Institute Timetable

B.Tech. Project Report [1]

Saeed Mirza (92228)
Final Year B.Tech.
Department Of Computer Science and Engineering
I.I.T. Kanpur

Project Guide : Dr. Dheeraj Sanghi

8th April 1996

[1]The software is being built for the Academic office of the institute

CERTIFICATE

This is to certify that the project **Institute TimeTabling** by *Saeed Mirza* (92228) has been carried out under my supervision and that, to the best of my knowledge, it has not been submitted elsewhere for a degree.

April 1996

(Dr.Dheeraj Sanghi)
Assistant Professor
Dept. of CS&E
I.I.T. Kanpur

Acknowledgment

I am extremely thankful to Dr. Dheeraj Sanghi for suggesting this project and providing enthusiastic guidance and constant encouragement during the course of the project.

I am also thankful to Dr. Vijay Gupta (Dean Of Academic Affairs) for helping me in getting the requirements for the software.

I would also like to thank the CSE Lab staff, and my batch mates without whom the going would have been difficult.

April 1996 (Saeed Mirza)

Abstract

This document describes the algorithms used in a new timetabling system that has been implemented at Indian Institue Of Technology, Kanpur in April 1996. We are given a set of courses and time periods of the week ,and a collection of available rooms on campus. We must determine an acceptable assignment of the time slots and rooms to these courses based on a variety of their requirements that measure their desirability for a particular time slot or room, or their desirability to be scheduled with another course or separate from another course.

The problem is subdivided into two separate components. Given the conflicts between the courses (i.e., some courses can't be scheduled in the same time slot) we assign time slots for the lectures and tutorials for the courses. During scheduling this constraints of room capacity requirement is also seen. Then the preferences for a particular slot and room is also taken care of. Preferences are considered in a global sense so that most of the courses have their constarints satisfied. We assmune that all the slots are one hour slots.

Contents

Chapter 1

Introduction

1.1 Motivation

In our Institute there are a set of classrooms available. Every semester some courses are offered. Each course has an expected enrollment . These courses are scheduled based on some policy directions of the academic office. Currently the scheduling is done manually (like most of the institutes). This is because there is no known algorithm which runs in polynomial time to find the solution. Only brute force search can be done which will require exponential time. Because the timetable is made manually the whole idea of having open electives given to the students is lost, because finally due to the clashes between courses, they are left with only few courses among which to choose. Due to the increasing number of courses and number of slots being same it is becomming more and more difficult for making the time table manually. Also the infrastructure with regard to room availability is poor making it difficult to find a good schedule. This leads to courses running in weekends, classes in afternoons, etc. The academic office would like to automate the scheduling process. So some methods have to be developed which use some heuristics to cut down the search space so that a solution is found in polynomial time.

1.2 Background

Over the past 20 years timetabling has become increasingly difficult in many North American schools, where the trend has been towards a flexible system of electives and programs tailored to individual needs and preferences. Examination timetabling problem and course time tabling problem have a lot of resemblance. The examinations scheduling problem in its simplest form, can be defined as assigning a set of examinations at any time. A solution of this form is called a "conflict-free" assignment.

It is difficult to draw a clear distinction between the examination timetabling problem and the course timetabling problem. Course timetabling often involves situations in which students have requested a set of courses, and the objective is to *minimize* the total

number of conflicts. In course timetabling periods often overlap and have varying lenghts. However, if a practical course timetabling problem requires a conflict-free schedule in uniform time periods, an examination timetabling algorithm would be appropriate. In the examination timetabling problem considered in this section, courses as well as examinations must be scheduled conflict-free and examination periods must be non-overlapping and of uniform size. So this can be easily seen as our problem in hand. *But these do not explicitly take care of the room assignment problem along with the time assignment problem.* There are various algorithms for room assignment also but these are applied only when the time slots are already given, so quite different from our problem case.

Before proceeding with a description of actual applications, we briefly outline the theoritical basis underlying each approach. The simple timetabling problem is equivalent to the vertex coloring problem in graph theory. The latter problem has been studied extensively, and a wide variety of heuristics is available in the literature.

1.2.1 The Vertex Coloring Problem

The problem of finding a conflict-free timetable is structurally similar to the vertex coloring problem studied extensively in the literature on graph theory. For a given examination timetabling problem, a graph is constructed as follows.

1. each course is represented as a vertex;
2. an edge connects two vertices if the corresponding courses have at least one student in common and hence cannot be scheduled in the same time period.

The graph coloring problem is usually posed as a question. Can the vertices of a graph be colored using a set of p colors so that no two vertices connected by an edge are both assigned the same color? The analogy with the examination timetabling is completed by associating the p available exam periods with the p "colors." The minimum number of colors required to color the vertices (denoted by χ) is called the *chromatic number* of the graph. The problem of computing the chromatic number of a graph is NP-Complete. The implications for timetabling depends on the structure of the particular problem. If the number of periods p is much larger than χ, then the problem of assigning p conflict-free periods becomes relatively easy.

Grimmett and McDiarmid (in 1975) have shown that, at least for random graphs, the simplest graph coloring heuristic will "almost always" use at most 2χ colors. So it is likely that for a graph coloring in which $p > 2\chi$, most heuristics will be sufficient to find conflict-free schedule. Carter (in 1983) presented some evidence to indicate that the graphs associated with timetabling problem are, in some sense, easier to solve than more general random graphs.

Practical timetabling (examinations) problems differ from the graph coloring problems when the following type of *secondary constraints* are added on the use of periods:

1. a limit on the number of classes in one period;

2. room capacity constraints;

3. consecutive examination constraints (i.e., certain exams must occur in adjacent time periods);

4. nonconsectutive conflict constraints (e.g., no examinations in succession for any student);

5. preassignments (i.e., certain examinations are preassigned to specific periods);

6. exclusions and time preferences (i.e., certain examinations are excluded from particular periods);

7. each student's examinations should be evenly spread over the examination period.

Our previous discussion implies that, for a particular school or university, if p is much greater than 2χ, then there is likely to be considerable flexibility in accomodating the secondary constraints. If p is much close to χ, the finding a conflict-free schedule becomes primary objective, and secondary constraints will typically be violated. Some schools try only to *minimize* the number of conflicts.

Graph Coloring Heuristics

Here we describe briefly those algorithms that have been applied to practical timetabling problems.

1. **Largest Degree First**. In this the vertices (courses) are ordered by degree (the number of courses with which it conflicts). Coloring proceeds by selecting courses from the top of the list and assigning the "lowest numbered" nonconflicting color. The rationale is that the vertices with most edges will be the hardest to color (if we wait until their neighbours have been colored).

2. **Largest degree first: fill from top**: As before the vertices are sorted by degree. In this method, we scan the list, placing as many courses as possibe in the first time slot (lowest color) and then go back to the top of the list and fill the second period, and so forth.

3. **Largest modified degree first** Williams (in 1974) proposed that the degree was not sufficient to determine how difficult a course was to schedule. he conjectured that a course was critical if a large number of its neighbors were critical. He used the following formula for computing the critical property of a vertex:

$$d_1(v_i) = \sum_{j \in A_i} d_0(v_j) \qquad\qquad i = 1, \ldots, n,$$

where A_i is the set of vertices adjacent to vertex v_i and $d_0(v_j)$ is the degree of vertex v_j. Hence $d_1(v_i)$ is the sum of the degrees of the vertices adjacent to v_i. He normalizes the $d_1(v_i)$ values and then computes

$$d_2(v_i) = \sum_{j \in A_i} d_1(v_j) \qquad\qquad i = 1, \ldots, n,$$

and recursively,

$$d_{k+1}(v_i) = \sum_{j \in A_i} d_k(v_j) \qquad\qquad i = 1, \ldots, n,$$

After a while, these "modified degrees" values will stabilize. (One can show that they converge to the principal eigenvector of the vertex adjacency matrix). Williams demonstrated that this approach is more expensive, but better, than the simple "largest first" heuristic in terms of the number of colors used.

4. **Smallest Degree last recursive**, the rationale is similar to "largest degree" methods in that vertices of *lowest* degree are *easy* to color. They are removed from the graph and placed at the end of the list. The degree of the remaining vertices are recalculated. When the list is complete, we then color vertices from top as before.

5. **Smallest degree last recursive with interchange**: The interchange rules applies equally to any of the largest first heuristic and proceeds as follows:

   (a) The vertex, denoted by c_i on the top of the list of unassigned courses is assigned to the lowest numbered nonconflicting color that has already been used.

   (b) If vertex c_i conflicts with all the current colors, find a color k_j for which there is only one conflicting course c_j. If possible, recolor vertex c_j. Otherwise look for a bichromatic interchange. Specifically, for each color r, locate the set of vertices C_r with color r that conflicts with vertex c_j. If the set C_r does not conflict with vertex c_i or any of the other vertices in color k_j, then interchange vertex c_j with the set C_r, which allows vertex c_i to be assigned color k_j. If no such interchange can be found, a new color is created for the course and the algorithm continues with the next course.

1.2.2 Timetabling Applications

This section presents a chronological survey of timetabling applications. Each algorithm is described in terms of its underlying vertex coloring algorithm , the modifications required to handle secondary constraints, and an overview of the implementation results.

One of the earliest published examples of timetabling is presented by Broder (1964). His stated objective is to minimize the number of student conflicts. His algorithm is basically a *largest degree first* algorithm; in case of ties, each course in the list is randomly assigned to one of the time periods that creates the fewest number of conflicts. Broder suggests a Monte Carlo simulation approach. The algorithm is run several times with a different random selection to break ties. The best run is selected.

In 1968, Wood devised an examination scheduling algorithm that was implemented at the University of manchester, England for more tham 1,000 courses , 6,000 students and 30 periods. The average number of courses conflicts for each examination was 15, producing a conflict matrix density of 1.5 %. The interaction between the various faculties was very low. The low density and separability allowed Wood considerable flexibility in estabilising his optimization criteria.

Wood's primary concern was that examinations had to be scheduled into a set of designed rooms on the campus with limited capacities. For this reason, he sorted the

courses according to the size of the rooms required. Within each group he used the "largest degree first" rule. For each course in the list, he searched for a feasible period with:

1. no adjacent conflict, or, if none;
2. no conflict on the same day (2 examinations per day) or, if none;
3. the minimum number of students with another examination on the same day.

In the event of a tie, he computed the total number of unscheduled courses that conflicted with the current course and could feasibly be scheduled into each of the given periods. The period with the minimum interaction was selected. This "look ahead" feature tries to avoid later scheduling problems. He then selected the room with the "closest fit" (i.e., the least acceptable number of places).

Wood claims that, when the algorithm failed, inspection of the conflict pattern related to the unscheduled courses "clearly reveal the subjects which cause the difficulty". These subjects are preassigned manually and the algorithm is repeated. using manual intervention, Wood was able to schedule all examinations in 24 periods, even though 30 periods were available.

The "look-ahead" feature of Wood's algorithm reflects an important difference between timetabling and coloring. In practical problems, the unscheduled courses may be restricted to particular time periods. When constraints of this type are present, the "look-ahead" feature of an algorithm is useful in tie-breaking.

Also in 1978, Carter developed an algorithm for final examination scheduling at the University of Waterloo. The system developed from this algorithm has now been in use for several years, and was also implemented by the Waterloo County Board of Education for scheduling all area high school examinations. The algorithm basically uses the "largest degree first: fill from top" rule. In case of ties, a frequent occurence in this model, preference is given to large enrollment courses and then to certain special courses designated as "preferred" by the faculties. The major constarints were:

1. several courses must be preassigned to fixed time periods;
2. no student should be required to sit for three or more consecutive examinations;
3. certain examinations are designated as evening or Saturday only.

Restriction 2 was not implemented literally, due to the sheer volume of data associated with maintaining and veryfying each student's record. Instead, a more restrictive rule disallowed scheduling any examination that had conflicts in the two previous periods.

The University of Waterloo has 17,000 students taking 600 examinations in 36 periods. There are 3 examination periods per day, 6 days per week, for 2 weeks (or 36 periods).

During the fall term in 1981, there were 552 examinations at Waterloo, with a conflict density of 5%. Over 100 courses conflicted with more than 90 other courses, and one course had conflicts with 312 others. There was at least one group of 22 mutually conflicting examinations. Since none of these examinations could be scheduled in any 3 consecutive periods, the schedule required a minimum of 32 examination periods. The situation was actually even more complicated, since many of these courses were constrained to the 16 evening or Saturday periods.

In the light of the complexity, it is not surprising that Carter's algorithm has encountered problems scheduling all examinations in 36 available periods. In some semesters, it was necessary to relax the "3-in-a-row" constraint for the first few days. Even so, few examinations are usually left out. These are inserted manually by shifting one or two "blocking" examinations. This manual procedure could be computerized using a constrained version of the "bichromatic interchange" routine.

Many such implementations have been done but each was developed for some specific problem at a particular school. None of these "packages" have been used by more than one or two sites, and none of their developers compare their results against alternative approaches. In fact, most authors were unaware of the existence of other published material.

Chapter 2

Requirements

2.1 Time Tabling in I.I.T. Kanpur

After speaking with Dr. Vijay Gupta (Dean of Academic affairs) we found that the the present system of scheduling operates as follows

1. Each instructor gives the info about the course he will be offering like the expected enrollment and some of his preferences for rooms or time slots or the information of the previous semesters is used for the same.
2. There are five slots 8-9 am, 9-10am, ..., 12-1pm for five days (Mon. - Fri.)
3. Various modules are made for the courses , some of the modules are ESO ,BSO, HSS-1 ,etc. Courses are put in these modules. Now these courses are scheduled together i.e. in the same time slot so any student can take atmost one course from each module. This makes the problem size smaller atleast for time allotment But then it gives less number of choices for the students. But these modules are made on the basis of department requirements ,like some department might want to have Linear algebra and Intro. to E.E. taken by their students ,so these courses must be put in different modules.
4. Tutorials are scheduled for Tuesdays and Thursdays.
5. There are two types of courses - Regular and Slow pace . Each of these have different requirements. Also in slow pace there are two categories Category 'A' Students-who have to take only one Slow pace course or the students who have to take 2 slow pace courses one of which is English. Category 'B' Students - all Slow paced students excluding 'A'. These courses have also to be scheduled along with the regular courses so that there is no conflict.
6. There should not be any 1st year class from 8-9 a.m. in the morning.
7. Lectures and tutorials are to be in the morning and labs in the afternoons. But it may not be possible to do that ,so the last choice will be to schedule Tutorial in the afternoon. some classes may also be required to be scheduled on Saturdays.

8. Chemistry tutorials should be in two groups (2 sections meeting at different Hours will be easier for the tutors)

9. There are sections for labs. These sections are same for a student for all the core lab courses . Now labs are scheduled section wise so as to have no conflict.

10. Courses like M101 and M203 should'nt be at the same time because students might be common.

11. Ta202 has 2 labs for each student (4+3 hours). So there should be no class from 12-1 for students having lab at 1-5 p.m. So can say for Section A free that slot and for section B give a course slot.

12. Certain rooms in the Department or Lecture Hall Complex or Tutorials are reserved for the Department courses for some time slots. No Core course should be scheduled in that room for that slot. The preferred rooms with their capacity is given below:

ROOM NO	CAPACITY	RESERVED FOR
L1	244	
L2	244	
L3-L6	120	
L7	460	
T101-T111	30	
T112	30	Aerospace
T201-T212	30	
WL218	58	Electrical
WL221	58	Aerospace
WL222	55	Civil
WL225	45	Metallurgy
WL226	48	Central Schedule
WL237	34	"
WL229	20	"
WL228	20	Preparatory
FB656, 657	55	HSS
FB668	35	
FB556	45	Maths
FB563, 564	15	Maths
FB470	50	Chem
FB482	40	Phy
FB370	60	Mechanical

The above rooms reserved are not complusory and may not be be adhered to, but it makes the problem somewhat easier for the departments to schedule their courses.

13. Avoid scheduling classes in L5 , keep it for the department.

14. HSS tutorials require large rooms .

15. Two tutorials on adjacent days not wanted.

This was the way the scheduling is done keeping in mind the above restraints. Last years time table is sometimes used to see how scheduling was done. Or sometimes everything is started from scratch. Courses are assigned randomly with some intutive idea, if some is not able to be scheduled then some changes in the schedule till now is done.

One of the most important distinctions between the various algorithms involves a classification on a scale between the two extreme cases:

1. those that concentrate exclusively on finding a conflict-free time-table, and
2. those that also try to minimize the number of violations of secondary conflicts.

For our case the later will be preferred.

2.2 Problem Definition

The problem requirements can be divided into following main categories :

1. **Time Slot scheduling**
   - There are certain conflicting courses with common students and so can't be scheduled in the same time slot. This is a primary constraint.
   - Allow the user to enter the courses which should be preferably scheduled in the same time slots. At present this is a secondary constraint.
   - Allow the user to enter preferences for the time slots he wants and the slots he won't prefer . This is a secondary constraint.
2. **Room allotment for courses**
   - For each course and each class (lecture or tutorial) of that course a room of suitable capacity should be allocated so that all the students who take that course can be accomodated in that room . This is a primary constraint.
   - Allow the user to enter preferences for the room he wants and the rooms he won't prefer. This is a secondary constraint.
3. **Some Implicit Restraints**
   - Lectures and tutorials should be in the morning .
   - A tutorial for all sections in a course must be in same time slot
   - Two tutorials for a course must not be on two consecutive days.
   - Two lectures for same course must not be there on same day.
   - Allow a one-hour lunch break to each student.
4. **Flexibility Allowed**

- After the timetable has been made the user must be allowed to change some slots and system must respond to it.

5. **Functionality Requirements**

   - The system should develop a schedule which meets all the primary constraints and as many secondary constraints as possible for all the courses.

6. **Outputs**

   - Output the time table as course number and the slots+rooms alloted or as slots and the courses in that slots along with their rooms.
   - Output all courses which could not be scheduled because some primary constraints could not be satisfied.

2.3 Hardware Requirements

- A PC (80386 being the least)
- A Mouse for interface

2.4 Software Requirements

- Borland C++ Version 4.0, and it's Dynamic Link Libraries accessible to the software.
- MS-Windows version 3.1 on the PC.

Chapter 3

Design of the System

3.1 Introduction

3.1.1 Purpose

This section describes the design of the project **Timetabling** , with a specialised case being the Time Tabling for the Core courses of an Institute ,such as IIT Kanpur. Along with the allotment of **time slots** for the lectures and tutorials of the course it takes care of the **room allotment** also.

3.1.2 Scope

The scope of the project includes the designing the timetable for any institute with the requirements similar to those of our institute or whose requirements can be converted to those being handled by this system. This can be used for both the core courses timetabling or the departmental timetabling or both combined, where we will treat all the courses as similar. Also if the coloring alogorithm used by this software is not good for the particular data of that institute then some other algorithm can be used by changing just one module in the system without affecting the other modules.

3.2 Definitions, Acronyms, Abbreviations

info information

BC4 Borland C++ Version 4.0 for Windows

he Any reference to 'he' should be taken as 'he' or 'she'.

3.3 Overview

The package implements the following items :

1. Input the data for timetabling
    - Input Courses :
        - Courses Online
            * Open for adding
            * Add an item
            * Add more Courses
            * Cancel the last course
            * OK addition (add the last item and end)
        - Courses From File (Format Specified in User MAnual)
    - Input Rooms :
        - Rooms Online
            * Open for adding
            * Add an item
            * Add more Rooms
            * Cancel the last room
            * OK addition (add the last item and end)
        - Rooms From File (Format Specified in User Manual)
    - Input Conflicts/Preferences
        - Conflicts/Preferences Online
            * Conflicts between Courses (Group of courses to be scheduled together or separate)
                · Select courses from the course list.
                · Group them to be scheduled together or separate.
                · End grouping .
            * Preference of Time Slots for a course (Group of time slots this course likes or does not like)
                · Enter Course no.
                · Select time slots from the slot list.
                · Group them as good slots or bad slots for the course .
                · End grouping .
            * Preference of Rooms for a course (Group of rooms this course likes or does not like)
                · Enter Course no.
                · Select rooms from the rooms list.
                · Group them as good rooms or bad rooms for the course .
                · End grouping .
        - Conflicts/Preferences From File (Format specified in User Manual)

2. Allocate the time slots and rooms
3. Display
    - Display Slot Wise
    - Display Course Wise
    - Print to File (both Slot Wise and Course Wise)
4. Quit the Software after saving the data

3.4 General Description

3.4.1 Product Perspectives and Product Functions

The system is to produce a schedule for the institute which specifies the time and room assignments for the different courses. The preferences given by the instructors should be taken into account as far as possible. If the system is not able to schedule certain courses than it should mention those courses .

3.4.2 User Characteristics

- The users of the system will be Academic office officers who should have some familiarity with the computer system. Familiarity with use of mouse is required. The software has been made very user friendly keeping in view of a very naive user.

3.4.3 General Constraints

- The system is to run on PC's 386 series onwards.

3.4.4 Assumptions and Dependencies

- The user should have access to the Borland C 4.0 dynamic link libraries in his environment variable PATH.

3.5 Design Decisions and Implementation

The following sections describe the design and implementation decisions taken together with the interface commands. The actual interface is described in the Appendix : User Manual.

An object-oriented design approach had been undertaken. The major reasons for this approach are :

- Reusability : The approach is such that the software can be modified to incorporate the needs of other institutes. Also if the basic graph coloring alogorithm needs to be changed then also only one module will be changed.
- Incremental Building : The software has been built in an incremental way. First the time assignment was carried out. Then the room assignment was incorporated in it. The the secondary constraints for the slots and rooms preferences was also taken care of. This was simplified by this design.

The project was carried out in BC4, an object-oriented language.

3.5.1 Model Of The System

The system was modelled as a graph where the nodes correspond to the courses and the edges between them means that there is a conflict between the courses. The **Largest Degree First** heuristic was used to color the graph. The secondary constraints were also taken care of while coloring the graph. This will be more clear later .

3.5.2 Data Structures Used

The project has one major object :

- The Main Interface Window Object **TTWindow**

The following data structures were used in the implementation of the above design

1. **Course**: It is used to store the courses info. It has the following fields -
   - The name of the Course .
   - The Course number .
   - No of lectures .
   - No of tutorials.
   - No of labs.
   - No of Sections.
   - Expected enrollment.
   - Course id, given by the system
   - Degree of the node (assigned during system operation, it denotes the number of courses which conflict with this course)

- The time slots assigned to it.
- The rooms assigned to it.

2. **Room**: It stores the information about the rooms available in the institute. It has the following fields -
   - The Room number .
   - Capacity of the Room.
   - The id of the room given by the system.
3. **Lcolors**: Structure for maintaining the list of time slots either for storing the forbidden slots, liked slots or the sorted list of slots. It has the following fields :-
   - the color (time slot number),
   - the likeness value of the slot (indicating how much is this slot liked globally,
   - a pointer to Lcolors.(So this is a linked list.)
4. **RoomsList**: Is is a linked list of rooms, with the fields as room and a link to RoomsList.
5. **CourseList**: Is is a linked list of courses, with the fields as Course and link to CourseList.

The following structures are also used

1. **conflict**: this is a matrix for storing the conflicts between courses,
2. **slot_pref**: this is a matrix for storing the preferences for the courses for the differnt time slots,
3. **room_pref**: this is a matrix for storing the preferences for the courses for the different rooms.

The following classes were used basically to use the dialog boxes for inputs and outputs

1. **CourseDialog**: For using the dialog box for entering the courses online. It has some edit boxes for the coursename, coursenumber, no. of lectures, no of tutorials, capacity, and some button responses functions to end the input, or to go to the next input.
2. **RoomDialog**: For using the dialog box for entering the rooms online. It has some edit boxes for roomno and capacity, and some button responsese for ending the input or to go to the next input.
3. **ConflictDialog**: For using the dialog box for entering the conflicts between courses. It has two listboxex. One for the courses and other to list the selected courses. There are functions for selecting some course, for making the selceted courses as conflicting, for making the separate courses schedule in the same time slot, and for ending the input.

4. **RoomPrefDialog**: For using the dialog box for entering the preferences of a course for the rooms. It has one editbox for entering the course number, two listboxes, one for the rooms list and other to list the selected rooms . There are functions for selecting some room , for making the selected rooms as good rooms for the course, for making the selected rooms as bad rooms for the course , and for ending the input.
5. **SlotPrefDialog**: For using the dialog box for entering the preferences of a course for the time slots . It has one editbox for entering the course number, two listboxex, one for the slots list and other to list the selected slots. There are functions for selecting some slot, for making the selected slots as good slots for the course, for making the selected slots as bad slots for the course , and for ending the input.
6. **TTDayWiseDialog**: For using the dialog box for outputting the timetable slot-wise. It has 25 list boxex, one for each time slot to display the courses scheduled in that slot, functions to end examining the timetable.
7. **TTCourseWiseDialog**: For using the dialog box for outputting the timetable coursewise. It has one list box for displaying the courses along with their slots and rooms, and function to end examining the timetable .
8. **HelpDialog**: For using the help dialog box for giving online help to the user. It has one list box for comments, and function for ending the help browsing.

3.5.3 Implementation

The project has four major sub-parts, each of which is as described later :

- Inputs
- Allocate
- Display
- Quit

Inputs

This part inputs (both new data or appends to the older data), mainpulates the older data for the courses info, rooms info and the conflicts between courses and the preferences of a course for rooms and time slots. The following functions are available. The detailed Inputs interface is mentioned in the User Manual.

- Input the Courses
- Input the rooms
- Input the Preferences/Conflicts

Each of the above interface functions is followed by the following events

Input the Courses

The course could have been entered online or from a file. This command creates a course node in the course list : The course is checked whether it is a duplicate one, if yes then the previous data is overwritten by this new data. If it is a new course then it is appended at the beginning of the course list.

Input The Rooms

The room could have been entered online or from a file. This command creates a room node in the rooms list : The room is checked whether it is a duplicate one, if yes then the previous data is overwritten by this new data. If it is a new room then it is inserted in the rooms list which is maintained in decreasing order of capacity .

Input the Conflicts/Preferences

This can be taken either from a file or online. These inputs are of following types: (For all the following inputs it checks whether the course, or room is a valid entry or not)

1. A group of two or more courses to be scheduled separately. It results in following manipulations - For all pair of courses it enters a value 1 in the corresponding entry for the pair of courses in the *conflict* matrix.
2. A group of two or more courses to be scheduled together . It results in following manipulations - For all pair of courses in the selected list it enters a value -1 in the corresponding entry for the pair of courses in the *conflict* matrix.
3. A group of time slots selected as good slots for a course. It results in following manipulations - For all the time slots in the selected list enter a value 1 in the corresponding entry for the (course, time slot) pair in the *slot preference* matrix.
4. A group of time slots selected as bad slots for a course. It results in following manipulations - For all the time slots in the selected list enter a value -1 in the corresponding entry for the (course, time slot) pair in the *slot preference* matrix.
5. A group of rooms selected as good rooms for a course. It results in following manipulations - For all the rooms in the selected list enter a value 1 in the corresponding entry for the (course, room) pair in the *room preference* matrix.
6. A group of rooms selected as bad rooms for a course. It results in following manipulations - For all the rooms in the selected list enter a value -1 in the corresponding entry for the (course, room) pair in the *room preference* matrix.

Allocate The algorithm used in allocation is as follows:

```
algorithm Allocate
{
    Calculate the degree of each course node from the conflict matrix
    Sort the courses according to their degree in descending order
    Add pseudo nodes in the course list, one node for each section of a course
    While the course list is not empty do
    {
       Pick the course from the beginning of the list
       Calculate the forbidden slots of the course
       Calculate the common slot of the course
       Arrange the slots for this course according to preferences
       Assign the time slots and rooms for this course
    }
    Remove the pseudo nodes by putting the data back into the original node.
    For each time slot implicitly assign rooms to the courses in that slot.
}
```

Pseudo nodes are a copy of the courses which have delivered them. For them the number of lectures is 0. They areone for each section and behave same way as the original course. Also the size of room requirement is the total size/no of sections. The original course has number of tutorials set to 0.

Forbidden slots are the list of slots which are forbidden for the course. These are the slots which are already assigned to courses which conflict with this course. For pseudo courses also it is calculated in the same way except that it also has the slots which were already assigned to the lectures of this course.

Common slots are the list of slots which are the most preferred slots of the course. These are the slots which are already assgned to courses which need to be scheduled with this particular course. For pseudo courses also it is calculated in the same way except that it also contains the slots which have already been assigned to the other sections of this course.

The time slots are arranged in such a way so that the beginning of the list has the slots he prefers, next the slots he doesn't care for and in the last the slots he doesn't like. So these have to be broken into three classes. For this a particular number is added to the likeness value of the slot for the course we are dealing with at present. Next the arrangement between the slots in the class is such that most of the preferences of all the courses are satisfied.

```
algorithm Arrange slots
{
    for each time slot
    {
       Calculate likeness value of the slot as follows
       {
           for all the courses sum the preference value of that course for this slot
```

```
            add the number number to break this into the particular class of slots
        }
        insert this slot in the sorted slot list as follows
        {
            if this slot is a preferred slot then add a number to its likeness value
            so that this slot will be presented before other slots
            Put the slot in the sorted list in decreasing order of the likeness value
        }
    }
}


algorithm Assign
{
    For all the lectures and tutorials in the course do
    {
        while a satisfactory slot is found
        {
            pick a slot from the beginning of the sorted list of slots
            if slot list is empty then say that the course can't be assigned and retrun
            Check for Rooms in this slot
            if room available then assign this slot, this is a satisfactory slot
        }
        reshuffle the sorted slot list
    }
}


algorithm Check for Rooms
{
    for all the preferred list of rooms for this course
    {
        if this room is free allot this room to the course
        check whether the courses already alloted in this slot have enough room
        if yes then allot this room permanently to this course and remove the room from
        the list of rooms for this slot also put this course in the slot_courses list
        if no then continue with other rooms
    }
    if the course has a negative preference
         check for the don't care rooms one by one as in the preferred room case
    else
    {
         check whether this course along with the other courses in the list have
         enough rooms in this slot
         If yes then put this course in the slot_courses list with room unassigned
         If no then room can't be assigned in this time slot
    }
}
```

To check whether a set of courses have enough rooms from the list of rooms left in the slot, we note that the rooms are arranged according to the capacities and the courses also according to the capacity requirement. So can assign the rooms from one end one by one, if at certain stage it is not possible then it means that enough room is not there.

The purpose fo reshuffling the sorted slots list is to prevent two lectures of a course to be scheduled in the same day, and two tutorials on two consecutive or same day. For this the slots of the same day (or day before or after also for tutorials) are taken and placed at the end of the list. Also if a course lecture is scheduled in some slot in some day it is preferred that on the other lectures are also scheduled in the same time on other days. For this the corresponding slots are placed at the beginning of the list. This operation is performed before the previous operation.

Finally when all the courses have been assigned the slots, the courses in the pool for each slots which were not given rooms for the time being are given rooms now. As these are arranged in increasing order of their capacity requirements these are assigned rooms from the pool of available rooms which are also arranged similarly.

Display This part displays the slots and the rooms allocated to all the lectures, tutorials for all the courses. It has the following parts:

- Display Slot Wise
- Display Course Wise
- Print to File

Display Slot Wise

It displays the time table slot wise. For all the slots it displays the courses (and the corresponding room alloted to it) scheduled in that slot. It also displays the section number if it was a tutorial.

Display Course Wise

It displays the time slots and the rooms allocated to each course for its differnt lectures and tutorials. It also displays the section number if it was a tutorial.

Print To file

It prints the time table both course wise and slot wise in separate files. Closing the database just cleans up the system of it's internal structures, and updates the .ind file.

Quit The system saves the courses from the course list into the courses.l file, the rooms list in the rooms.l file, the conflicts and preferences matrices in the conflict.l file. It frees the system of all the internal structure and quits.

3.6 External Interface

3.6.1 User Interface

General Description :

The platform used and assumed is MS-Windows and hence, the user interface will be menu and window-based, using a mouse, or keyboard for entering the data .

The graphical interface that has been given has been shown in the User Manual (Appendix A).

3.6.2 Hardware Interface

The following hardware is required for the software.

- Mouse interface required
- PC with 4 MB primary memory .
- PC : 80386 (at least)

3.6.3 Software Interface

- MS-Windows will be the platform for the software. The messages passed by it to the application will be keyboard input , mouse status and handles for reading/writing data on the screen and the file-system.

3.7 Design Constraints

The software is to run on MS-Windows, on a DOS platform.

Chapter 4

Conclusion

4.1 Results

The software was tested on various dummy data and also on the full time tabling requirement of the core courses of our institute. This was able to schedule all the courses in the morning slots in the 5 days of the week with a few slots left unassigned (called free slots).

4.2 Extensions to the project

There are many extensions possible to this project. They are as :

- Allow the user to change the schedule obtained finally online and see the effect of changes.
- Provide the user with the list of unassigned courses which can be manually put in some slots and see the effect of it.
- It can be adopted for the use of other institutes by changing the coloring strategy if they need much better heuristic because they have more number of constraints.

Chapter 5

References

- BC4 Programmer's Manual : Borland International
- BC4 Reference Manual : Borland International
- BC4 User's Guide : Borland International
- A Survey Of The Practical Applications Of Examination Timetabling Algorithms By: Michael W. Carter (Operations Research Vol.34. No.2, March April 1986).

Appendix A

- The User's Manual is provided.
- The file organization is also mentioned in the User's Manual.