SSN COLLEGE OF ENGINEERING

KALAVAKKAM -603110

Department of Computer Science and Engineering

UCS-2201 FUNDAMENTALS AND PRACTICES OF SOFTWARE DEVELOPMENT

PROJECT DOCUMENTATION FOR REVIEW 2

Exam Time Tabling in a University

Team name: D3VA

TEAM MEMBERS

1. V. A. Balaji	-	3122 21 5001 018
2. R Darshan	-	3122 21 5001 021
3. D. Aswanth Ram	-	3122 21 5001 013

STAFF GUIDE

Dr Kanchana R

B.E.(CSE), M.E.(SE), Ph. D

Associate Professor

PROBLEM STATEMENT

Project : Exam Time Tabling in a University

The time table creation is a tedious process, since so many constraints to be considered before make it. There is a need for an automatic system. To create a exam schedule for University examination for an academic institution by considering possible constraints.

Sample Constraints:

- Students should not be required to write two exams at the same time.
- The examinations must be well spaced over the timetable .
- Examinations with a large number of students must be scheduled early in the examination timetable. • Some examinations may have to be scheduled simultaneously or after other exams ..
- The number of students writing examinations in a particular venue and session must not exceed the capacity of the venue.
- Each student has a schedule of exams he/she must attend • Each room has multiple features, such room size .
- Each exam has a set of required features including the number of students attending it .
- Rooms can only accommodate exams if they have enough seats to fit all the attending students and have at least all the features required for the exam.

ABSTRACT

Scheduling problem is defined in many areas and it's hard to solve because it includes many constraints that should be solved. Final university examination scheduling is one type of Scheduling problem, and it's hard to be solved because each university may include hard and soft constraints that are different for each case. So this is our attempt to solve one of the scheduling problems by considering the soft and hard constraints. This could be very useful to minimize human error and save time.

INTRODUCTION

In this project we will solve the final exam scheduling problem for our esteemed SSN institution , and we will develop the system, make experiment for the problem to generate final exam scheduling timetable and compare it with other systems if existed. In the following pages we will describe the existing problems for this topic and how far this solution would help to solve the problem and then define the hard constraints and soft constraints, then describe the problem, and finally our solution for the problem and the experiment result.

EXISTING PROBLEM

In previous days, the exam time table is formed manually by checking the hard constraints. In traditional methods, only hard constraints are attempted to be satisfied with lot of manual works. The soft constraints are not taken into account. Refining and Development of the generated exam time table is not possible. It consumes more time and human effort. It will result in high occurrence of error. Due to presence of manual work, wastage of paper cannot be avoided. Soft constraints are left unsatisfied. There fore in many ways, the traditional method of preparing exam time table will not produce 100% accurate results with constraints satisfied.

CONSTRAINTS

Constraints are the limitations or restrictions to check while executing the problem statement. There are 2 types of constraints , Soft constraints and Hard constraints.

Hard Constraints

Hard constraints are **constraints that are absolutely non-negotiable**. The scheduling engine will always respect every hard constraint that you give it. When you specify a hard constraint, you're effectively telling the scheduler that "If I can't have this constraint met, then I don't want any schedule at all."

Hard Constraints handled in this problem:

- Minimum of 2 – 3 days of gap between every exam.
- Exams should not be scheduled on the input holidays
- Consecutive days should not be scheduled for exams.
- Exam dates of one semester must not overlap with other semester exam dates.

Soft constraints

A "SOFT" constraint is a "WANT" to be satisfied as much as possible if the cost for doing so is not too great. Soft constraints allow us to be a bit more flexible in our decision making.

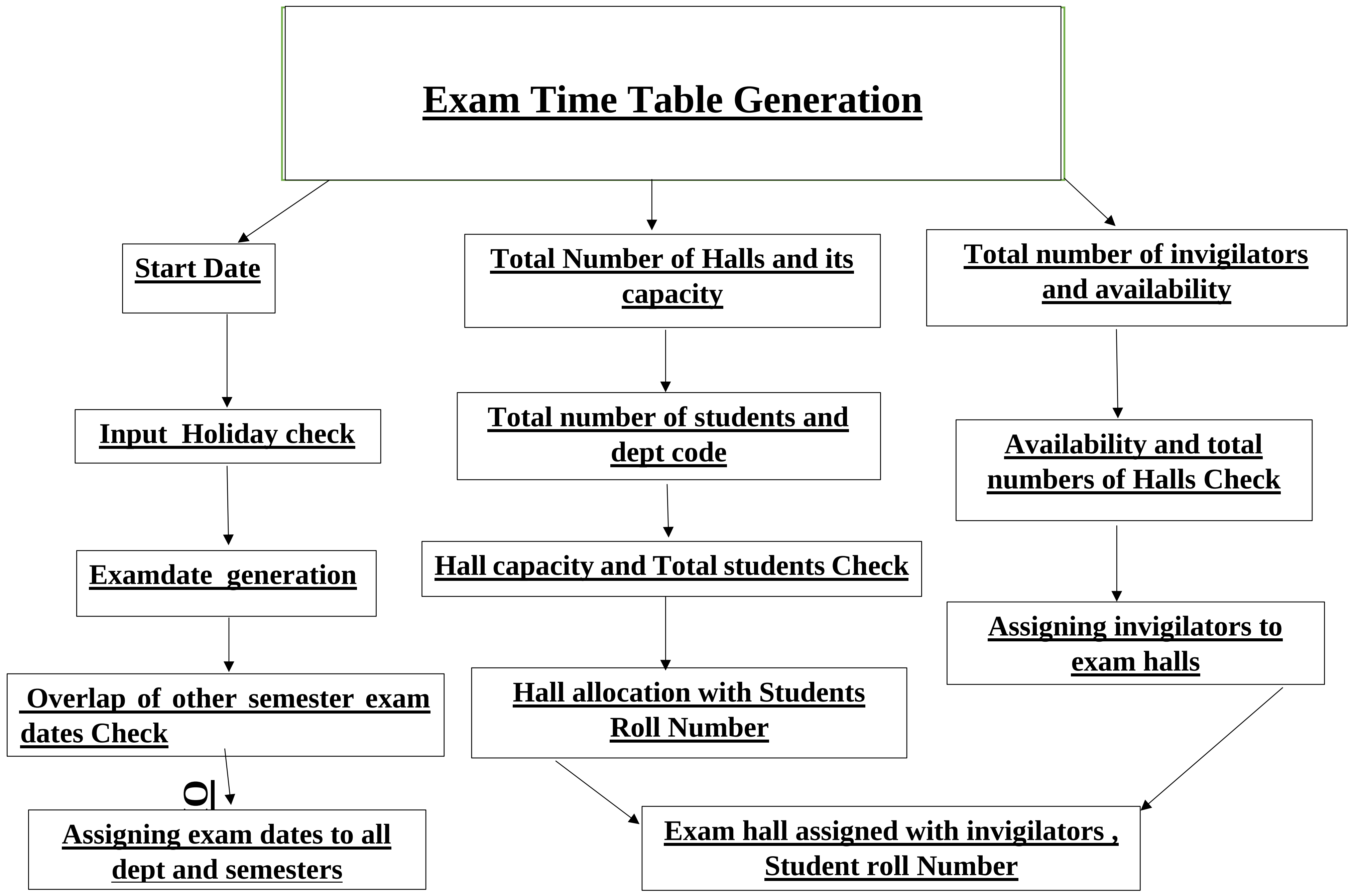
Soft constraints handled in this problem

- Availability of invigilator is checked before assigning them to the respective exam halls.

- Availability of exam halls is checked by comparing the total number of students with sum of capacity of all exam halls.

- If there are insufficient number of exam halls , requirement of extra exam halls is needed to assign for all students.

- If the number of halls exceeds the number of invigilators, there are insufficient number of invigilators. Requirements of additional invigilators is mandatory here.


Exam Time Table Generation
Start Date
Input Holiday check
Examdate generation
Overlap of other semester exam dates Check
Assigning exam dates to all dept and semesters
Total Number of Halls and its capacity
Total number of students and dept code
Hall capacity and Total students Check
Hall allocation with Students Roll Number
Exam hall assigned with invigilators , Student roll Number
Total number of invigilators and availability
Availability and total numbers of Halls Check
Assigning invigilators to exam halls

MODULE DESIGN

MODULE 1:

**prepareSchedule(struct date st, struct course CS[ ], int n,
struct schedule S[ ], int day, struct date holis[ ])**

INPUT:

Start date , List of holidays

OUTPUT:

Exam dates for each semesters

MODULE 2:

**scoreandsort(char cse[][25], char it[][25], char ece[][25], char eee[][25],
char civil[][25], char mech[][25])**

INPUT:

Unsorted 2D Array of all departments with respective subjects

OUTPUT:

Sorted 2D Array of all department based on their occurrence

(subjects which have high occurrence will be assigned at first and other subjects will be assigned in the descending order of the occurrence)

MODULE 3:

printSchedule(char deptlist[][10], char cse[][25], char it[][25],
char ece[][25], char eee[][25], char civil[][25],
char mech[][25], struct schedule *S, int sem, FILE *fp)

OUTPUT:

Exam dates are assigned with students Roll number and respective Invigilator for both forenoon and afternoon sessions.

MODULE 4:

holidaycheck(struct date stdate, struct date holidays[10], int N)

INPUT:
List of holidays , Start date

OUTPUT:
List of Exam dates without holiday dates.

MODULE 5:

leapyear(struct date st)

INPUT:

Year of the exam

OUTPUT:

Year is checked for leap year and dates of February month is incremented accordingly.

MODULE 6:

allocateHalls(struct room halls[], int nh, struct class classes[], int nc, struct hallAllocation Halls[])

INPUT:

Total number of halls, halls capacity, total number of students, roll number of respective department.

OUTPUT:

Halls are allocated for students with maximum utilization of hall capacity .

MODULE 7:

printhalls(struct room halls[30], FILE *fp1, struct class classes[],FILE *fpd, struct Invigilator *inv, FILE *fph1, struct hallAllocation Halls[30], FILE *fpi, int sem, struct schedule S[10])

INPUT:

Assigned halls for students, List of Invigilators

OUTPUT:

Invigilators are assigned to Exam halls by checking their availability.

MODULE 8:

getInvigilators(struct Invigilator *inv, int ni, FILE *fpi)

INPUT:

Invigilators name , Availability.

OUTPUT:

Null

MODULE 9:

getHalls(struct room halls[], int nh, FILE *fp)

INPUT:

Total number of halls , Hall name , Hall Capacity.

OUTPUT:

Null

MODULE 10:

getClasses(struct class classes[], int nc, FILE *fpd)

INPUT:

Total number of students , dept code , Roll Numbers

OUTPUT:

Null

MODULE 11:

checkinvi(struct Invigilator *inv, int t, FILE *fph,
struct hallAllocation Halls[], int sem)

INPUT :

List of invigilators , Availbility

OUTPUT:

List of available invigilators

DATA STRUCTURE

1) struct date

```
        int d;
        int m;
        int year;
```

2) struct course

```
  char courseCode[10];
  char courseName[20];
  char type[7];
  char session[7];
```

3) struct schedule

```
  struct date dt;
  char day[10];
  char courseCode[10];
  char courseName[20];
  char type[7];
  char session[7];
```

4) struct room

```
  char roomNo[5];
  int roomCapacity;
  int status;
  int remainingSeats;
```

5) struct Invigilator

```
  char facultyName[20];
  int avail;
```

6) struct student

```
  int rno;
  int regNo;
  char name[20];
```

7) struct class

```
  char dept[10];
  int deptCode;
  int noOfStudents;
  int regNos[60];
```

8) struct hallAllocation

```
  char hallNo[5];
  int regNoAssigned[30]
  int currentCnt;
```

Reference Materials:

Final Exam Scheduling Timetable a Case Study. Available from:

https://www.researchgate.net/publication/273521527_Final_Exam_Scheduling_Timetable_a_Case_Study .

Komijan, A.R., Koupaei, M.N. A new binary model for university examination timetabling: a case study. J Ind Eng Int 8, 28 (2012). https://doi.org/10.1186/2251-712X-8-28

Qu R, Burke E, McCollum B, Merlot LTG, Lee SY (2009) A Survey of Search Methodologies and Automated System Development for Examination Timetabling. Journal of Scheduling 12:55–89, DOI 10.1007/s10951-008-0077-5