A System for Efficient Exam Seat Allocation

Charitha Tuniki[1], Vanitha Kunta[1], M. Trupthi[1]

[1] Chaitanya Bharathi Institute of Technology, Hyderabad, India.
charithathuniki@gmail.com, vanithakunta@gmail.com, mtrupthi_it@cbit.ac.in

Abstract: Examination seating arrangement is a major issue faced by an institution. A university consists of a large number of students and classrooms, which makes it difficult for the university authorities to design seating arrangement manually. The proposed solution provides an efficient set of algorithms for examination seating allocation problems. It also gives the best combination of rooms to be utilized for the exam and dynamically organizes seating based on the orientation of the room and students. The described solution encompasses methods to address some common issues like eliminating students who are ineligible to write one or more exams and adding the students who are retaking the exams. The presented system is made examiner-friendly such that the user can swiftly get a perfect seating arrangement based upon the above cases without manually excluding the ineligible students and rearranging the system. Excel sheets of the classroom view are generated automatically along with the room number and capacity of the classroom. These ready-to-go sheets can be printed and used by the examiners.

Keywords: exam seat allocation, room optimization, dynamic, algorithms, ready-to-go sheets.

1 Introduction

Exam hall seating arrangement is an important strategy for avoiding malpractice and conducting a fair examination. University authorities find it difficult to allocate students into exam halls due to high number of courses and departments. In the worst case a student may even sit adjacent to other student belonging to same class even if there is place to arrange them away from each other. These kind of seating arrangements may cause trouble to the invigilators and also the university management.

The main criteria for examination seating arrangement is to properly arrange the students in an efficient and costly friendly way with minimum utilization of resources. For this, the collection of appropriate data such as total number of courses, students, rooms availability, room orientation etc. is needed. The aim of the proposed system is to utilize all the available rooms optimally and allocate the students into rooms in a structured and systematized manner. The room optimization algorithm described later provides a method to utilize the available rooms to the maximum by eliminating the extra rooms. Further, taking the utilized rooms into consideration, the students are seated according to their IDs such that students belonging to the same course do not sit in the each other's vicinity.

The excel sheets representing the exam hall generated for each room provide a clear view of the room number, capacity of the room and allotment of each student represented by his ID number. The examination authorities can easily know the capacity of each room and thus make use of it to assign invigilators and know the number of exam scripts to be issued per classroom.

2 Related Work

Exam hall seating arrangement has widely been researched for a number of years. The methodology proposed by Aashti Fatima Alam[1] considered each room to be of equal seating capacity and Prosanta Kumar Chaki et al. [2] proposed a system where students belonging to the same course were allowed to sit in the nearest row, but not in the nearest column. R.Gokila et al.[3] put forward a GUI implementation which showcases the range of student IDs allotted to each class and generates a report to the admin. Room Penalty Cost Model is formulated by Ariff Malik et al.[4] to minimize the students shifting between the rooms allotted to them when there are continuous exams in a single day.

PHP and Java based application is created by Sowmiya. S et al. [5] where the students are arranged in seats behind each other, considering the last digits of their roll numbers. S.PriyaDharshini et al.[6] suggested a GUI where the students can register and locate their classroom for examination. This research manually updates the examination hall information prior releasing it to the students. The research done by Dinesh Chandewaret al.[7] arranges students behind each other using a dynamic array. A software is created by T. Prabnarong et al. [8] which allows the examination invigilator to choose their scheduled time and based on the room type and capacity, each subject is assigned to each available room. Constraints regarding the distance between rooms and dividing the courses across several rooms is addressed by M.N.M. Kahar et al.[9] . Graphs and networks are used by de Werra D[10] to provide arranging of courses and examinations in a university.

3 Existing System

The existing manual system is slow and takes a lot of human effort. The rate of errors made by a human are higher than those made by a computer. The criteria where the work is split among many people to decrease workload creates a havoc.

The traditional way of examination hall seating arrangement involves collection of huge data about the students and rooms and processing it manually to arrange students in the classroom.

The limitations of existing system are as follows:

- May lead to missing of few students, allotting a student multiple times etc.
- Students ineligible for examination are removed and seating rearrangement is done manually.
- Less efficiency.
- Hectic work and time taking.

4 Proposed Method

Our research aims to aid the management of the universities to solve the problem of exam seating plan. It is sensitive and efficient enough to work even with small number of courses and less rooms. The students and classrooms data is collected and is operated on with the algorithms to create an efficient seating arrangement. This system is built to be effective and scalable. The reference model that is used in the proposed method is shown in Fig. 1.

5001	8007
7005	6003
5002	8008
7006	6004
5003	8009
7007	6005

5004	8010
7008	6006
5005	9001
7009	6007
5006	9002
7010	6008

5007	9003
8001	6009
5008	9004
8002	6010
5009	9005
8003	7001

5010	9006
8004	7002
6001	9007
8005	7003
6002	9008
8006	7004

Fig. 1. Reference Model

5 Methodology

A. System Description

In this research, the algorithms are designed for three phases. They are as follows:

Phase I: The first phase deals with collection of input data such as total number of rooms available, the room orientation and total number of students attending the examination. In this phase, We mark the rooms that are to be used for examination, hence eliminating the extra rooms.

Phase II: The second phase consists of two algorithms. In first algorithm, a set of students of each course are allocated to a particular room. That is, each room is filled with a batch of students who belong to different courses. In the second algorithm of this phase, we allocate the ID numbers of the students to each room, with reference to the first algorithm in this phase.

Phase III: In the third phase, we place the students according to their ID numbers in the room. This arrangement is done by the algorithm in an efficient manner such that no two students writing exam of same course are placed in the proximity of each other.

In Fig.2. , the phases of the discussed methodology are represented using a flow chart.

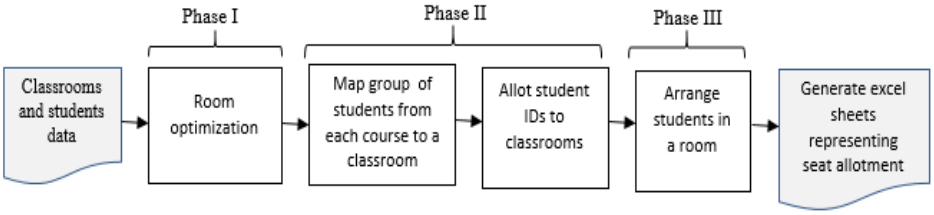


Fig.2. Flow Chart depicting the phases of proposed methodology

B. Algorithms

Phase I: Algorithm 1.1

In this algorithm we mark rooms for utilization using their capacities and the number of students attending the examination. Extra seats are calculated using the total capacity of rooms and total students. These extra seats are further used in the algorithm to choose the optimal rooms for seating arrangement.

Parameters used in algorithm:

roomCount= total number of rooms

roomCapacity[roomCount]= list of capacities of each room in sorted order

totalCapacity= $\sum_{i=0}^{roomCount}$ roomCapacity[i]

totalStudents= total number of students attending the exam
utilizedRooms[]= list containing the indices of rooms that should be used for exam
ignoredRooms[]= list containing the indices of rooms that should be ignored for exam
The pseudo code for the Algorithm 1.1 is given in Fig. 3.

Inputs: roomCapacity[], roomCount, totalCapacity, totalStudents
Output: utilizedRooms[], ignoredRooms[]

1. Initialize extraSeats to the difference of totalCapacity and totalStudents
2. Declare utilizedRooms, ignoredRooms are empty list
3. Initialize roomIndex to one less than roomCount
4. Function RoomOptimization(extraSeats, roomIndex, ignoredRooms)
   a. While extraSeats is less than roomCapacity[roomIndex] and roomIndex is greater than or equal to zero
      i. Decrement roomIndex by one
      ii. If roomIndex is less than zero
          Return
      End If
      End While
   b. Add roomIndex in ignoredRooms
   c. Deduct roomCapacity[roomIndex] from extraSeats
   d. If extraSeats is greater than or equal to roomCapacity[0]
      i. Decrement roomIndex by one
      ii. If roomIndex is greater than or equal to zero
          value = RoomOptimization(extraSeats, roomIndex, ignoredRooms)
      End If
      End If
   e. Set outValue to ignoredRooms
   f. Return outValue
5. If ignoredRooms is not empty
   a. For every room
      i. If a room is not in ignoredRooms
         Add that room in utilizedRooms
      End If
   End For
   End If
6. Else
   Add all rooms in utilizedRooms

Fig. 3. Pseudo code of Algorithm 1.1

Phase II: Algorithm 2.1

In this algorithm, we calculate the number of students from each course to be accommodated in a particular room. This is calculated by the formula 1 given below:

Students from a course in a room=

$$\frac{Total\ capacity\ of\ students\ from\ the\ course}{Total\ number\ of\ utilized\ rooms} \quad (1)$$

The pseudo code for the Algorithm 2.1 is given in Fig. 4.

```
Inputs: utilizedRooms[], coursesCapacity[]
Output: roomCoursesCapacity[[]], roomCourseIDs[[]]
1. For each room in utilizedRooms
    a. For each course in the list of Courses
        i. Set j to coursesCapacity[course] // total utilizedRooms
        ii. If utilizedRoomsCapacity[room] is greater than or equal to j
                Add j to roomCoursesCapacity[room]
                Subtract j from utilizedRoomsCapacity[room]
                        End If
        iii. Else
                If utilizedRoomsCapacity[room] is greater than or equal to 0
                    Add utilizedRoomsCapacity[room] to
                    roomCoursesCapacity[room]
                    Set utilizedRoomsCapacity[room] to 0
                                End If
                Else
                    Add 0 to roomCoursesCapacity[room]
2. For each room in utilizedRooms
    a. If sum of roomCoursesCapacity in a room is greater than or equal to
       utilizedRoomsCapacity of that room
        i. Add 0 to roomEmptySeats of that room
    b. Else
        i. Find difference between (utilizedRoomsCapacity of a room and sum
       of roomCoursesCapacity in a room)
        ii. Add calculated difference to roomEmptySeats of that room
3. For each course in the list of Courses
    a. If coursesCapacity of a course is equal to the sum of that course
     capacity in all the rooms assigned to that course
        i. Add 0 to courseCapacityNotFilled of that room
    b. Else
        i. Find difference between (coursesCapacity of a course and sum of that
       course capacity in all the rooms assigned to that course)
        ii. Add calculated difference to courseCapacityNotFilled of that room
4. For each course in the list of Courses
        a. If courseCapacityNotFilled of course is not equal to 0
            i. For each room in utilizedRooms
                i.a. If roomEmptySeats of room is not equal to 0
                    If roomEmptySeats of room is greater than or equal
                    to courseCapacityNotFilled of course
                        Subtract courseCapacityNotFilled of course
                        from roomEmptySeats of room
                        Add courseCapacityNotFilled of course to
                        roomCoursesCapacity of room
                        Set courseCapacityNotFilled of course to 0
                    Else
                        Subtract roomEmptySeats of room from
                        courseCapacityNotFilled of course
                        Add roomEmptySeats of room to
                        roomCoursesCapacity of room
                        Set roomEmptySeats of room to 0
                i.b. If courseCapacityNotFilled of course is equal to 0
                        break
```

Fig. 4. Pseudo code of Algorithm 2.1

List of parameters for both algorithm 2.1 and calculation:
coursesCapacity[] = capacity of students in each course.
utilizedRoomsCapacity[] = capacity of utilized rooms in sorted order.
roomCoursesCapacity[[]] = number of students from each course in a particular room.
courseCapacityNotFilled[]=number of students of a course who are not allocated.
roomEmptySeats[]=number of empty seats in each room.
Description:
Step 1:
If the capacity of a particular room is sufficient enough to accommodate a group of students belonging to a course (calculated using formula 1), then those students are mapped to that classroom. Else, if the capacity is not sufficient enough, then the appropriate number of students are accommodated and rest are directed to next room.
Step 2:
We calculate the number of empty seats in each room
to further reference it in Step 4.
Step 3:
We calculate the number of students who are not accommodated into any of the rooms. It will be further referenced in Step 4.
Step 4:
The students who are not yet accommodated (calculated in Step 3) are assigned to the classrooms having empty seats (calculated in Step 2). The algorithm terminates if all the students from each course are assigned to classrooms.

Algorithm 2.2

This algorithm is designed to allot student IDs of each course to a room.
List of parameters for both algorithm and calculation:
studentIDs[]= student IDs of each course. It is computed with reference to starting and ending IDs including the IDs to be added and excluding the IDs of ineligible students.
roomCourseIDs[[]] = student IDs of each course who are accommodated in each room.
Description:
The capacity of students from a course in each room is considered and the IDs of the students are assigned to each classroom.
The pseudo code for the Algorithm 2.2 is given in Fig. 5.

```
Inputs: roomCoursesCapacity[[]], studentIDs[[]]
Output: roomCourseIDs[[]]
1.      For each room in utilizedRooms
   a.   For each course in the list(Courses)
          i.    Set k to roomCoursesCapacity[room][course]
          ii.   For n =1 to k
                  Add studentIDs[course][n] to roomCourseIDs[room]
                        End For
          iii.  While k is greater than 0
                  remove studentIDs[course][0] from studentIDs[course]
                  Decrement k by 1
                        End While
                End For
        End For
```

Fig. 5. Pseudo code of Algorithm 2.2

Phase III:
This phase uses the Xlsx Writer available in python. Xlsx writer helps to visually represent the student's IDs in an excel sheet and creates an easy view of seat allocation to the invigilator.
Description: The students are arranged as shown in the process diagram Fig. 6. We assume that each bench can accommodate two students.
Step 1: Each student is allocated in an alternative manner along the left side of the bench, starting from first row in the first column, as represented by (1) in Fig.6.
Step 2: At the last column, after the filling of last possible row (6) on left side of bench, the next students are placed starting from right side of the bench in the second row of first column (7).
Step 3: In the last column, after the filling of last possible row (12) on the right side of bench, the next students are placed starting from the left side of bench in the second row of first column (13) until the last possible row is filled.
Step 4: After the completion of Step 3, the students are filled starting from the right side of the bench in the first row of second column.

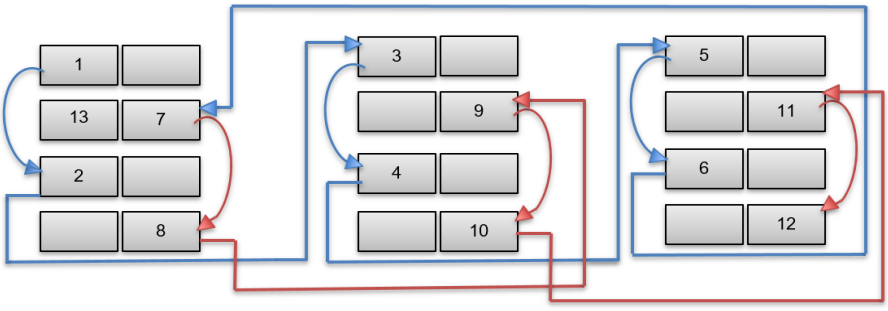


Fig. 6. Process diagram of Phase III

6 Results

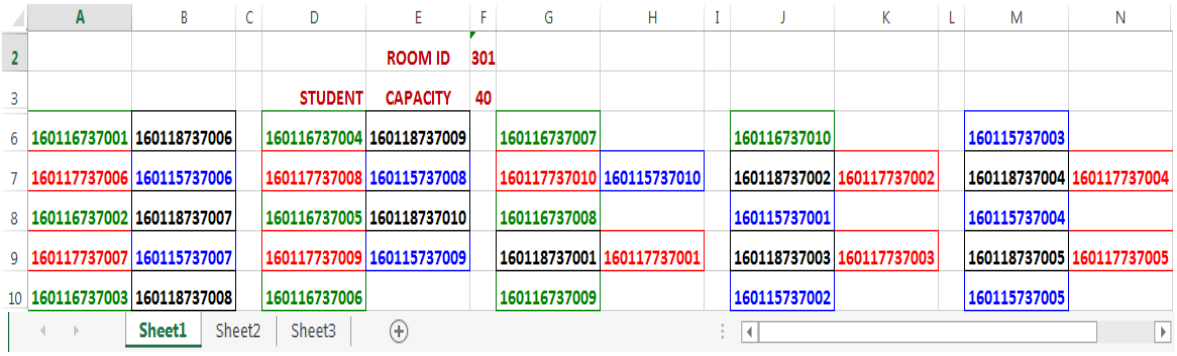
Table 1 depicts the input data about the classrooms and the students' courses.

Table 1. Input data

SNo	Rooms input			Students input					
	Rows	Columns	Room ID	Course Name	Course Capac-ity	Starting Student's ID	Ending Student's ID	Any ID to be deleted?(Y/N)	Any ID to be added?(Y/N)
1	5	5	301	C	30	160116737001	160116737030	Y,160116737025	Y,160116737032
2	5	5	302	C++	30	160115737001	160115737030	N	N
3	5	5	303	Python	30	160117737001	160117737030	N	N
4	-	-	-	Java	30	160118737001	160118737030	N	N

Three rooms with IDs 301, 302, 303 are considered. Each room has 5 rows and 5 columns. The capacity of each room is 50. Four different courses are considered with each course having a student capacity of 30.

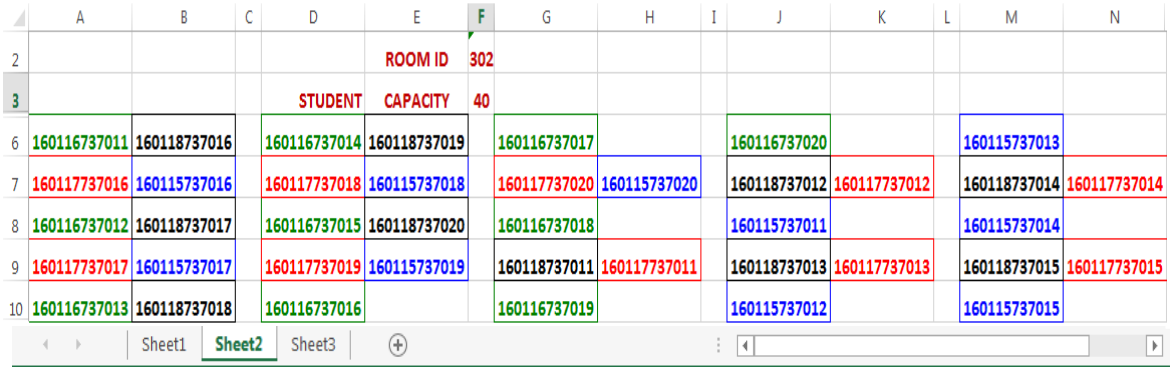
i. Initially, the extra seats are calculated, which is difference between the total capacity available and total number of students.
ii. Algorithm 1.1 is run to optimize rooms utilization. The extra rooms are further not taken into consideration.
iii. The number of students from a course to be alloted in a room is calculated using the formula (1). Thus 10 members from each course are allocated to every room.
iv. We recheck if all students are allocated in the rooms using Algorithm 2.1.
v. The students IDs of each course is allocated to rooms using Algorithm 2.2.
vi. The allocation of seats takes place according to process diagram of Phase III described in Fig.6.
vii. The student IDs are color coded, with each color representing a course. In the displayed result, we represent the four courses using four different colors.
viii. Excel sheets are generated separately for each classroom in a single excel workbook. Each sheet consists of the classroom ID and the total students capacity in the room. Fig. 7, 8 shows the results obtained in the excel sheets.
ix. The third room-303 would be arranged similar to the rooms 301,302 as shown in Fig.7, 8.



2				ROOM ID	301								
3			STUDENT	CAPACITY	40								
6	160116737001	160118737006	160116737004	160118737009		160116737007			160116737010			160115737003	
7	160117737006	160115737006	160117737008	160115737008		160117737010	160115737010		160118737002	160117737002		160118737004	160117737004
8	160116737002	160118737007	160116737005	160118737010		160116737008			160115737001			160115737004	
9	160117737007	160115737007	160117737009	160115737009		160118737001	160117737001		160118737003	160117737003		160118737005	160117737005
10	160116737003	160118737008	160116737006			160116737009			160115737002			160115737005	

Sheet1 Sheet2 Sheet3

Fig. 7. Sheet-1 representing classroom 301



2				ROOM ID	302								
3			STUDENT	CAPACITY	40								
6	160116737011	160118737016	160116737014	160118737019		160116737017			160116737020			160115737013	
7	160117737016	160115737016	160117737018	160115737018		160117737020	160115737020		160118737012	160117737012		160118737014	160117737014
8	160116737012	160118737017	160116737015	160118737020		160116737018			160115737011			160115737014	
9	160117737017	160115737017	160117737019	160115737019		160118737011	160117737011		160118737013	160117737013		160118737015	160117737015
10	160116737013	160118737018	160116737016			160116737019			160115737012			160115737015	

Sheet1 Sheet2 Sheet3

Fig. 8. Sheet-2 representing classroom 302

7 Conclusion

This system has overcome the disadvantages of existing system. It has the advantages of using less human effort, time, reduced rate of errors, user friendly. The system has been tested on various constraints in which students are removed or extra students are added. Each room's capacity and total students capacity is taken into consideration and extra rooms are eliminated efficiently. The results displayed in excel sheets can be easily interpreted by anyone.

The proposed system can be further enhanced by adding a few more constraints like seat arrangement for session based examinations. This project can also be implemented as a mobile application which would be student friendly so that students can know which room they are allocated in through this application. The hall tickets can be generated automatically with the seat allocated to the student printed on it. These algorithms can be improved to be more scalable and robust. Apart from limiting the software to examinations, this could be useful to arrange members attending a conference or a workshop and allot beds to the patients in a hospital.

8 References

[1] Aashti Fatima Alam, "Automatic Seating Arrangement Tool For Examinations In Universities/Colleges", IJEAST, 2016.

[2] Prosanta Kumar Chaki, Shikha Anirban, "Algorithm For Efficient Seating Plan For Centralized Exam System", ICCTICT, 2016.

[3] R.Gokila, Antony Rohan Dass, "Examination Hall and Seating Arrangement Application using PHP", IJESC, 2018.

[4] Masri Ayob and Ariff Malik, "A New Model for an Examination-Room Assignment Problem", IJCSNS, October 2011.

[5] Sowmiya. S, Sivakumar. V, Kalaimathi. M, Kavitha. S. V, "Automation of exam hall seating arrangement", IJARIIT, 2018.

[6] S.PriyaDharshini, M.SelvaSudha, Mrs.V.Anithalakshmi, "Exam Cell Automation System", IJESC, 2017.

[7] Dinesh Chandewar, Mainak Saha, Pushpraj Deshkar, Pankaj Wankhede, Prof. Suwarna Hajare, "Automatic Seating Arrangement of University Exam", IJSTE, March 2017.

[8] T. Prabnarong and S. Vasupongayya, "Examination management system: room assignment and seating layout," Proceeding of the Office of Academic Resources International Conference.

[9] M.N.M.Kahar,G.Kendall,"The examination timetabling problem at Universiti Malaysia Pahang:"MIT Press and McGraw-Hill.

[10] de Werra D., 1985. "An introduction to timetabling", European Journal of Operational Research, Elsevier, vol. 19(2), pages 151-162, February.

[11] W. Noodam, and P. Kongyong, "Developing Examination Management System: Senior Capstone Project, a Case Study", WASET Vol:7, No:7, 2013.

[12] R. D. Danielsen, A. F. Simon, and R. Pavlick, "The Culture of Cheating: From the Classroom to the Exam Room," Journal of Physician Assistant Education, 2006.

[13] L. T. G. Merlot, N. Boland, B. D. Hughes, and P. J. Stuckey. (2003). "A Hybrid Algorithm for the Examination Timetabling Problem," 4th PATAT, Springer.207-231.

[14] Abdelaziz Dammak, Abdelkarim Elloumi, "Classroom assignment for exam timetabling", Advances in Engineering Software Volume 37, Issue 10, October 2006, Pages 659-666.

[15] Burke EK, Elliman DG, Weare R. "A university timetabling system based on graph coloring and constraint manipulation", Journal of Research on Computing in Education 1994.

[16] Carter MW,Tovey CA."When is the classroom assignment problem hard",Oper Res 1992.

[17] Abdennadher SM, Saft S, Will S. "Classroom assignment using constraint logic programming", PACLP 2000, Manchester, April 2000.