8

REFERENCES AND BIBLIOGRAPHY

1. "Dynamic Time Table Generation Conforming Constraints a Novel Approach" Tahir Afzal Malik, Hikmat Ullah Khan, Sajjad Sadiq.

2. "Implementation of a time table generator using visual basic.net" ISBN 1819 6608, Joseph M. Mom and Jonathan A. Enokela.

3. "Solving Job Shop Scheduling Problem Using Genetic Algorithm with Penalty Function" Volume 1, Number 2 Liang Sun, Xiaochun Cheng, Yanchun Liang.

4. "A Genetic Algorithm for Resource-Constrained Scheduling" by Matthew Bartschi Wall.

5. "The Complete Reference C++" 4th Edition by Herbert Schildt Tata Mcgraw Hill Publication.

6. Wikipedia.

CHAPTER 9

APPENDIX

9.1 GLOBAL VARIABLES AND STRUCTURES

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#include<fstream.h>
#include<time.h>
#include<windows.h>

using namespace std;

FILE *fp;               // input file pointer

char ch;

void listfac(char *);
void listsub(char *);
void listcls(char *);

void addfac(char *);
void addsub(char *);
void addcls(char *);

void allocate();

void cls1();
void sub1();
void fac1();

int no_of_act;

struct fac              // structure of faculty data structure
{
     char name[20];
     int code;
     struct fac *next;
};
```

```
struct sub                // structure of subject data structure
{
      char name[20];
      int code;
      struct sub *next;
};

struct cls                // structure of batch data structure
{
      char name[20];
      int code;
      struct cls *next;
};

struct facnode            // structure of faculty node in activity data structure
{
          int code;
          facnode *link;
   struct subnode *sub_link;
          char name[20];
};

struct subnode            // structure of subject node in activity data structure
{
          int code;
          subnode *link;
   struct clsnode *cls_link;
          char name[20];
};

struct clsnode            // structure of batch node in activity data structure
{
          int code;
          clsnode *link;
          char name[20];
};

facnode *startfac=NULL,*start=NULL;   // pointers for respective node
subnode *startsub=NULL;
clsnode *startcls=NULL;

facnode *a;
subnode *b;
clsnode *c;

clsnode n1[2000];
subnode n2[1000];
facnode n3[100];
```

[[AATTG]]

```
cls n6[1000];
sub n5[1000];
fac n4[1000];

int count1=0,count2=0,count3=0,count4=0,count6=0,count5=0;

fac *x,*x1,*q1;
sub *y,*y1,*q2;
cls *z,*z1,*q3;

static fac *start1=NULL;
static sub *start2=NULL;
static cls *start3=NULL;

int searchfac(fac *, char *);
int searchsub(sub *, char *);
int searchcls(cls *, char *);

void addact(facnode *,subnode *,clsnode *);

int arr[5000];                  // array to store activity
int i=0;

int xx1=0,xx2=0,xx3=0;

int tfitness=0;

int batch_t[16][6][9];          // array to store batch Timetable
int room_t[11][6][8];           // array to store room Timetable
int lroom_t[4][6][8];           // array to store lab Timetable
int fact_t[99][6][8];           // array to store faculty Timetable

class Allocation        // class to create instances of Timetables
{
 public:
        int faculty, subject, batch;
        int algo(int);
        int algo1(int);
        void create();
        void print();
        int fitness();
        void output();
};

Allocation allow,allow1;    // objects of class Allocation
```

9.2 CREATE DATA STRUCTURE

```
void listfac(char *name)        // function to add faculty name to linked list
{
    if(start1==NULL)
    {
      start1=&n4[count4];
      n4[count4].next=NULL;
      strcpy((start1->name),(name));
      n4[count4++].code=1;
    }
    else
    {
      x=start1;
      while(x!=NULL)
        {
                if(!strcmp((x->name),(name)))
                {
                   return;
                }
                else
                {
                   x1=x;
                   x=x->next;
                }
         }
         x1->next=&n4[count4++];
         x=x1->next;
         strcpy((x->name),(name));
         x->code=x1->code+1;
         x->next=NULL;

    }
}

void listsub(char *name)            // function to add subject name to linked list
{
    if(start2==NULL)
    {
       start2=&n5[count5++];
       start2->next=NULL;
       strcpy((start2->name),(name));
       start2->code=1;
    }
```

```
    else
    {
      y=start2;
      while(y!=NULL)
       {
                if(!strcmp((y->name),(name)))
                {
                   return;
                }
                else
                {
                   y1=y;
                   y=y->next;
                }
        }
        y1->next=&n5[count5++];
        y=y1->next;
        strcpy((y->name),(name));
        y->code=y1->code+1;
        y->next=NULL;
    }
}

void listcls(char *name)          // function to add batch name to linked list
{
    if(start3==NULL)
    {
      start3=&n6[count6];
      n6[count6].next=NULL;
      strcpy((start3->name),(name));
      n6[count6++].code=1;
    }
    else
    {
       z=start3;
       while(z!=NULL)
       {
                if(!strcmp((z->name),(name)))
                {
                   return;
                }
                else
                {
                   z1=z;
                   z=z->next;
                }
        }
        z1->next=&n6[count6++];
        z=z1->next;
        strcpy((z->name),(name));
```

```
        z->code=z1->code+1;
        z->next=NULL;
    }
}
void fac1()              // function to read faculty name from input File
{
    int i=0;
    char facname[3];
    ch=getc(fp);
    ch=getc(fp);
    while(ch!=';')
     {

       facname[i]=ch;
       ch=getc(fp);
       i++;
     }
    facname[i]=NULL;
          listfac(facname);
          addfac(facname);
          sub1();
}

void sub1()               // function to read subject name from input File
{
    int i=0;
    char subname[20];
    ch=getc(fp);
    while(ch!=',')
     {
       subname[i]=ch;
       ch=getc(fp);
       i++;
     }
    subname[i]=NULL;
          listsub(subname);
          addsub(subname);
    cls1();
}

void cls1()               // function to read batch name from input File
{
    int i=0;
    char clsname[20];
    ch=getc(fp);
    while(ch!=';'&&ch!=':'&&ch!=','&&ch!=EOF)
     {
       clsname[i]=ch;
       ch=getc(fp);
       i++;
```

```
     }
    clsname[i]=NULL;
    char mmm=clsname[i-1];
    clsname[i-2]=NULL;
    no_of_act=mmm;
    if(ch==',')
     {
              listcls(clsname);
              addcls(clsname);
              cls1();
     }
    else if(ch==';')
     {
       listcls(clsname);
       addcls(clsname);
       sub1();
     }
   else if(ch==':')
     {
       listcls(clsname);
       addcls(clsname);
       fac1();
     }
   else if(ch==EOF)
     {
       listcls(clsname);
       addcls(clsname);
     }
   else
    {
    }
}
```

9.3 ACTIVITY GENERATION FUNCTION

```
void addact(facnode *l,subnode *m,clsnode *n)        // function to generate acitivity
{
     int bbb;
     bbb=l->code*10000 + m->code*100 +n->code;
     for(int y=1;y<=no_of_act-48;y++)
     {
       arr[i++]=bbb;
     }
}

int searchfac(fac *d,char *name)            // function to search faculty name in linked list
{
   while(d!=NULL)
   {
                if(!strcmp((d->name),(name)))
                {
                  return d->code;
                }
                else
                {
                d=d->next;
                }
   }
}

int searchsub(sub *d,char *name)            // function to search subject name in linked list
{
   while(d!=NULL)
   {
                if(!strcmp((d->name),(name)))
                {
                  return d->code;
                }
                else
                {
                  d=d->next;
                }
   }
}
```

```
int searchcls(cls *d,char *name)          // function to search batch name in linked list
{
   while(d!=NULL)
   {
              if(!strcmp((d->name),(name)))
              {
                return d->code;
              }
              else
              {
               d=d->next;
              }
   }
}
void addcls(char *name)          // function to add batch node to activity data structure
{

          if(startcls==NULL)
          {
        b->cls_link=&n1[count3++];
        startcls=&n1[count3];
        c=startcls;
        strcpy((startcls->name),(name));
        n1[count3].code=searchcls(start3, name);
        n1[count3].link=NULL;
   }
   else
   {
      c=startcls;
      while((c->link)!=NULL)
      {
       c=c->link;
      }
      c->link=&n1[count3++];
      c=c->link;
      strcpy((c->name),(name));
      c->code=searchcls(start3, name);
      c->link=NULL;
   }
   addact(a,b,c);
 }

void addsub(char *name)          // function to add subject node to activity data structure
{
          if(startsub==NULL)
          {
        startsub=&n2[count2];
        a->sub_link=&n2[count2];
        b=&n2[count2];
        strcpy((b->name),(name));
```

```
        b->code=searchsub(start2, name);
        b->link=NULL;
        b->cls_link=NULL;
        count2++;
   }
   else
   {
      b=startsub;
      while(b->link!=NULL)
      {
       b=b->link;
      }
      b->link=&n2[count2];
      b=b->link;
      strcpy((b->name),(name));
      b->code=searchsub(start2, name);
      b->link=NULL;
      b->cls_link=NULL;
      count2++;
    }

}
void addfac(char *name)          // function to add faculty node to activity data structure
{
   startfac=start;
         if(startfac==NULL)
         {
      startfac=&n3[count1];
      a=&n3[count1];
      strcpy((a->name),(name));
      n3[count1].code=searchfac(start1, name);
      n3[count1].link=NULL;
      n3[count1++].sub_link=NULL;
   }
   else
   {
      a=startfac;
      while(a->link!=NULL)
      {
       a=a->link;
      }
      a->link=&n3[count1];
      a=a->link;
      strcpy((a->name),(name));
      a->code=searchfac(start1, name);
      a->link=NULL;
      a->sub_link=NULL;
      count1++;
   }
```

9.4 ALLOCATION

```
void Allocation::create()   // function to initialize timetable to null
{
    for(int i=0;i<99;i++)
    {
       for(int j=0;j<6;j++)
       {
           for (int k=0;k<8;k++)
           {
                fact_t[i][j][k]=-1;
           }
       }
    }
    for(int i=0;i<4;i++)
    {
       for(int j=0;j<6;j++)
       {
           for (int k=0;k<8;k++)
           {
                lroom_t[i][j][k]=-1;
           }
       }
    }
    for(int i=0;i<11;i++)
    {
       for(int j=0;j<6;j++)
       {
           for (int k=0;k<8;k++)
           {
               room_t[i][j][k]=-1;
           }
       }
    }
    for(int i=0;i<16;i++)
    {
        for(int j=0;j<6;j++)
        {
            for (int k=0;k<9;k++)
            {
               if(k==8)
               batch_t[i][j][k]=0;
               else
```

```
                batch_t[i][j][k]=-1;
              }
           }
      }
}

int kul;

void Allocation::print()    // function to print coded Timetable
{
    int i,j,k;
    for(i=0;i<16;i++)
    {
       for(j=0;j<6;j++)
       {
          for(k=0;k<9;k++)
          {
             cout<<batch_t[i][j][k]<<" ";
          }
          cout<<"\n";
       }
       cout<<"\n";
    }
getch();
}

int Allocation::algo(int cond)        // definition of activity selection function
{
   int ret;
   time_t seconds;
   time(&seconds);
   srand((unsigned int) seconds) ;
   do
    {
     int k;
     kul=0;
     for(k=0;k<5000;k++)
     {
       if(arr[k]!=0)
       {
         kul++;
       }
       else
       {
       break;
       }
     }
    k=rand()% kul;
    int ca=arr[k];
    int temp=ca;
```

```
    batch=ca%100;
    ca=ca/100;
    subject=ca%100;
    ca=ca/100;
    faculty=ca;
    if(cond==0)
    {
      ret=allow.algo1(k);
    }
    else
    {
      ret=allow1.algo1(k);
    }
    if(ret==0)
    {
      break;
    }
   }
    while(arr[0]!=0);
     if(ret==0)
     {
      return 0;
     }
    else
    {
      return 1;
    }
}

int Allocation::algo1(int t)          // definition of slot selection function
{
    int temp=1;
    time_t seconds;
    time(&seconds);
    srand((unsigned int) seconds);
    while(temp<1000)
    {
                q2=start2;
                for(int g=0;g<subject-1;g++)
                {
                      q2=q2->next;
                }
      if(q2->name[0]=='L')
      {
      if(temp<500)
      {
         temp++;
      int l=rand()%24;
      int day;
      int slot;
```

```
        if(l%4!=0)
         {
           day=(l/4)+1 ;
         }
        else
         {
           if(l!=0)
            {
              day=l/4;
              slot=4;
            }
            else
             {
               day=1;
               slot=1;
             }
         }
         if(l%4!=0)
         {
           slot=(l%4);
         }
         if(fact_t[faculty-1][day-1][slot-1]==-1)
         {

         }
         else
         {
           continue;
         }
         if(batch_t[batch-1][day-1][slot-1]==-1)
         {
         }
         else
         {
          continue;
         }
         if(slot==4)
         {
          if(batch_t[batch-1][day-1][4]==-1)
           {
             batch_t[batch-1][day-1][4]=0;
           }
           else
           {
             continue;
           }
         }
           if(slot==5)
           {
             if(batch_t[batch-1][day-1][3]==-1)
```

```
            {
              batch_t[batch-1][day-1][3]=0;
            }
           else
           {
              continue;
           }
         }
         q2=start2;
    while(q2->code!=subject)
    {
      q2=q2->next;
    }
    int room=-1;
    if(q2->name[0]!='P')
    {
      if(batch_t[batch-1][day-1][8]==5)
        continue;
      int l=0;
      for(int k=0;k<8;k++)
      {
        if(((batch_t[batch-1][day-1][k]/100)%100)==subject)
        {
          l=1;
          break;
        }
      }
      if(l==1)
       continue;
      room=-1;
      for(int temp3=0;temp3<11;temp3++)
      {
       if(room_t[temp3][day-1][slot-1]==-1)
       {
         room=temp3;
         break;
       }
      }
      if(room==-1)
      {
         continue;
      }
   }
   else
   {
     if((slot==4)||(batch_t[batch-1][day-1][slot]!=-1)||(fact_t[faculty-1][day-1][slot]!=-1))
     continue;
     room=-1;
     for(int temp3=0;temp3<4;temp3++)
     {
```

```
        if(lroom_t[temp3][day-1][slot-1]==-1)
          {
            room=temp3;
            break;
          }
       }
        if(room==-1)
        {
          continue;
        }
     }
     if(q2->name[0]!='P')
      {
        fact_t[faculty-1][day-1][slot-1]=arr[t];
        batch_t[batch-1][day-1][slot-1]=arr[t];
        if(q2->name[0]=='L')
        batch_t[batch-1][day-1][8]=batch_t[batch-1][day-1][8]+1;
        room_t[room][day-1][slot-1]=arr[t];
      }
     else
      {
        fact_t[faculty-1][day-1][slot-1]=arr[t];
        fact_t[faculty-1][day-1][slot]=arr[t];
        batch_t[batch-1][day-1][slot-1]=arr[t];
        batch_t[batch-1][day-1][slot]=arr[t];
        lroom_t[room][day-1][slot-1]=arr[t];
        lroom_t[room][day-1][slot]=arr[t];
      }
      int temp4=t;
      do
       {
          arr[temp4]=arr[temp4+1];
          temp4++;
       }
      while(arr[temp4]!=0);
     kul=0;

      }
      else if(temp>=500&&temp<800)
      {
          temp++;
      int l=rand()%35;
      int day;
      int slot;
      if(l%6!=0)
       {
         day=(l/6)+1 ;
       }
      else
       {
```

```
 if(l!=0)
  {
    day=l/6;
    slot=6;
  }
  else
   {
     day=1;
     slot=1;
   }
}
if(l%6!=0)
{
  slot=(l%6);
}
if(fact_t[faculty-1][day-1][slot-1]==-1)
{

}
else
{
  continue;
}
if(batch_t[batch-1][day-1][slot-1]==-1)
{
}
else
{
 continue;
}
if(slot==4)
{
 if(batch_t[batch-1][day-1][4]==-1)
  {
    batch_t[batch-1][day-1][4]=0;
  }
  else
  {
    continue;
  }
 }
  if(slot==5)
  {
    if(batch_t[batch-1][day-1][3]==-1)
     {
       batch_t[batch-1][day-1][3]=0;
     }
    else
    {
       continue;
```

```
            }
          }
          q2=start2;
      while(q2->code!=subject)
      {
        q2=q2->next;
      }
      int room=-1;
      if(q2->name[0]!='P')
      {
        if(batch_t[batch-1][day-1][8]==5)
          continue;
        int l=0;
        for(int k=0;k<8;k++)
        {
          if(((batch_t[batch-1][day-1][k]/100)%100)==subject)
          {
            l=1;
            break;
          }
        }
        if(l==1)
         continue;
        room=-1;
        for(int temp3=0;temp3<11;temp3++)
        {
          if(room_t[temp3][day-1][slot-1]==-1)
          {
            room=temp3;
            break;
          }
        }
        if(room==-1)
        {
          continue;
        }
    }
    else
    {
      if((slot==4)||(batch_t[batch-1][day-1][slot]!=-1)||(fact_t[faculty-1][day-1][slot]!=-1))
      continue;
      room=-1;
      for(int temp3=0;temp3<4;temp3++)
      {
        if(lroom_t[temp3][day-1][slot-1]==-1)
          {
            room=temp3;
            break;
          }
      }
```

```
        if(room==-1)
        {
          continue;
        }
      }
      if(q2->name[0]!='P')
       {
        fact_t[faculty-1][day-1][slot-1]=arr[t];
        batch_t[batch-1][day-1][slot-1]=arr[t];
        if(q2->name[0]=='L')
        batch_t[batch-1][day-1][8]=batch_t[batch-1][day-1][8]+1;
        room_t[room][day-1][slot-1]=arr[t];
       }
      else
       {
        fact_t[faculty-1][day-1][slot-1]=arr[t];
        fact_t[faculty-1][day-1][slot]=arr[t];
        batch_t[batch-1][day-1][slot-1]=arr[t];
        batch_t[batch-1][day-1][slot]=arr[t];
        lroom_t[room][day-1][slot-1]=arr[t];
        lroom_t[room][day-1][slot]=arr[t];
       }
      int temp4=t;
      do
       {
         arr[temp4]=arr[temp4+1];
         temp4++;
       }
      while(arr[temp4]!=0);
      kul=0;
       }
       else
       {
        temp++;
        int l=rand()%45;
        int day;
        int slot;
        if(l%8!=0)
        {
          day=(l/8)+1 ;
        }
        else
        {
         if(l!=0)
          {
            day=l/8;
            slot=8;
          }
          else
           {
```

```
            day=1;
            slot=1;
          }
        }
        if(l%8!=0)
        {
          slot=(l%8);
        }
        if(fact_t[faculty-1][day-1][slot-1]==-1)
        {

        }
       else
       {
         continue;
       }
       if(batch_t[batch-1][day-1][slot-1]==-1)
       {
       }
       else
       {
        continue;
       }
       if(slot==4)
       {
        if(batch_t[batch-1][day-1][4]==-1)
        {
          batch_t[batch-1][day-1][4]=0;
        }
        else
        {
          continue;
        }
       }
        if(slot==5)
        {
          if(batch_t[batch-1][day-1][3]==-1)
           {
             batch_t[batch-1][day-1][3]=0;
           }
          else
          {
             continue;
          }
         }
        q2=start2;
    while(q2->code!=subject)
    {
      q2=q2->next;
    }
```

```
      int room=-1;
      if(q2->name[0]!='P')
      {
        if(batch_t[batch-1][day-1][8]==5)
          continue;
        int l=0;
        for(int k=0;k<8;k++)
        {
          if(((batch_t[batch-1][day-1][k]/100)%100)==subject)
          {
            l=1;
            break;
          }
        }
        if(l==1)
         continue;
        room=-1;
        for(int temp3=0;temp3<11;temp3++)
        {
         if(room_t[temp3][day-1][slot-1]==-1)
         {
           room=temp3;
           break;
         }
        }
        if(room==-1)
        {
           continue;
        }
    }
    else
    {
      if((slot==8||slot==4)||(batch_t[batch-1][day-1][slot]!=-1)||(fact_t[faculty-1][day-
1][slot]!=-1))
       continue;
       room=-1;
       for(int temp3=0;temp3<4;temp3++)
       {
         if(lroom_t[temp3][day-1][slot-1]==-1)
           {
             room=temp3;
             break;
           }
       }
         if(room==-1)
         {
           continue;
         }
     }
     if(q2->name[0]!='P')
```

```
 {
    fact_t[faculty-1][day-1][slot-1]=arr[t];
    batch_t[batch-1][day-1][slot-1]=arr[t];
    if(q2->name[0]=='L')
    batch_t[batch-1][day-1][8]=batch_t[batch-1][day-1][8]+1;
    room_t[room][day-1][slot-1]=arr[t];
 }
else
 {
    fact_t[faculty-1][day-1][slot-1]=arr[t];
    fact_t[faculty-1][day-1][slot]=arr[t];
    batch_t[batch-1][day-1][slot-1]=arr[t];
    batch_t[batch-1][day-1][slot]=arr[t];
    lroom_t[room][day-1][slot-1]=arr[t];
    lroom_t[room][day-1][slot]=arr[t];
  }
 int temp4=t;
 do
  {
     arr[temp4]=arr[temp4+1];
     temp4++;
  }
 while(arr[temp4]!=0);
kul=0;
}
}
else
{
   if(temp<500)
  {
     temp++;
  int l=rand()%35;
  int day;
  int slot;
  if(l%6!=0)
   {
     day=(l/6)+1 ;
   }
  else
   {
     if(l!=0)
      {
        day=l/6;
        slot=6;
      }
      else
       {
         day=1;
         slot=1;
       }
```

```
    }
    if(l%6!=0)
    {
      slot=(l%6);
    }
    if(fact_t[faculty-1][day-1][slot-1]==-1)
    {

    }
    else
    {
      continue;
    }
    if(batch_t[batch-1][day-1][slot-1]==-1)
    {
    }
    else
    {
     continue;
    }
    if(slot==4)
    {
     if(batch_t[batch-1][day-1][4]==-1)
      {
        batch_t[batch-1][day-1][4]=0;
      }
      else
      {
        continue;
      }
     }
      if(slot==5)
      {
        if(batch_t[batch-1][day-1][3]==-1)
         {
           batch_t[batch-1][day-1][3]=0;
         }
        else
        {
           continue;
        }
       }
       q2=start2;
  while(q2->code!=subject)
  {
   q2=q2->next;
  }
  int room=-1;
  if(q2->name[0]!='P')
  {
```

```
        if(batch_t[batch-1][day-1][8]==5)
          continue;
        int l=0;
        for(int k=0;k<8;k++)
        {
           if(((batch_t[batch-1][day-1][k]/100)%100)==subject)
           {
              l=1;
              break;
           }
        }
        if(l==1)
         continue;
        room=-1;
        for(int temp3=0;temp3<11;temp3++)
        {
          if(room_t[temp3][day-1][slot-1]==-1)
          {
            room=temp3;
            break;
          }
        }
        if(room==-1)
        {
           continue;
        }
    }
    else
    {
      if((slot==4)||(batch_t[batch-1][day-1][slot]!=-1)||(fact_t[faculty-1][day-1][slot]!=-1))
      continue;
      room=-1;
      for(int temp3=0;temp3<4;temp3++)
      {
        if(lroom_t[temp3][day-1][slot-1]==-1)
          {
            room=temp3;
            break;
          }
      }
        if(room==-1)
        {
          continue;
        }
     }
     if(q2->name[0]!='P')
      {
        fact_t[faculty-1][day-1][slot-1]=arr[t];
        batch_t[batch-1][day-1][slot-1]=arr[t];
        if(q2->name[0]=='L')
```

```
        batch_t[batch-1][day-1][8]=batch_t[batch-1][day-1][8]+1;
        room_t[room][day-1][slot-1]=arr[t];
      }
     else
      {
        fact_t[faculty-1][day-1][slot-1]=arr[t];
        fact_t[faculty-1][day-1][slot]=arr[t];
        batch_t[batch-1][day-1][slot-1]=arr[t];
        batch_t[batch-1][day-1][slot]=arr[t];
        lroom_t[room][day-1][slot-1]=arr[t];
        lroom_t[room][day-1][slot]=arr[t];
      }
     int temp4=t;
     do
      {
         arr[temp4]=arr[temp4+1];
         temp4++;
      }
     while(arr[temp4]!=0);
    kul=0;

      }
      else if(temp>=500&&temp<800)
      {
          temp++;
     int l=rand()%40;
     int day;
     int slot;
     if(l%7!=0)
      {
        day=(l/7)+1 ;
      }
     else
      {
        if(l!=0)
         {
           day=l/7;
           slot=7;
         }
        else
         {
           day=1;
           slot=1;
         }
      }
      if(l%7!=0)
      {
        slot=(l%7);
      }
      if(fact_t[faculty-1][day-1][slot-1]==-1)
```

```
        {

        }
        else
        {
          continue;
        }
        if(batch_t[batch-1][day-1][slot-1]==-1)
        {
        }
        else
        {
         continue;
        }
        if(slot==4)
        {
         if(batch_t[batch-1][day-1][4]==-1)
          {
            batch_t[batch-1][day-1][4]=0;
          }
          else
          {
            continue;
          }
         }
          if(slot==5)
          {
            if(batch_t[batch-1][day-1][3]==-1)
             {
               batch_t[batch-1][day-1][3]=0;
             }
            else
            {
               continue;
            }
           }
          q2=start2;
      while(q2->code!=subject)
      {
        q2=q2->next;
      }
      int room=-1;
      if(q2->name[0]!='P')
      {
        if(batch_t[batch-1][day-1][8]==5)
         continue;
        int l=0;
        for(int k=0;k<8;k++)
        {
          if(((batch_t[batch-1][day-1][k]/100)%100)==subject)
```

```
            {
              l=1;
              break;
            }
          }
          if(l==1)
           continue;
          room=-1;
          for(int temp3=0;temp3<11;temp3++)
          {
            if(room_t[temp3][day-1][slot-1]==-1)
            {
              room=temp3;
              break;
            }
          }
          if(room==-1)
          {
             continue;
          }
      }
      else
      {
        if((slot==4)||(batch_t[batch-1][day-1][slot]!=-1)||(fact_t[faculty-1][day-1][slot]!=-1))
        continue;
        room=-1;
        for(int temp3=0;temp3<4;temp3++)
        {
          if(lroom_t[temp3][day-1][slot-1]==-1)
            {
              room=temp3;
              break;
            }
        }
          if(room==-1)
          {
            continue;
          }
       }
       if(q2->name[0]!='P')
        {
          fact_t[faculty-1][day-1][slot-1]=arr[t];
          batch_t[batch-1][day-1][slot-1]=arr[t];
          if(q2->name[0]=='L')
          batch_t[batch-1][day-1][8]=batch_t[batch-1][day-1][8]+1;
          room_t[room][day-1][slot-1]=arr[t];
        }
       else
        {
          fact_t[faculty-1][day-1][slot-1]=arr[t];
```

```
        fact_t[faculty-1][day-1][slot]=arr[t];
        batch_t[batch-1][day-1][slot-1]=arr[t];
        batch_t[batch-1][day-1][slot]=arr[t];
        lroom_t[room][day-1][slot-1]=arr[t];
        lroom_t[room][day-1][slot]=arr[t];
      }
     int temp4=t;
     do
      {
         arr[temp4]=arr[temp4+1];
         temp4++;
      }
     while(arr[temp4]!=0);
    kul=0;
     }
     else
     {
       temp++;
       int l=rand()%45;
       int day;
       int slot;
       if(l%8!=0)
       {
          day=(l/8)+1 ;
       }
       else
       {
         if(l!=0)
          {
            day=l/8;
            slot=8;
          }
          else
           {
             day=1;
             slot=1;
           }
       }
       if(l%8!=0)
       {
         slot=(l%8);
       }
       if(fact_t[faculty-1][day-1][slot-1]==-1)
       {

       }
       else
       {
          continue;
       }
```

```
    if(batch_t[batch-1][day-1][slot-1]==-1)
    {
    }
    else
    {
      continue;
    }
    if(slot==4)
    {
      if(batch_t[batch-1][day-1][4]==-1)
      {
        batch_t[batch-1][day-1][4]=0;
      }
      else
      {
        continue;
      }
     }
      if(slot==5)
      {
        if(batch_t[batch-1][day-1][3]==-1)
         {
           batch_t[batch-1][day-1][3]=0;
         }
        else
        {
           continue;
        }
       }
       q2=start2;
  while(q2->code!=subject)
  {
    q2=q2->next;
  }
  int room=-1;
  if(q2->name[0]!='P')
  {
    if(batch_t[batch-1][day-1][8]==5)
      continue;
    int l=0;
    for(int k=0;k<8;k++)
    {
      if(((batch_t[batch-1][day-1][k]/100)%100)==subject)
      {
        l=1;
        break;
      }
    }
    if(l==1)
     continue;
```

```
        room=-1;
        for(int temp3=0;temp3<11;temp3++)
        {
         if(room_t[temp3][day-1][slot-1]==-1)
         {
           room=temp3;
           break;
         }
        }
        if(room==-1)
        {
           continue;
        }
    }
    else
    {
      if((slot==8||slot==4)||(batch_t[batch-1][day-1][slot]!=-1)||(fact_t[faculty-1][day-
1][slot]!=-1))
       continue;
       room=-1;
       for(int temp3=0;temp3<4;temp3++)
       {
         if(lroom_t[temp3][day-1][slot-1]==-1)
           {
             room=temp3;
             break;
           }
       }
         if(room==-1)
         {
           continue;
         }
     }
     if(q2->name[0]!='P')
      {
         fact_t[faculty-1][day-1][slot-1]=arr[t];
         batch_t[batch-1][day-1][slot-1]=arr[t];
         if(q2->name[0]=='L')
         batch_t[batch-1][day-1][8]=batch_t[batch-1][day-1][8]+1;
         room_t[room][day-1][slot-1]=arr[t];
      }
     else
      {
         fact_t[faculty-1][day-1][slot-1]=arr[t];
         fact_t[faculty-1][day-1][slot]=arr[t];
         batch_t[batch-1][day-1][slot-1]=arr[t];
         batch_t[batch-1][day-1][slot]=arr[t];
         lroom_t[room][day-1][slot-1]=arr[t];
         lroom_t[room][day-1][slot]=arr[t];
       }
```

```
      int temp4=t;
      do
       {
          arr[temp4]=arr[temp4+1];
          temp4++;
       }
      while(arr[temp4]!=0);
     kul=0;
     }
     }
     break;
    }
    if(temp<1000)
    {
     return 1;
    }
    else
    {
     return 0;
    }
}
```

9.5 FITNESS CALCULATION

```
int Allocation::fitness()       // to calculate the fitness of a particular Timetable instance
{
    int currentact,currentsub;
    tfitness=0;
    char c;

    for(int i=0;i<16;i++)
    {
      for(int j=0;j<6;j++)
      {
         for(int k=0;k<8;k++)
         {
           currentact=batch_t[i][j][k];
           currentsub=(currentact/100)%100;
           q2=start2;
           for(int l=1;l<currentsub;l++)
           {
            q2=q2->next;
           }
            c=q2->name[0];
           if(c=='L'&&k<=4)
           {
             tfitness=tfitness+10;
           }
           if(c=='L'&&k==5)
           {
             tfitness=tfitness-1;
           }
           if(c=='L'&&k==6)
           {
             tfitness=tfitness-2;
           }
           if(c=='L'&&k==7)
           {
             tfitness=tfitness-2;
           }
           if(c!='L'&&(k==5||k==6))
           {
             tfitness=tfitness+3;
           }
           if(c!='L'&&k==7)
```

```
            {
              tfitness=tfitness-1;
            }
            if(c!='L'&&k<4)
            {
              tfitness=tfitness-10;
            }
            if(currentact==-1)
            {
              if(k>=0&&k<=2)
              {
              tfitness=tfitness-10;
              }
              if(k==5)
              {
              tfitness=tfitness-3;
              }
            }
        }
    }
}
int currentfact;
int currentact1;
int currentact2;
int currentact3;
int currentsub1;
int currentsub2;
int currentsub3;

char c1,c2,c3;

for(int i=0;i<16;i++)
{
   for(int j=0;j<6;j++)
   {
      for(int k=0;k<6;k++)
      {
         currentact1=fact_t[i][j][k];
         currentact2=fact_t[i][j][k+1];
         currentact3=fact_t[i][j][k+2];
         if(currentact1!=-1&&currentact2!=-1&&currentact3!=-1)
         {
             currentsub1=(currentact1/100)%100;
             currentsub2=(currentact2/100)%100;
             currentsub3=(currentact3/100)%100;
             q2=start2;
             for(int l=1;l<currentsub1;l++)
             {
               q2=q2->next;
             }
```

```
            c1=q2->name[0];
            q2=start2;
            for(int l=1;l<currentsub2;l++)
            {
              q2=q2->next;
            }
            c2=q2->name[0];
            q2=start2;
            for(int l=1;l<currentsub3;l++)
            {
              q2=q2->next;
            }
            c3=q2->name[0];
            if(c1=='L'&&c2=='L'&&c3=='L')
             {
               return -1;
             }
          }
       }
   }
}
return tfitness;
}
```

9.6 TIME TABLE GENERATION

```
void allocate()              // Start up function
{
    int i;
    for(i=0;i<5000;i++)
    {
     arr[i]=0;
    }
     fp=fopen("D:\database3.txt","r");
          if(fp == NULL)
    {
     printf("File does not exist,please check!\n");
    }
    else
    {
     printf("we are connected to the file!\n");
    }
    fac1();

    q1=start1;
    q2=start2;
    q3=start3;

    while(q1!=NULL)
    {
     xx1++;
     cout<<q1->name<<" "<<q1->code<<"\n";
     q1=q1->next;
    }
    while(q2!=NULL)
    {
     xx2++;
     cout<<q2->name<<" "<<q2->code<<"\n";
     q2=q2->next;
    }
    while(q3!=NULL)
    {
     xx3++;
     cout<<q3->name<<" "<<q3->code<<"\n";
     q3=q3->next;
    }
    i=0;
```

```
int arr2[5000];
while(arr[i]!=0)
{
 arr2[i]=arr[i];
 i++;
}
fclose(fp);
int ret,ret1;
kul=i;
int kuldeep=0;
int prashant=0;
int kuld=1,prash=1;
int generation=0;
while(generation<1000)
{
 cout<<"generation no. "<<generation<<"\n";
 generation++;
 if(kuld==1)
 {
  do
  {
   i=0;
   while(arr2[i]!=0)
   {
    arr[i]=arr2[i];
    i++;
   }
   kul=i;
   allow.create();
   ret=allow.algo(0);
  }
  while(ret==0);
 }
 if(prash==1)
  {
   do
   {
    i=0;
    while(arr2[i]!=0)
    {
     arr[i]=arr2[i];
     i++;
    }
    kul=i;
    allow1.create();
    ret1=allow1.algo(1);
   }
   while(ret1==0);
  }
  kuldeep=allow.fitness();
```

```
        prashant=allow1.fitness();
        if(kuldeep>prashant)
         {
            prash=1;
            kuld=0;
         }
         else
         {
           kuld=1;
           prash=0;
         }
       }
     if(kuldeep>prashant)
     {
      allow.print();
      cout<<"fitness is "<<kuldeep;
     }
     else
     {
       allow1.print();
       cout<<"fitness is "<<prashant;
     }
  getch();
}
```