LECTURE TIMETABLING USING GENETIC ALGORITHM

UNDER

Prof V.N.Murlidhara

IIITB

BY

1. Joshi Shruti Sudesh (MT2011051)
2. Kadam Dipali Anandrao (MT2011058)
3. M Maninya (MT2011072)
4. Mahesh Babu S (MT2011076)

ABSTRACT

Lecture Timetabling Using Genetic Algorithm is the implementation of a computer program which employs Genetic Algorithm (GA) in the quest for an optimal lecture timetable generator.

A timetable is explained as, essentially, a schedule with constraints placed upon it. The constraints can be hard as well as soft, Genetic Algorithm guides to optimize the time tabling problem considering all the constraints and generate the optimal solution to it. Genetic algorithm incorporates various strategies to ensure there is no violation of the constraints.

The essence of the GA lies in the fact that, it works on the underlying principle of "SURVIVAL OF THE FITTEST".

With this fundamental concept, the time table generator removes the weakest contender and breeds the remaining ones till the desired optimum level is reached.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Time table problem has to consider number of constraints and the final outcome has to comply with all the constraints. The constraints vary from institution to institution. The constraints set can contain constraints such as room size, number of available rooms, lecturer availability constraint, number of lectures to be conducted per day, clashes of classes being doubly booked per room, lecturer should not be double booked etc. The time table has to take care of all these things which is a difficult task if the institution is very big.

Till date, the time table management has been done manually which is a cumbersome and tedious effort. Our project aims in making time scheduling problem automated. Lecture Timetabling problem uses GENETIC ALGORITHM to schedule the time-table for IIITB.

CHAPTER 2: BACKGROUND

2.1 OVERVIEW OF GENETIC ALGORITHM :

Our code is based on the following guidelines:

```
Create a population of creatures.
Evaluate the fitness of each creature.
While the population is not fit enough:
{
    Kill all relatively unfit creatures.
    While population size < max size
    {
        Select two population members.
        Combine their genetic material to create a new creature.
        Cause a few random mutations on the new creature.
        Evaluate the new creature and place it in the population.
    }
}.
```

2.2 CONCEPTS OF GENETIC ALGORITHM:

1) Population Size :

The first step in a GA is to initialise an entire population of chromosomes. The size of this population must be chosen. Depending on the available computing techniques, different sizes are optimal. If the population size chosen is too small then there is not enough exploration of the global search space, although convergence is quicker. If the population size is too large then time will be wasted by dealing with more data than is required and convergence times will become considerably larger .

Here the population size refers to the number of the time tables generated.

2) Evaluation of fitness of each creature :

Here the creature refers to the time table. Each time table created is put to rigourous test to figure out the number of constraints it violates. Each constraint has a cost of violation associated with it. Depending upon the number of constraints violated, total cost of violation is calculated. The genetic algorithm aims in minimizing the cost to zero when we have the final time table ready.

3) Breeding and mutation of the creatures :

Depending on the cost (higher the cost, more number of constraints are violated which are to be avoided), half number of the total population size of the time tables is destroyed. Effectively, it means that the unfit populations are destroyed and best ones are chosen for breeding and the child time table is produced.

Child time table is again mutated to introduce disparities and again the best of all the time tables are chosen for breeding and the process continues till the time of the goal of cost minimization is achieved.

4) Repair Strategies :

Two timetables, chosen as parents, might each have only one instance of each class. Yet, if these parents are not identical, then a child produced by their crossover might have multiple bookings of some classes. Such a timetable might be outside the search space of the problem in which case a repair strategy could be used to remap the chromosome to within the search space. For our project, the repair strategy would alter the child’s genes to ensure that exactly two booking of each class is made in a week.

Also, another aspect of repair strategy is that the subjects which weren't included at all are alloted slots in the schedule. This scheme is also used initially for initialization of the time table and then to allocate schedules to the subjects which weren't even once included in the timetable.

CHAPTER 3: METHOD

3.1 INPUT FORM :

The input file contains records for all the students, the subject they have chosen, the faculty name and the subject description, the faculty id and the course id. The csv file having these records is given to the program, which parses the file and all the records are collected in the structure.

This structure forms the master data for the program from where the desired data is taken for the specific function dealing with the time table.

```
struct record
{
	char rollno[10];
	char course_id[20];
	char course_name[100];
	char faculty_name[100];
	int faculty_id;
}record_arr[MAX_NO_OF_RECORDS];

/* STRUCTURE USED TO STORE THE LIST OF DISTINCT FACULTY MEMBERS AND
THEIR DETAILS IN AN ARRAY */

struct faculty
{
	char faculty_name[100];
	int availability_time_table[DAYS_IN_WEEK][HOURS_IN_DAY];
	int faculty_id;
	int days_available;
}faculty_arr[MAX_NO_OF_FACULTY];
```

```
/* STRUCTURE USED TO STORE THE LIST OF DISTINCT STUDENTS AND THEIR
DETAILS IN AN ARRAY */

struct student
{
    char roll_no[10];
    int courses[20];
    int index;
}student_arr[MAX_NO_OF_RECORDS];


/* STRUCTURE USED TO STORE THE LIST OF DISTINCT COURSES AND THEIR
DETAILS IN AN ARRAY */

struct course
{
        char course_id[20];
        char course_name[100];
        int class_size;
        int faculty_id;
        int faculty_reference;
        int course_preference[DAYS_IN_WEEK][HOURS_IN_DAY];
}course_arr[MAX_NO_OF_COURSES];

/*STRUCTURE USED TO STORE THE LIST OF DISTINCT CLASS-ROOMS AND THEIR
DETAILS IN AN ARRAY*/
struct room
{
        int room_number;
        int capacity_of_room;
}room_arr[NUMBER_OF_ROOMS];
```

```
/*STRUCTURE USED TO STORE THE DETAILS OF A 3D TIME-TABLE */
struct time_table
{
	struct time_table *next;
	int table[NUMBER_OF_ROOMS][DAYS_IN_WEEK][HOURS_IN_DAY];
	int cost;
	int class_clash_error;
	int room_small_error;
	int faculty_double_booked_error;
	int faculty_unavailable_error;
	int class_twice_error;
	int course_preference_error;
	//int rooms_different_error;
};
/*STRUCTURE USED TO STORE A COLLECTION OF 3D TIME-TABLES IN THE
POPULATION */
struct colony
{
	struct time_table *first_time_table;
	struct time_table *last_time_table;
	int population_size;
	int average_cost;
	int class_clash_error;
	int room_small_error;
	int faculty_double_booked_error;
	int faculty_unavailable_error;
	int class_twice_error;
	int course_preference_error;
	//int rooms_different_error;
}solution_colony;
```

3.2 REPAIR STRATEGY :

A repair strategy is used which ensures that all classes appear exactly twice. For robustness, this is done in two stages. Firstly, any classes which appear more than twice are altered such that they appear only twice .

```
For each course:
    set the Count to 0.
    For each time:
    For each room:
        If the current course is booked at this location:
        Add 1 to the Count.
        Add the location of this class to a linked list.
    If the class occurred more than twice then:
        keep doing the following until there is only two bookings left:
            randomly choose one of the bookings.
            turn it into a NULL booking.
        Free the linked list.
```

Secondly, any classes which did not appear at all are booked to a spare space (regardless of room size, etc).

```
For each course:
    For each time:
        Look in each room until either the class is seen or you get to the end.
    If you got to the end without finding the class then randomly find a NULL booking and
    book that course to it.
```

If this repair strategy is applied to an empty timetable the result is a timetable with each course booked to a random time and place. As such, the repair strategy is also used for initializing a random population.

3.3 MULTIPLE CONSTRAINTS TO EVALUATE A TIMETABLE :

We have considered only hard constraints for the present problem.

These constraints are:

- Class clash errors.
- Room too small errors.
- Faculty double booked errors.
- Faculty unavailable errors.
- Class twice errors.
- Course preference errors.

```
For the timetable being evaluated:
    Initialise the cost field to zero.
    For each of the hard constraints:
        Record how many times that constraint is violated.
        Add the count to the timetable's cost field.
```

Class clash errors:

This calculates how many times any student has two or more classes at the same time.

```
For each time:
    For each room:
        Does the course booked at this time related to the courses at same time and day?
        If so then add "1" to the Count.
Record (Count) "Related class clash" errors.
```

Room too small errors:

This calculates how many classes in a timetable are booked to rooms which are too small to accommodate them.

```
For each room we must check:
    For each time:
        For each course, given that the room capacity
            Is the size of the class bigger than room capacity?
            If so then add "1" to the Count.
Record (Count) "Room too small" errors.
```

Lecturer double booked errors :

A count is performed to see how often lecturers are expected to be in two (or more) lectures at once.

```
For each lecturer we must check:
    For each time:
        For each class:
            Is this class taught by the current lecturer?
            If so then add "1" to the Count.
If (Count>1)
    Record (Count-1) "Lecturer double booked" errors.
```

Lecturer unavailable errors :

As well as lecturers having more than one lecture at a time, there is the possibility that they will have lectures at times when they have prior commitments. To penalise such cases a test is performed which counts all instances when they are booked to give a lecture at a time when they have indicated that they have prior commitments.

```
For each lecturer we must check:
    For each time, given that at that time the Lecturer's "Availability chart" shows
    either a 1 (available) or 0 (unavailable):
        For each room:
            Is the lecturer of that class the current lecturer?
            If so then if the lecturer is unavailable add "1" to the Count.
Record (Count)  "Lecturer Unavailable" errors.
```

Class twice errors :

This calculates the number of times a course is scheduled twice in a same day, but if the faculty is available on only one day or if the course preference is given on only one day, then this is not an error.

Course preference errors :

This calculates the number of times the obtained time-table violates the course preference.

3.4 SELECTING TIMETABLES FOR EXTINCTION :

The timetables are kept in an ordered linked list this is particularly easy to implement. In each generation a fixed portion of the timetables are eradicated. For the purposes of this thesis the portion was maintained at 50%.

Work out how many timetables are going to survive.

Sort through the list until you find the last one to survive.

Assign that one as now being last in the population.

Systematically free each of the remaining timetables from the memory.

3.5 BREEDING :

Timetables are randomly selected from the population and used for breeding.

A child timetable is bred by performing unity order based crossover on the parents.

This means that each parent has an equal chance of providing each gene.

3.6 MUTATION :

The method of mutation is given. This means that the chance of any one gene undergoing mutation is approximately twice the mutation rate divided by one thousand. A mutation rate equal to ten, for example, implies that approximately twenty in every thousand genes will be mutated.

There is a fixed mutation rate.

```
For each gene
{
        Randomly choose a number between 1 and 1000.
        If the number is less than the mutation rate then
        {
                Randomly choose a gene from the current timetable
                and swap it with the current gene.
        }.
}.
```

3.7 OUTPUT FORMAT :

The program first parses the file and displays the following:
Total number of records
Total number of faculty
Total number of courses (distinct)

Then it displays the flow of operation of the genetic algorithm:

It begins with a population of 3 random time-tables and displays their individual and average costs:

```
TT1: 52  TT2: 56  TT3: 58  Average cost = 55
```

Then through progressive breeding and mutation it tries to reduce the cost of the population of time-tables, continuing till one of the time table's cost becomes 0. This is also displayed.
The total number of time-tables generated during this is also calculated and displayed.
The schedule for each of the courses is displayed next followed by the actual time-table.

The program generates a Time-table that includes all the semesters. The time-slots per day have been labelled as A, B, C, D and so on. The courses scheduled for each day of the week (monday to saturday) are shown in each of the cells of the time-table, along with the room number and faculty ID. A table mapping the faculty IDs to the faculty names is also shown.

REFERENCES:

1.Lecture Timetabling Using Genetic Algorithms , the thesis submitted by Author Leon Bambrick and Supervisor Dr B Lovell from Department of Electrical and Computer Engineering, The University of Queensland.