**AUTOMATED TIMETABLE GENERATOR**

**USING GENETIC ALGORITHM**

Project report submitted in partial fulfilment of the

Requirements for the Award of the Degree of

B.Tech

In Computer Science and Engineering

BY

**PRANAV AGRAWAL**

**2008441**

Under the Guidance

of

**MR. MANISH SHARMA**

**Assistant Professor(Department of CS)**



**Department of Computer Science and Engineering**

**Graphic Era Deemed to be University**

**Dehradun-248002**

**2018**

**CERTIFICATE**

This is to certify that the project report entitled AUTOMATED TIMETABLE GENERATOR USING GENETIC ALGORITHM being submitted by

PRANAV AGRAWAL 2008441

SHIVAM CHAUDHARY 2008765

in partial fulfilment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Graphic Era Deemed to be University is a record of bonafied work carried out under my guidance and supervision.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Mr. Manish Sharma

Assistant Professor

Date: Head of the Department

**ACKNOWLEDGEMENT**

We wish to express our sincere gratitude to Professor Dr.Devesh Pratap Singh head of department (HOD), Computer Science Engineering, Graphic Era University, Dehradun, for providing us an opportunity to do this project  report “AUTOMATED TIMETABLE GENERATOR USING GENETIC ALGORITHM” . We sincerely thanks  Mr. Manish Sharma Assoc. Professor, Graphic Era University, Dehradun, for his guidance and encouragement throughout this project. I also wish to express my gratitude to the official and staff member of computer science department, Graphic Era University, Dehradun, who rendered their valuable help during this period of this project.  Last but not least, we wish to avail ourselves  of this opportunity, express a sense of gratitude and love to our beloved family for their manual support, strength and help for everything else.

**ABSTRACT**

Automatic Timetable Generator is a Java based application used to generate timetable automatically. Currently timetable is managed manually. It will help to manage all the periods automatically and also will be helpful for faculty to get timetable by using application. It will also manage timetable when any teacher is absent , late coming or early going. Maximum and minimum work load for a Faculty for a day, week and month will be specified for the efficient generation of timetable.

**Table of Contents**

1 Introduction …………………………………………………….1

1.1 Purpose ……………………………………………………….2

1.2 Beneﬁts ……………………………………………………….2

1.3 Similar Products ……………………………………………....2

2 Review of Literature ……………………………………………3

2.1 Linear Programming/Integer Programming ………………….3

2.2 Evolutionary and Genetic Algorithms ………………………..4

3 Automated TimeTable Generator……………………………… 6

3.1 Product Name …………………………………………………6

3.2 Product Description …………………………………………..6

3.3 Product Features……………………………………………….6

4 Requirement Analysis …………………………………………..8

4.1 System Conﬁguration ………………………………………… 8

4.2 Developer tools ………………………………………………..8

4.3 User tools ………………………………………………………9

4.4 Tools for Documentation ……………………………………..10

5 Overview ………………………………………………………...11

5.1 Use case diagram ……………………………………………... 11

5.2 Work Breakdown Structure …………………………………. 13

5.3 Data ﬂow diagram ……………………………………………..13

6 Implementation ………………………………………………....15

6.1 Interface Implementation ……………………………………..16

6.1.1 Login Interface ……………………………………………....16

6.1.2 Room Data……………………………………………..........16

6.1.3 TimeSlots / Lecture Data…………………………………… 16

6.1.4 Professor Data……………………………………………….16

6.1.5 Subject/Module Data………………………………………..17

6.1.6 Group Data………………………………………………….17

6.1.7 Timetable Output Table…………………………………….17

6.1.8 Save Table. ………………………………………………….18

6.2 Algorithm Implementation …………………………………...18

7 Source Code……………………………………………………...20

8 Testing ……………………………………………………………64

8.1 Functional Test Criteria ……………………………………… 65

8.2 Integration Testing …………………………………………… 65

8.3 User Acceptance Test ………………………………………….66

8.4 System Test Criteria …………………………………………. 66

8.5 Test cases and Test results ………………………………….. 66

9 Appendix……………………………………………………. ….67

9.1 Instalation Manual ……………………………………………67

9.2 User interface Scenarios …………………….………………..67

Bibliography ……………………………………..……………….68

List of Figures

1 Class hierarchy of java swing. . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2 Use case Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .12

3 Work Breakdown Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .13

4 Data ﬂow Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .14

5 Login . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

6 Data Entry Form . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

7 TimeTable Name Entry. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .63

8 Sample output . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63

**CHAPTER 1**

**Introduction**

Time table scheduling has been in human requirements since they thought of managing time eﬀectively. It is widely used in schools, colleges and other fields of teaching and working like crash courses, couching centres, training programs etc . In early days, time table scheduling was done manually with a single person or some group involved in task of scheduling it with their hands, which take lot of eﬀort and time.

While scheduling even the smallest constraints can take a lot of time and the case is even worse when the number of constraints or the amount of data to deal with increases. In such cases perfectly designed time table is reused for whole generation without any changes, proving to be dull in such situations. Other cases that can cause problem is when the number of employers/workers are weak, resulting in rescheduling of time table or they need to ﬁll on empty seats urgently.

Institutions/Schools/Collages/Universities are the regular users of such time tables. They need to schedule their course to meet the need of current duration and facilities that are available to them. However, their schedule should meet the requirement of new course addition and newly enrolled students to fresh batches. This may result in rescheduling the entire time table once again for its entire batches and to be scheduled in shortest possible time before the batches course start. Another problem that occur when scheduling time table for exams. When multiple batches have exam on same day, they need to be schedules eﬀectively taking into account all problems related to facilities that are available to conduct these exams simultaneously.

**1.1 Purpose**

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.

Our aim here is to develop a simple, easily understandable, eﬃcient and portable application, which could automatically generate good quality time tables with in seconds.

**1.2 Beneﬁts**

Automated Timetable Generator, our software allows users to generate time table for newly occurring changes in less time, with less eﬀort and with more eﬃciency. It will allow users to work on and view time tables in diﬀerent platforms and view diﬀerent information simultaneously.

**1.3 Similar Products**

• aSa Timetable

• Mimsa Small School

• Timetable Mate

**CHAPTER 2**

**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.

**2.1 Linear Programming/Integer Programming**

The Linear and Integer Programming techniques, the ﬁrst applied to time tabling, 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 (Thompson, 1967). The word ”programming” means planning in this context and is related to the type of application (Feiring, 1986). 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 (Bunday, 1984).

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 (Feiring,1986). 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 (Bunday, 1984). Linear programming problems are generally acknowledged to be eﬃciently solved by just three methods, namely the graphical method, the simplex method, and the transportation method (see eg, Palmers and Innes, 1976; Makower and Williamson, 1985).

The construction of a linear programming model involves three successive problem solving steps. The ﬁrst step identiﬁes the unknown or independent decision variables. Step two requires the identiﬁcation of the constraints and the formulation of these constraints as linear equations. Finally, in step three, the objective function is identiﬁed and written as a linear function of the decision variables.

**2.2 Evolutionary and Genetic Algorithms**

Evolutionary Algorithms (EAs) are a class of direct, probabilistic search and optimisation 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 Genetic Algorithm diﬀers from other search techniques in the following ways:

• Genetic Algorithms optimise the trade-oﬀ between exploring new points in the search space and exploiting the information discovered thus far.

• Genetic Algorithm s have the property of implicit parallelism. Implicit parallelism means that the GAs eﬀect is 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.

• Genetic Algorithms are 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.

• Genetic Algorithms operate 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 .

• Genetic Algorithm s work with a coding of the parameter set, not the parameters themselves.

• Genetic Algorithms search from a population of points, not a single point.

• Genetic Algorithms 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.

**CHAPTER 3**

**Automated Timetable Generator**

In this chapter we are presenting various details of our ﬁnal product. Since the projectis currently in a developing stage so most of the details are yet to be veriﬁed. So belowwe present the basic details.

**3.1 Product Description**

Automated Timetable Generator is a general purpose tool for generating diﬀerent types of tables. It cangive good outputs for most of the systems like school system, our university systems, and French university systems and so on.

Some of the most common constraints to deal with are listed below. Some of these are soft constraints meaning they only increase the cost. Some are hard which cannot be violated.

Hard Constraints

• No teacher or student must be assigned to more than one class.

• There should be required number of periods for each course. • Each course must have required consecutive periods. For example lab is assigned to 3 or 4 consecutive hours.

Soft Constraints

• Courses must be evenly distributed • Same teacher must not have consecutive periods unless speciﬁed.

**3.2 Product Features**

Some of the most important features oﬀered by our software are

• Simple wizard type user interface.

• Login authentication .

• Message dialogs for user assistance.

• Mouse or/and keyboard for inputs, keyboard shortcuts are available.

• Separate file maintaining basic information, subjects, teachers, batches and their associations and other details.

• File for holding generated timetable and for storing required timetables.

• High portability, works on almost all systems available.

• Highly Scalable. Works for few to large amount of data.

• Highly eﬃcient, needs only few minutes to complete whole procedure.

**CHAPTER 4**

**Requirement Analysis**

This chapter gives minimum requirement your system should have inorder to make this software work. This software works ﬁne in any operating system in which the developer tools or the user tools can be installed. Since we had limited resources we could only test in Widows 7, Windows XP, Ubuntu 11.04, Ubuntu 10.10. So usually the requirement speciﬁcation will be same as that of the operating system. So we are providing a standard speciﬁcation.

**4.1 System Conﬁguration**

• 1 GHz x86 processor

• 1GB of system memory (RAM)

• 15GB of hard-drive space

• Monitor to display output

• Keyboard/Mouse for data input

**4.2 Developer tools**

ECLIPSE IDE FOR JAVA

Eclipse is a community for individuals and organizations who wish to collaborate on commercially-friendly open source software. Its projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. The Eclipse Foundation is a not-for-profit, member supported corporation that hosts the Eclipse Projects and helps

cultivate both an open source community and an ecosystem of complementary products and services.



**User Interface Development**

**JAVA SWING** is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

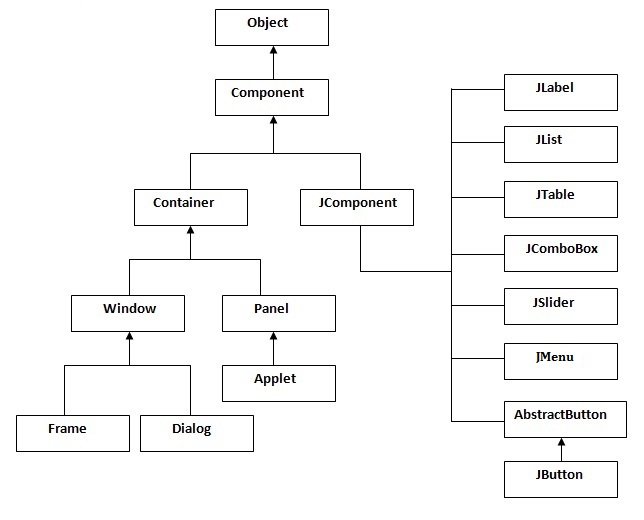


Fig: 1 Class hierarchy of java swing

**4.3 Tools for Documentation**

• Microsoft Office Word

• Windows Screenshot

• Adobe Photoshop

• Snipping Tool

**CHAPTER 5**

**Overview**

This chapter gives an overview of the system in the use case diagram, overview of the activites in the work break down diagram, overview of the working of activities in data ﬂow diagram, and overview of the database in ER diagram

**5.1 Use case diagram**

As shown in the ﬁgure the Data Entry Operator(DEO) is responsible of the system. He should be aware of various courses and subjects available in the college, he should also know the various rules and regulations of the institution. His job is as follows

• Collect teachers’ information

• Collect students’ information

• Input these information to the system

• Provide the output to others

• Make necessary modiﬁcations

• Maintain the file generated for future works

The DEO must have some basic knowledge about computers. He or she must have the skill to properly address the priorities. Time Gene will take care of the rest. It will provide teachers view for teachers and subject view for students.

A diagram of a diagram of people

Description automatically generated

Fig 2: Use Case Diagram

A diagram of a structure

Description automatically generated

**5.2 Work Breakdown Structure**

Above is a activity based work breakdown structure. The diagram shows how we divided the whole process into simpler ones. Each of the process is carried out by various group members. Most of the steps are repeated again and again in the life cycle of the project

**5.3 Data ﬂow diagram**

The data ﬂow through database is described in following diagram. Each process represent the working of user with the interface and followed by its associated operation performed in the database. Here open square represents the operation that is performed in databases.

A diagram of a workflow

Description automatically generated

**CHAPTER 6**

**Implementation**

This application has been developed using Java as front end tool and text document as its back end tool. The application has been coded to be platform independent running on Java Virtual Machine.Eclipse IDE has been chosen as its development environment because of the

following features

• Designing interface for the application has been simpliﬁed by its drag and drop GUI pallet.

• Debugging can be easily done using the Logger class.

• Simpliﬁed automated editor error detection.

• Automatic code generation.

• Automatic documentation.

• Simpliﬁed class factory method lookups.

• Easy to create jar ﬁles using build option.

• Proﬁling option.

• Project can be run on debugging mode which provides current state of the variables with the help of break points.

While coding Timetable generator application several constraints related to its computation has been taken into account.Timetable generating problem provides us with various alternatives in the designof the algorithm, interface and the database. Among the various designs what we have implemented is detailed below

**6.1 Interface Implementation**

There are twelve classes contains a Frame which is associated with an interface.The association are as follows

• Class.class

• GeneticAlgorithm.class

•Group.class

• Individual.class

• Batches.class

• Module.class

• Population.class

• Professor.class

• Room.class

• TimeSlot

• Timetable

• TimetableGA

• password.txt

**6.1.1 Login Interface**

Enter the login name and password as a primary information

**6.1.2 Room Data**

Enter the Room ID, Enter the Room name and Enter the Room Capacity each in the next line.

**6.1.3 TimeSlots / Lecture Data**

Enter the TimeSlots ID and enter TimeSlots duration each in the next line.

**6.1.4 Professor Data**

Enter the Professor ID and Professor name each in the next line.

**6.1.5 Subject/Module Data**

Enter the Module ID , Module code, Module name and Professor IDs who teaches these modules each in the next line and each column separated by comma ”,”.

**6.1.6 Group Data**

Select the Group ID , Group Size and Module ID corresponding to each group.

**6.1.7 Timetable Output Table**

After clicking Submit button a prompt appear . Prompt ask for the Timetable name. Corresponding to it a new frame appear with the timetable generated.

**6.1.8 Save Table**

Name entered for the name of the timetable is used as saving the file name also which is a text file.

**6.2 Algorithm Implementation**

TimeTable generation is an NP-Complete problem; speciﬁcally speaking NP-Hard. So it lacks a proper time bound for execution i.e. problems like these often can have many diﬀerent outputs. So we assign cost to each output which gives the measure of deviation of the output from the desired one. So our aim is to get the output with minimum cost if there is one.

Genetic algorithm can give best results but the time needed for it to compute cannot be determined so we have developed an alternative approach which can be applied to solve most of the NP-Complete problems.

• First determine the various constraints which the output must satisfy.

• We then categorize them as soft and hard constraints.

• Third step is to make a procedure which can generate an output for most of the possible inputs.

• The ﬁnal step is to reduce the cost.

The current working scenarios these can be explained as follows. The ﬁrst two steps are explained earlier. The procedure mentioned in the third step here is the gene() function in the TimeTableGA class. What this function does is to assign properties to teacher, subject and position. So that if we arrange timetable according

to this priority there is a greater probability to end up in the output which satisﬁes all the hard constraints.

First and foremost priority is that of teachers. Subject priority and position priority depends on teacher priority, also teacher is the most important resource in the time table.

The priority of teacher increases if the number of periods he handled increases, also if the number of batches he is present increases and decreases if an alternative teacher is available. Subject priority is also similar to that of teacher priority. It must also increase with the continuous hours needed. So in the algorithm we have considered subject which have diﬀerent consecutive periods need as diﬀerent subjects. Position has priority if it correctly ﬁts and the adjacent portions are not that of the same subject.

Finally we need to manually assign priority and optimize the table to reduce cost. The main logic of our algorithm resides in the TimetableGA class. The gene function in the class reads the various data needed for generation of timetable for current batches from the GUI creates the table in an String and copies it back .

**CHAPTER 7**

**Source Code**

**Class.java**

package pro8;

public class Class {

private final int classId;

private final int groupId;

private final int moduleId;

private int professorId;

private int timeslotId;

private int roomId;

public Class(int classId, int groupId, int moduleId){

this.classId = classId;

this.moduleId = moduleId;

this.groupId = groupId;

}

public void addProfessor(int professorId){

this.professorId = professorId;

}

public void addTimeslot(int timeslotId){

this.timeslotId = timeslotId;

}

public void setRoomId(int roomId){

this.roomId = roomId;

}

public int getClassId(){

return this.classId;

}

public int getGroupId(){

return this.groupId;

}

public int getModuleId(){

return this.moduleId;

}

public int getProfessorId(){

return this.professorId;

}

public int getTimeslotId(){

return this.timeslotId;

}

public int getRoomId(){

return this.roomId;

}

}

**Group.java**

package pro8;

public class Group {

private final int groupId;

private final int groupSize;

private final int moduleIds[];

public Group(int groupId, int groupSize, int moduleIds[]){

this.groupId = groupId;

this.groupSize = groupSize;

this.moduleIds = moduleIds;

}

public int getGroupId(){

return this.groupId;

}

public int getGroupSize(){

return this.groupSize;

}

public int[] getModuleIds(){

return this.moduleIds;

}

}

**Individual.java**

package pro8;

public class Individual {

private int[] chromosome;

private double fitness = -1;

public Individual(Timetable timetable) {

int numClasses = timetable.getNumClasses();

int chromosomeLength = numClasses \* 3;

int newChromosome[] = new int[chromosomeLength];

int chromosomeIndex = 0;

for (Group group : timetable.getGroupsAsArray()) {

for (int moduleId : group.getModuleIds()) {

int timeslotId = timetable.getRandomTimeslot().getTimeslotId();

newChromosome[chromosomeIndex] = timeslotId;

chromosomeIndex++;

int roomId = timetable.getRandomRoom().getRoomId();

newChromosome[chromosomeIndex] = roomId;

chromosomeIndex++;

Module module = timetable.getModule(moduleId);

newChromosome[chromosomeIndex] = module.getRandomProfessorId();

chromosomeIndex++;

}

}

this.chromosome = newChromosome;

}

public Individual(int chromosomeLength) {

int[] individual;

individual = new int[chromosomeLength];

for (int gene = 0; gene < chromosomeLength; gene++) {

individual[gene] = gene;

}

this.chromosome = individual;

}

public Individual(int[] chromosome) {

this.chromosome = chromosome;

}

public int[] getChromosome() {

return this.chromosome;

}

public int getChromosomeLength() {

return this.chromosome.length;

}

public void setGene(int offset, int gene) {

this.chromosome[offset] = gene;

}

public int getGene(int offset) {

return this.chromosome[offset];

}

public void setFitness(double fitness) {

this.fitness = fitness;

}

public double getFitness() {

return this.fitness;

}

public String toString() {

String output = "";

for (int gene = 0; gene < this.chromosome.length; gene++) {

output += this.chromosome[gene] + ",";

}

return output;

}

public boolean containsGene(int gene) {

for (int i = 0; i < this.chromosome.length; i++) {

if (this.chromosome[i] == gene) {

return true;

}

}

return false;

}

}

**Module.java**

package pro8;

public class Module {

private final int moduleId;

private final String moduleCode;

private final String module;

private final int professorIds[];

public Module(int moduleId, String moduleCode, String module, int professorIds[]){

this.moduleId = moduleId;

this.moduleCode = moduleCode;

this.module = module;

this.professorIds = professorIds;

}

public int getModuleId(){

return this.moduleId;

}

public String getModuleCode(){

return this.moduleCode;

}

public String getModuleName(){

return this.module;

}

public int getRandomProfessorId(){

int professorId = professorIds[(int) (professorIds.length \* Math.*random*())];

return professorId;

}

}

**Professor.java**

package pro8;

public class Professor {

private final int professorId;

private final String professorName;

public Professor(int professorId, String professorName){

this.professorId = professorId;

this.professorName = professorName;

}

public int getProfessorId(){

return this.professorId;

}

public String getProfessorName(){

return this.professorName;

}

}

**Room.java**

package pro8;

public class Room {

private final int roomId;

private final String roomNumber;

private final int capacity;

public Room(int roomId, String roomNumber, int capacity) {

this.roomId = roomId;

this.roomNumber = roomNumber;

this.capacity = capacity;

}

int getRoomId() {

return this.roomId;

}

public String getRoomNumber() {

return this.roomNumber;

}

public int getRoomCapacity() {

return this.capacity;

}

}

**Timeslot.java**

package pro8;

public class Timeslot {

private final int timeslotId;

private final String timeslot;

public Timeslot(int timeslotId, String timeslot){

this.timeslotId = timeslotId;

this.timeslot = timeslot;

}

public int getTimeslotId(){

return this.timeslotId;

}

public String getTimeslot(){

return this.timeslot;

}

}

GeneticAlgorithm.java

package pro8;

public class GeneticAlgorithm {

private int populationSize;

private double mutationRate;

private double crossoverRate;

private int elitismCount;

protected int tournamentSize;

public GeneticAlgorithm(int populationSize, double mutationRate, double crossoverRate, int elitismCount,

int tournamentSize) {

this.populationSize = populationSize;

this.mutationRate = mutationRate;

this.crossoverRate = crossoverRate;

this.elitismCount = elitismCount;

this.tournamentSize = tournamentSize;

}

public Population initPopulation(Timetable timetable) {

Population population = new Population(this.populationSize, timetable);

return population;

}

public boolean isTerminationConditionMet(int generationsCount, int maxGenerations) {

return (generationsCount > maxGenerations);

}

public boolean isTerminationConditionMet(Population population) {

return population.getFittest(0).getFitness() == 1.0;

}

public double calcFitness(Individual individual, Timetable timetable) {

Timetable threadTimetable = new Timetable(timetable);

threadTimetable.createClasses(individual);

int clashes = threadTimetable.calcClashes();

double fitness = 1 / (double) (clashes + 1);

individual.setFitness(fitness);

return fitness;

}

public void evalPopulation(Population population, Timetable timetable) {

double populationFitness = 0;

for (Individual individual : population.getIndividuals()) {

populationFitness += this.calcFitness(individual, timetable);

}

population.setPopulationFitness(populationFitness);

}

public Individual selectParent(Population population) {

Population tournament = new Population(this.tournamentSize);

population.shuffle();

for (int i = 0; i < this.tournamentSize; i++) {

Individual tournamentIndividual = population.getIndividual(i);

tournament.setIndividual(i, tournamentIndividual);

}

return tournament.getFittest(0);

}

public Population mutatePopulation(Population population, Timetable timetable) {

Population newPopulation = new Population(this.populationSize);

for (int populationIndex = 0; populationIndex < population.size(); populationIndex++) {

Individual individual = population.getFittest(populationIndex);

Individual randomIndividual = new Individual(timetable);

for (int geneIndex = 0; geneIndex < individual.getChromosomeLength(); geneIndex++) {

if (populationIndex > this.elitismCount) {

if (this.mutationRate > Math.*random*()) {

individual.setGene(geneIndex, randomIndividual.getGene(geneIndex));

}

}

}

newPopulation.setIndividual(populationIndex, individual);

}

return newPopulation;

}

public Population crossoverPopulation(Population population) {

Population newPopulation = new Population(population.size());

for (int populationIndex = 0; populationIndex < population.size(); populationIndex++) {

Individual parent1 = population.getFittest(populationIndex);

if (this.crossoverRate > Math.*random*() && populationIndex >= this.elitismCount) {

Individual offspring = new Individual(parent1.getChromosomeLength());

Individual parent2 = selectParent(population);

for (int geneIndex = 0; geneIndex < parent1.getChromosomeLength(); geneIndex++) {

if (0.5 > Math.*random*()) {

offspring.setGene(geneIndex, parent1.getGene(geneIndex));

} else {

offspring.setGene(geneIndex, parent2.getGene(geneIndex));

}

}

newPopulation.setIndividual(populationIndex, offspring);

} else {

newPopulation.setIndividual(populationIndex, parent1);

}

}

return newPopulation;

}

}

**Population.java**

package pro8;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Random;

public class Population {

private Individual population[];

private double populationFitness = -1;

public Population(int populationSize) {

// Initial population

this.population = new Individual[populationSize];

}

public Population(int populationSize, Timetable timetable) {

this.population = new Individual[populationSize];

for (int individualCount = 0; individualCount < populationSize; individualCount++) {

Individual individual = new Individual(timetable);

this.population[individualCount] = individual;

}

}

public Population(int populationSize, int chromosomeLength) {

this.population = new Individual[populationSize];

for (int individualCount = 0; individualCount < populationSize; individualCount++) {

Individual individual = new Individual(chromosomeLength);

this.population[individualCount] = individual;

}

}

public Individual[] getIndividuals() {

return this.population;

}

public Individual getFittest(int offset) {

Arrays.*sort*(this.population, new Comparator<Individual>() {

@Override

public int compare(Individual o1, Individual o2) {

if (o1.getFitness() > o2.getFitness()) {

return -1;

} else if (o1.getFitness() < o2.getFitness()) {

return 1;

}

return 0;

}

});

return this.population[offset];

}

public void setPopulationFitness(double fitness) {

this.populationFitness = fitness;

}

public double getPopulationFitness() {

return this.populationFitness;

}

public int size() {

return this.population.length;

}

public Individual setIndividual(int offset, Individual individual) {

return population[offset] = individual;

}

public Individual getIndividual(int offset) {

return population[offset];

}

public void shuffle() {

Random rnd = new Random();

for (int i = population.length - 1; i > 0; i--) {

int index = rnd.nextInt(i + 1);

Individual a = population[index]; //swap index population values

population[index] = population[i];

population[i] = a;

}

}

}

**Timetable.java**

package pro8;

import java.util.HashMap;

public class Timetable {

private final HashMap<Integer, Room> rooms;

private final HashMap<Integer, Professor> professors;

private final HashMap<Integer, Module> modules;

private final HashMap<Integer, Group> groups;

private final HashMap<Integer, Timeslot> timeslots;

private Class classes[];

private int numClasses = 0;

public Timetable() {

this.rooms = new HashMap<Integer, Room>();

this.professors = new HashMap<Integer, Professor>();

this.modules = new HashMap<Integer, Module>();

this.groups = new HashMap<Integer, Group>();

this.timeslots = new HashMap<Integer, Timeslot>();

}

public Timetable(Timetable cloneable) {

this.rooms = cloneable.getRooms();

this.professors = cloneable.getProfessors();

this.modules = cloneable.getModules();

this.groups = cloneable.getGroups();

this.timeslots = cloneable.getTimeslots();

}

private HashMap<Integer, Group> getGroups() {

return this.groups;

}

private HashMap<Integer, Timeslot> getTimeslots() {

return this.timeslots;

}

private HashMap<Integer, Module> getModules() {

return this.modules;

}

private HashMap<Integer, Professor> getProfessors() {

return this.professors;

}

public void addRoom(int roomId, String roomName, int capacity) {

this.rooms.put(roomId, new Room(roomId, roomName, capacity));

}

public void addProfessor(int professorId, String professorName) {

this.professors.put(professorId, new Professor(professorId, professorName));

}

public void addModule(int moduleId, String moduleCode, String module, int professorIds[]) {

this.modules.put(moduleId, new Module(moduleId, moduleCode, module, professorIds));

}

public void addGroup(int groupId, int groupSize, int moduleIds[]) {

this.groups.put(groupId, new Group(groupId, groupSize, moduleIds));

this.numClasses = 0;

}

public void addTimeslot(int timeslotId, String timeslot) {

this.timeslots.put(timeslotId, new Timeslot(timeslotId, timeslot));

}

public void createClasses(Individual individual) {

Class classes[] = new Class[this.getNumClasses()];

int chromosome[] = individual.getChromosome();

int chromosomePos = 0;

int classIndex = 0;

for (Group group : this.getGroupsAsArray()) {

int moduleIds[] = group.getModuleIds();

for (int moduleId : moduleIds) {

classes[classIndex] = new Class(classIndex, group.getGroupId(), moduleId);

classes[classIndex].addTimeslot(chromosome[chromosomePos]);

chromosomePos++;

classes[classIndex].setRoomId(chromosome[chromosomePos]);

chromosomePos++;

classes[classIndex].addProfessor(chromosome[chromosomePos]);

chromosomePos++;

classIndex++;

}

}

this.classes = classes;

}

public Room getRoom(int roomId) {

if (!this.rooms.containsKey(roomId)) {

System.*out*.println("Rooms doesn't contain key " + roomId);

}

return (Room) this.rooms.get(roomId);

}

public HashMap<Integer, Room> getRooms() {

return this.rooms;

}

public Room getRandomRoom() {

Object[] roomsArray = this.rooms.values().toArray();

Room room = (Room) roomsArray[(int) (roomsArray.length \* Math.*random*())];

return room;

}

public Professor getProfessor(int professorId) {

return (Professor) this.professors.get(professorId);

}

public Module getModule(int moduleId) {

return (Module) this.modules.get(moduleId);

}

public int[] getGroupModules(int groupId) {

Group group = (Group) this.groups.get(groupId);

return group.getModuleIds();

}

public Group getGroup(int groupId) {

return (Group) this.groups.get(groupId);

}

public Group[] getGroupsAsArray() {

return (Group[]) this.groups.values().toArray(new Group[this.groups.size()]);

}

public Timeslot getTimeslot(int timeslotId) {

return (Timeslot) this.timeslots.get(timeslotId);

}

public Timeslot getRandomTimeslot() {

Object[] timeslotArray = this.timeslots.values().toArray();

Timeslot timeslot = (Timeslot) timeslotArray[(int) (timeslotArray.length \* Math.*random*())];

return timeslot;

}

public Class[] getClasses() {

return this.classes;

}

public int getNumClasses() {

if (this.numClasses > 0) {

return this.numClasses;

}

int numClasses = 0;

Group groups[] = (Group[]) this.groups.values().toArray(new Group[this.groups.size()]);

for (Group group : groups) {

numClasses += group.getModuleIds().length;

}

this.numClasses = numClasses;

return this.numClasses;

}

public int calcClashes() {

int clashes = 0;

for (Class classA : this.classes) {

int roomCapacity = this.getRoom(classA.getRoomId()).getRoomCapacity();

int groupSize = this.getGroup(classA.getGroupId()).getGroupSize();

if (roomCapacity < groupSize) {

clashes++;

}

for (Class classB : this.classes) {

if (classA.getRoomId() == classB.getRoomId() && classA.getTimeslotId() == classB.getTimeslotId()

&& classA.getClassId() != classB.getClassId()) {

clashes++;

break;

}

}

for (Class classB : this.classes) {

if (classA.getProfessorId() == classB.getProfessorId() && classA.getTimeslotId() == classB.getTimeslotId()

&& classA.getClassId() != classB.getClassId()) {

clashes++;

break;

}

}

}

return clashes;

}

}

**TimetableGA.java**

package pro8;

import java.awt.Button;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.Frame;

import java.awt.Label;

import java.awt.Panel;

import java.awt.TextArea;

import java.awt.TextField;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\* ;

import javax.swing.table.DefaultTableModel;

import javax.swing.text.SimpleAttributeSet;

import javax.xml.crypto.Data;

public class TimetableGA extends Frame implements ActionListener {

String usnm;

String pwd;

Label l1=new Label("user name");

Label l2=new Label("password");

Label l3=new Label(" ");

Label l4=new Label("Want to Reset your password ?");

Label l5=new Label("User name ");

Label l6=new Label("New Password ");

Label l7=new Label("Confirmed password");

Label l8=new Label(" ");

Label l9=new Label("");

TextField t1=new TextField("username"); //user name

TextField t2=new TextField("password");//password

TextField t3=new TextField("user name");//User name

TextField t4=new TextField("password1");//new password

TextField t5=new TextField("con-password");//confirmed password

Button b= new Button("Submit");

Button b2= new Button("Reset");

Label AA=new Label("ROOM DATA");

Label BB=new Label("TIMESLOT/LECTURE DATA");

Label CC=new Label("PROFESSORS DATA");

Label DD=new Label("SUBJECT/MODULE DATA");

Label EE=new Label("GROUP DATA");

Label l11=new Label("Enter rooms' ID:");

Label l22=new Label("Enter rooms' name:");

Label l33=new Label("Enter rooms' capacity:");

Label l44=new Label("Enter timeslots' ID:");

Label l55=new Label("Enter timeslots' duration:");

Label l66=new Label("Enter professors' ID:");

Label l77=new Label("Enter professor' name:");

Label l88=new Label("Enter modules' ID:");

Label l99=new Label("Enter modules' code:");

Label l1010=new Label("Enter modules' name:");

Label l1111=new Label("Enter professors' ID"); //who teaches these modules:");

Label l1212=new Label("Enter groups' ID:");

Label l1313=new Label("Enter groups' size:");

Label l1414=new Label("Enter modules IDs");//corresponding to each group:");

Label l1616=new Label("..");

TextArea ta1=new TextArea("Enter IDs separated by enter");

TextArea ta2=new TextArea();

TextArea ta3=new TextArea();

TextArea ta4=new TextArea();

TextArea ta5=new TextArea();

TextArea ta6=new TextArea();

TextArea ta7=new TextArea();

TextArea ta8=new TextArea();

TextArea ta9=new TextArea();

TextArea ta10=new TextArea();

TextArea ta11=new TextArea();

TextArea ta12=new TextArea();

TextArea ta13=new TextArea();

TextArea ta14=new TextArea();

Button b123= new Button("CREATE");

public static String rid[],rname[],rcap[],tid[],tslot[],pid[],pname[],mid[],mcode[],mname[],mpid[],gid[],gsize[],gmid[];

public static void main(String[] args) {

TimetableGA l=new TimetableGA();

l.setSize(new Dimension(400,400));

l.setTitle("Login");

l.setVisible(true);

}

public TimetableGA(int i){

AA.setBounds(10,45,1350,20);

AA.setAlignment(Label.CENTER);

AA.setBackground(Color.YELLOW);

AA.setForeground(Color.BLUE);

AA.setFont(new Font("Serif", Font.BOLD, 19));

BB.setBounds(10,175,1350,20);

BB.setAlignment(Label.CENTER);

BB.setBackground(Color.YELLOW);

BB.setForeground(Color.BLUE);

BB.setFont(new Font("Serif", Font.BOLD, 19));

CC.setBounds(10,305,1350,20);

CC.setAlignment(Label.CENTER);

CC.setBackground(Color.YELLOW);

CC.setForeground(Color.BLUE);

CC.setFont(new Font("Serif", Font.BOLD, 19));

DD.setBounds(10,435,1350,20);

DD.setAlignment(Label.CENTER);

DD.setBackground(Color.YELLOW);

DD.setForeground(Color.BLUE);

DD.setFont(new Font("Serif", Font.BOLD, 19));

EE.setBounds(10,565,1350,20);

EE.setAlignment(Label.CENTER);

EE.setBackground(Color.YELLOW);

EE.setForeground(Color.BLUE);

EE.setFont(new Font("Serif", Font.BOLD, 19));

b123.setBounds(640,695,100,20);

b123.setBackground(Color.GREEN);

b123.setFont(new Font("Serif", Font.BOLD, 19));

l11.setBounds(60,65,100,20);

ta1.setBounds(160,65,150,70);

l22.setBounds(400,65,115,20);

ta2.setBounds(520,65,150,70);

l33.setBounds(750,65,130,20);

ta3.setBounds(880,65,145,70);

l44.setBounds(180,200,120,20);

ta4.setBounds(300,200,150,70);

l55.setBounds(540,200,150,20);

ta5.setBounds(700,200,150,70);

l66.setBounds(180,330,120,20);

ta6.setBounds(300,330,150,70);

l77.setBounds(540,330,150,20);

ta7.setBounds(700,330,150,70);

l88.setBounds(40,455,110,20);

ta8.setBounds(150,455,150,70);

l99.setBounds(330,455,118,20);

ta9.setBounds(450,455,150,70);

l1010.setBounds(630,455,130,20);

ta10.setBounds(760,455,150,70);

l1111.setBounds(950,455,110,20);/////////////who teaches these modules:

Label l11a=new Label("who teaches these "); l11a.setBounds(950,475,110,20);

Label l11b=new Label("modules:"); l11b.setBounds(950,495,110,20);

ta11.setBounds(1070,455,150,70);

l1212.setBounds(60,585,100,20);

ta12.setBounds(160,585,150,70);

l1313.setBounds(400,585,105,20);

ta13.setBounds(520,585,150,70);

l1414.setBounds(750,585,110,20);

Label l14a=new Label("corresponding to"); l14a.setBounds(750,605,110,20);

Label l14b=new Label("each group:"); l14b.setBounds(750,625,110,20);

ta14.setBounds(880,585,150,70);

b123.addActionListener(this);

//addWindowListener(new mwa1());

add(AA);

add(BB);

add(CC);

add(DD);

add(EE);

add(l11);

add(ta1);

add(l22);

add(ta2);

add(l33);

add(ta3);

add(l44);

add(ta4);

add(l55);

add(ta5);

add(l66);

add(ta6);

add(l77);

add(ta7);

add(l88);

add(ta8);

add(l99);

add(ta9);

add(l1010);

add(ta10);

add(l1111); add(l11a); add(l11b);

add(ta11);

add(l1212);

add(ta12);

add(l1313);

add(ta13);

add(l1414); add(l14a); add(l14b);

add(ta14);

add(b123);

//add(l1515);

add(l1616);

addWindowListener(new mwa());

}

public TimetableGA()

{

Panel p=new Panel();

p.setBackground(Color.pink);

add(l1);

add(t1);

add(l2);

add(t2);

add(b);

add(l3);

p.add(l4);

p.add(l5);

p.add(t3);

p.add(l6);

p.add(t4);

p.add(l7);

p.add(t5);

p.add(b2);

p.add(l8);

p.add(l9);

add(p);

add(l9);

l1.setBounds(20,45,70,20);

t1.setBounds(180,45,200,20);

l2.setBounds(20,95,70,20);

t2.setBounds(180,95,200,20);

b.setBounds(280,145,70,40);

b.addActionListener(this);//first button

t2.setEchoChar('\*');

l3.setBounds(20, 145, 300, 20);

b2.setBounds(360,395,70,20);

b2.addActionListener(this); //second button

l4.setBounds(20,195,300,20);

l5.setBounds(20,245,300,20);

t3.setBounds(380,245,200,20);

l6.setBounds(20,295,300,20);

t4.setBounds(380,295,200,20);

l7.setBounds(20,345,300,20);

t5.setBounds(380,345,200,20);

l8.setBounds(380,345,200,20);

p.setBounds(60, 200, 250, 160);

addWindowListener(new mwa());

}

public boolean validity(String userName , String password) throws FileNotFoundException{

File file = new File("C:\\Users\\PC\\workspace\\pro8\\src\\pro8\\password.txt");

Scanner scr = null;

String code="";

String name="";

try {

scr = new Scanner(file);

name=(scr.next());

code=(scr.next());

}

finally{

}

if((code.equals(t2.getText()))&& (name.equals(t1.getText()))){

return true;

}

else{

return false;

}

}

public void Reset() throws IOException

{

String newPassword=t4.getText();

String conPassword=t5.getText();

if(newPassword.equals(conPassword)){

try (FileWriter fileWriter = new FileWriter("C:\\Users\\PC\\workspace\\pro8\\src\\pro8\\password.txt")) {

fileWriter.write(t3.getText());

fileWriter.write("\n");

fileWriter.write(t4.getText());

}

finally{

l3.setText("password changed");

}

}

else{

l3.setText("new and confirmed passwords are not same");

}

}

public void actionPerformed(ActionEvent ae)

{

String str=ae.getActionCommand();

if(str.equals("Submit")){

usnm=t1.getText();

pwd=t2.getText();

try {

if(validity(usnm, pwd)){

l3.setText("Welcome "+t1.getText());

int i;

TimetableGA tga=new TimetableGA(1);

tga.setSize(new Dimension(1370,890));

tga.setTitle("Data Entry Form");

tga.setVisible(true);

}

else{

l3.setText("Invalid Username or Password");

}

}

catch (FileNotFoundException e) {

e.printStackTrace();

}

}

else{

try {

Reset();

} catch (IOException e) {

e.printStackTrace();

}

}

//try{

if(str.equals("CREATE")){

//write code here

rid=ta1.getText().split("\\r?\\n");

rname=ta2.getText().split("\\r?\\n");

rcap=ta3.getText().split("\\r?\\n");

tid=ta4.getText().split("\\r?\\n");

tslot=ta5.getText().split("\\r?\\n");

pid=ta6.getText().split("\\r?\\n");

pname=ta7.getText().split("\\r?\\n");

mid=ta8.getText().split("\\r?\\n");

mcode=ta9.getText().split("\\r?\\n");

mname=ta10.getText().split("\\r?\\n");

mpid=ta11.getText().split("\\r?\\n");

gid=ta12.getText().split("\\r?\\n");

gsize=ta13.getText().split("\\r?\\n");

gmid=ta14.getText().split("\\r?\\n");

mainFunction();

}

}

class mwa extends WindowAdapter

{ public mwa(){}

public void windowClosing(WindowEvent e)

{ System.exit(0);

}

}

public static int[] stoi(String s){

String[] integerStrings = s.split("[,]");//[.]

int[] integers = new int[integerStrings.length];

for (int i = 0; i < integers.length; i++){

integers[i] = Integer.parseInt(integerStrings[i]);

}

return integers;

}

public static void mainFunction(){

Timetable timetable = initializeTimetable();

GeneticAlgorithm ga = new GeneticAlgorithm(100, 0.01, 0.9, 2, 5);

Population population = ga.initPopulation(timetable);

ga.evalPopulation(population, timetable);

int generation = 1;

while (ga.isTerminationConditionMet(generation, 1000) == false

&& ga.isTerminationConditionMet(population) == false) {

System.out.println("Generation # " + generation + " Best fitness: " + population.getFittest(0).getFitness());

population = ga.crossoverPopulation(population);

population = ga.mutatePopulation(population, timetable);

ga.evalPopulation(population, timetable);

generation++;

}

// Print fitness

timetable.createClasses(population.getFittest(0));

System.out.println();

System.out.println("Solution found in " + generation + " generations");

System.out.println("Final solution fitness: " + population.getFittest(0).getFitness());

System.out.println("Clashes: " + timetable.calcClashes());

DefaultTableModel model = new DefaultTableModel();

JTable table = new JTable(model);

model.addColumn("S.No.");

model.addColumn("Module");

model.addColumn("Group");

model.addColumn("Room");

model.addColumn("Professor");

model.addColumn("Time");

Class classes[] = timetable.getClasses();

int classIndex = 1;

for (Class bestClass : classes) {

model.addRow(new Object[] { classIndex,timetable.getModule(bestClass.getModuleId()).getModuleName(),timetable.getGroup(bestClass.getGroupId()).getGroupId(),timetable.getRoom(bestClass.getRoomId()).getRoomNumber(),timetable.getProfessor(bestClass.getProfessorId()).getProfessorName(),timetable.getTimeslot(bestClass.getTimeslotId()).getTimeslot()});

classIndex++;

}

JFrame f = new JFrame();

String input = JOptionPane.showInputDialog("Enter Timtable Name");

f.setTitle(input);

f.setSize(700, 500);

f.add(new JScrollPane(table));

f.setVisible(true);

File file = new File("C:/Users/PC/workspace/pro8/src/pro8/"+input+".txt");

FileWriter fr = null;

try {

fr = new FileWriter(file);

classIndex = 1;

fr.write(String.format("%20s %20s %20s %20s %20s %20s \r\n", "S.No.", "Group", "Module", "Room", "Professor", "Time"));

fr.write("-------------------------------------------------------------------------------------------------------------------------------------");

fr.write("\r\n");

for (Class bestClass : classes) {

fr.write(String.format("%20s %20s %20s %20s %20s %20s \r\n", classIndex,timetable.getModule(bestClass.getModuleId()).getModuleName(),timetable.getGroup(bestClass.getGroupId()).getGroupId(),timetable.getRoom(bestClass.getRoomId()).getRoomNumber(),timetable.getProfessor(bestClass.getProfessorId()).getProfessorName(), timetable.getTimeslot(bestClass.getTimeslotId()).getTimeslot()));

classIndex++;

}

fr.write("-------------------------------------------------------------------------------------------------------------------------------------");

fr.write("\r\n");

} catch (IOException e) {

e.printStackTrace();

}finally{

try {

fr.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

private static Timetable initializeTimetable() {

// Create timetable

Timetable timetable = new Timetable();

try{

for(int i=0;i<rid.length;i++){

timetable.addRoom(Integer.parseInt(rid[i]), rname[i], Integer.parseInt(rcap[i]));

}

for(int i=0;i<tid.length;i++){

timetable.addTimeslot(Integer.parseInt(tid[i]), tslot[i]);

}

for(int i=0;i<pid.length;i++){

timetable.addProfessor(Integer.parseInt(pid[i]), pname[i]);

}

for(int i=0;i<mid.length;i++){

timetable.addModule(Integer.parseInt(mid[i]), mcode[i], mname[i], stoi(mpid[i]));

}

for(int i=0;i<gid.length;i++){

timetable.addGroup(Integer.parseInt(gid[i]), Integer.parseInt(gsize[i]), stoi(gmid[i]));

}

return timetable;

}

catch(Exception e){

final JPanel panel = new JPanel();

JOptionPane.showMessageDialog(panel, "Enter The Correct Data !!", "Error", JOptionPane.ERROR\_MESSAGE);

return null;

}

}

}

A screenshot of a computer

Description automatically generated

Fig 5: Login Form

A screenshot of a computer

Description automatically generated

Fig 6:Data Entry Form

A screenshot of a computer

Description automatically generated

Fig 7:TimeTable Name Entry

A screenshot of a computer

Description automatically generated

Fig 8:Sample output

**CHAPTER 8**

**Testing**

Testing is an important phase in software lifecycle. Testing improves reliablity and robustness of the application. The basic operations to be tested are

• There should be at least one working day and one working hour

• Diﬀerent inputs must be checked for its range. For example no of hours in morning or evening should be between 0-5, total number of periods for a course must be between 0-70, code for course must be 0-8 character long.

• User should not be given permission to edit or modify subjects, teachers or batches without releasing its associations

• Before generating table it should be checked if all subjects are assigned at least one teacher, no of periods available((morning hours +evening hours)\*no of days) must be equal to no of periods assigned.

• If any problem occurs during generation (due to constraints) it must be properly displayed.

**Test Approach**

The system is to be tested at various stages of the project development: Each user interface is tested individually for its function. Interfaces meant for data input are tested by entering data in the data tables through each interface. Similarly each data base operation is tested through interfaces.

All the user interfaces are joined in the desired sequence and their back end coding is tested for the desired result. Like previous window close as the next desired window opens and each button performs its desired task etc.

Every class in the java code is tested individually with the help of test cases.

Whole of the algorithm is tested with sample run of data to generate an optimal time table for the provided database.

**Test Planning**

Most of the testing requires checking connectivity of the user interfaces, so a properly designed text file is required for testing. Design interfaces and connect each of them to the database and test them for proper output as is it is in the database.

Inter connect all the user interfaces in the desired sequence. Check if each of the buttons result in the desired result. Develop the java classes for data retrieval. Test each of them according to test cases. Develop java code for timetable generation. Test the coding with a small database by generating a time table.

**8.1 Functional Test Criteria**

The objective of these test is to ensure that each element of the application meets the functional requirement of the user.

• Requirements Catalogue

• Other functional documents produced during the course of the project i.e. resolution to issues/change requests/feedback.

• Validation Testing - which is intensive testing of the new Front end ﬁelds and screens. Windows GUI Standards; valid, invalid and limit data input; screen look and appearance, and overall consistency with the rest of the application.

• Functional testing - these are low-level tests which aim to test the individual processes and data ﬂows.

**8.2 Integartion Testing**

This test proves that all areas of the system interface with each other correctly and

that there are no gaps in the data ﬂow. Final Integration Test proves that system works as integrated unit when all the ﬁxes are complete.

**8.3 User Acceptance Test**

This test, which is planned and executed by the User Representative(s), ensures that the system operates in the manner expected, and any supporting material such as procedures, forms etc. are accurate and suitable for the purpose intended. It is high level testing, ensuring that there are no gaps in functionality.

**8.4 System Test Criteria**

Entrance Criteria

• All developed code must be unit tested. Unit and Link Testing must be completed and signed oﬀ by development team.

• All human resources must be assigned and in place. • All test hardware and environments must be in place, and free for System test use.

Exit Criteia

• All High Priority errors from System Test must be ﬁxed and tested

**8.5 Test cases and Test results**

Outlined below are the main test types that are performed for this release. All test entries on wrong input has been tested to verify code stablilty and correctness. The test cases presented here are based on criterias presented above to validate its test implementation. Each test case table list the detailed test case results for each interface and its user inputs that can be assinged by the user to check the correctness of its implematation.

**CHAPTER 9**

**Appendix**

**9.1 Instalation Manual**

• Install jre

• On Windows double click on TimetableGA.jar

• On Linux run in terminal TimetableGA.sh or type in terminal ”sh path/ TimetableGA.sh”

• To run in shell/Command prompt type java -jar path/ TimetableGA.jar

• Enter necessary informations (descrided in user interface scenarios)

• Get the desired time table as output

**9.2 User interface Scenarios**

User interface Scenarios gives a brief preview of user interface.

**Bibliography**

[1] Automated Class Scheduling Generation in the Context of a University

Timetable Information System, Kuldeep Singh Sandhu.

[2] A Genetic Algorithm Based University Timetabling System, Edmund Burke,

David Elliman and Rupert Weare.

[3] An Evolutionary Algorithm for solving School Time-tabling problem, Calogero

Di Stefano and Andrea G. B. Tettamanzi.

[4] Student Time Table By Using Graph Coloring Algorithm, Baki Koyuncu ,Mah

mut Seir.

[5] Constraint Based Timetabling, A.M. Abbas, E.P.K Tsang.

[6] Generating Complete University Course Timetables by Using Local Search Meth

ods, Duong Tuan Anh, Vo Hoang Tam, and Nguyen Quoc Viet Hung.

[7] OPUS-College Timetable Module Design Document A. Cornelissen, M.J.

Sprengers, B. Mader.

[8] Construction of Timetables Based on Periodic Event Scheduling, Jonas

Christofer Villumsen.

[9] Application to a University Course Timetabling Problem byGeneral Project,

Scheduler Masanori Horio, Atsuo Suzuki.

[10] AUTOMATED COURSE TIMETABLING USING GAM-6, M. Nandani, Dr. S.

Kanmani , Gilbert S., Theepan S. and Venkatesan K.

[11] Dynamic Timetable Generator, Paula Okunieﬀ, Transit IDEA Project.