

# CHAPTER I

## INTRODUCTION

### A. Background of the Study

“Timetabling is the allocation, of subject to constraints, of given resources to satisfy as nearly as possible a set of desirable objectives.”[1] Real timetabling problems have many forms like educational timetabling (course, exam, and project presentations), employee timetabling, personnel scheduling, timetabling of sports events, transport scheduling, etc. [2,3]

Educational Timetabling Problems include finding the exact time allocation within a limited period (e.g. week), of a number of events (courses, exams, project presentation) and also assign to them a number of resources (a teacher, a room, etc.) in such a way that a number of constraints (in other words, restrictions) are satisfied [2 – 5]. Constraints involve, among others, overlapping of events with common participants, capacity of rooms, and student and teacher workload.

The University of the Philippines – Manila, like most universities, still implements the manual timetabling. In order to simplify the task course scheduling – which is a primary task of each college secretary – scheduling processes were distributed to the heads of each

department on each college. In doing so, the course scheduling problem is now divided into several subproblems of timetabling.

Each department head will then consult his department, and (knowing what courses they will be offering or are allowed to be offered) will decide on: number of sections for each course; slots allotted for each section in each course; number of meetings per week, lecture hours, and laboratory hours, and exam hours; and the instructors/lecturers which will be teaching each course, then provide a tentative schedule for the semester. The Office of the College Secretary, will then combine and resolve conflicts which may arise from the proposed schedules of all the departments in their jurisdiction.

## **B. Statement of the Problem**

As simple as it may seem, the timetabling problem is well known to be Nondeterministic Polynomial-complete (NP-complete) combinatorial problem [5, 6, 7], which means that it is difficult to find the best solution to the problem. Its difficulty increases exponentially whenever more scheduling parameters are added. “At present, science has no analytical solution method for all problems, due to the immense search of spaces of real problem cases of this category other than exhaustive search, which however cannot be applied but only to toy problems, due to immense search spaces or real problem cases”. [2]

“Large-scale timetable, such as university timetables,” in the College of Arts and Sciences, University of the Philippines Manila, “may need great effort and many hours of work spent, by a qualified person or a team, in order to produce high quality timetables with optimal constraint satisfaction and optimization of the timetable’s objectives at the same time” [2]. Hence, dealing with such by hand whenever a new semester starts does not benefit our educational institutions, since aside from not guaranteeing quality timetables, it also involves huge expenses in resources – entails more time, effort and paper works.

### **C. Objectives**

This project endeavored to propose a better approach to solving the timetabling problem, with the College of Arts and Sciences of the University of the Philippines – Manila as the pilot setting of the system. The College of Arts and Sciences Scheduler (CASS), aims to provide optimized set of schedule each semester from which eligible users can choose from.

CASS, the scheduler, has the following as its users with their respective functionalities:

- (1) An Instructor / Lecturer will be able to
  - a. Input scheduling parameters (preferred courses, preferred rooms, time of unavailability);
  - b. View implemented schedule; and

c. Change Password

(2) The Department Chairman will be able to

- a. Input scheduling parameters of his department (the subjects to be taught; the number of meetings; lecture hours, laboratory hours, and exam hours; the number of sections for each subject; the instructor/s or lecturer/s capable of handling each subject; check and verify the entries made by the each instructor or lecturer);
- b. Update courses, subjects, and faculty of his department
- c. View implemented schedule
- d. Change Password

(3) The Office of the College Secretary (OCS) Personnel (which will also be the system administrator) on the other hand, will be able to

- a. Input scheduling parameters (the rooms to be used; slots to be allotted for each subject – slots; the subjects to be taught for courses outside CAS);
- b. Generate Optimal Schedule;
- c. Accept Generated Schedule;
- d. View Implemented schedule;
- e. Add or dissolve a subject;
- f. Update users, rooms
- g. Change Password

#### **D. Significance of the Study**

Every Educational institution faces the problem of timetabling or scheduling. When done manually, timetabling requires so much time and effort especially if there is a limited amount of resources (rooms, instructors, etc.). Taking these into consideration, a system – *CASScheduler* – was developed to provide an optimal solution and guide the OCS to generate schedule in a minimal time. An optimal solution can allocate resources efficiently and this would be beneficial to the College of Arts and Sciences of the University of the Philippines Manila.

Starting from the gathering of data, which has been made more efficient through the use of technology, to the presentation of results and solutions, the system was designed to greatly reduce the overhead for manual timetabling. Also, instructors need not worry anymore with their schedules – they can now easily allot time for teaching and time for other works they do. Department Chairs won't have to be troubled anymore about having to schedule classes to same courses and year level on the same timeslot.

Moreover, since *CASScheduler* has made use of Non-dominated Sorting Genetic Algorithm II, it arrives to an optimal or set of optimal solutions in a shorter time compared to the manual way. Furthermore, everything that has to be considered (like room assignments, demand for subjects, assignment of instructors, etc) can be taken into account simultaneously.

## **E. Scope and Limitation**

The system will encompass the class (course) and final exam scheduling process, from the gathering of input data from the instructors and/or lecturers, department heads and OCS personnel, to the processing of input and generation of solutions itself, until the presentation and delivery of the set of viable solutions to the parties involved. The system will be able to process and generate results for the scheduling of undergraduate classes (courses) and final exams within the UP Manila College of Arts and Sciences, wherein evaluation of the solutions will be patterned after the criteria and considerations taken by experienced personnel from the said college, as well as general standards obtained from research.

The system would not know prior to the processing of data, if there could always be a solution (or even a partial one). *CASScheduler* is just a support system, and the decision of which among the solutions generated will be implemented still lies within the eligible personnel (an OCS Personnel), and choosing which among the solutions is best is outside the scope of the system. Furthermore, changes of schedule outside those entered by the department chairs and implemented by the OCS is not part of the system.

## **F. Assumptions**

Since the system is just concerned about optimizing a schedule based on a given sets of inputs, the following are assumed:

1. Before making a run of the algorithm (through the system), all the entries are considered final.
2. All the inputs are correct – there are no format errors. The system will provide the user an input format and it is assumed that users will adhere to that format.
3. Any other arrangements to be made between any user of the system (department head-to-instructor, college-to-department, etc.), are made prior to the scheduling proper.
4. It is also assumed that during final examinations period, all instructors are available, or if not, he may be able to provide a proctor in place of himself.
5. Restriction of rooms applies both to course and exam scheduling.
6. At least one instructor is assigned to teach a course being offered.
7. Department Chairs and instructors can only input scheduling parameters prior to the scheduling proper. If a schedule has already been accepted and implemented, they can only view and not modify whatever they have inputted previously.
8. Moreover, it is assumed that users of the system are well-trained and knows how to use the system effectively.

## **CHAPTER II**

### **REVIEW OF RELATED LITERATURE**

Several universities have tried to solve the timetabling problem which they always experience whenever a semester or school year starts. Some may have just stick to their old way of solving it, while others may have devised some ways to ease the mundane and arduous task of scheduling.

Schedule-EZ [8], is a tool that has been developed to assist chairs and secretaries of various departments to facilitate the mundane, error prone and time consuming task of faculty scheduling. It is a powerful database driven tool that was created with simplicity and specifics in mind. The program entirely written in Visual Basic, with MS-Access as the database and export capabilities to MS-Excel, was used by various departments in Northwest Missouri State University in 2003. It has been proven to be an effective tool for department chairs and secretaries. Schedule-EZ is merely an automation of the timetabling process which consists of three main components – the control panel, the daily schedule, and the entire week view. “The control Panel allows the user to customize the program to suit a department’s need. Department faculty names, classroom, and courses offered are stored ... this customization will personalize the program for ease of use later when the user begins scheduling...The daily schedule is the main part of the program ... where the faculty names, courses offered by the department and the room locations appear in a drop down menu... the



interesting feature that is available is the validate button”. This implies that it is still the user who schedules and not the system. After completing the scheduling the validate button, when pressed checks to see if scheduling conflicts exists.

QUICK Scheduler [9] which was used at Texas Tech University (TTU) back in 2005 is a web-based application that aimed to help students and academic advisors with the scheduling process. The user will input the courses he is about to take in a certain semester and the scheduler will select sections and courses that do not conflict with other classes or with other specified activities (such as their work outside school, basketball, practice, or family commitments, etc.). The final output is a one-page graphic schedule , showing activities the student has entered as well as his sleep time, study time, and class time. QUICK Scheduler also emphasizes the importance of allocating sufficient sleep and study time A backtracking algorithm was used for producing the results (schedule) [10]. “If an acceptable schedule is not found on the first try, the student or advisor can change one or more courses or other criteria and submit again. This can be repeatedly done until the optimal schedule is found” [10]. Again, this shows an implementation of a mere human way of solving the timetabling problem, even if backtracking algorithm was used.

To simplify the highly constrained timetabling problem, Swansea’s TISSUE examinations scheduling system [11, 12] divide it into two phases – first finding a feasible solution, then optimizing secondary constraints.

Tabu search has been applied successfully by Boufflet and N`egre to generate examinations timetables at the University of Technology of Compi`egne [13], Their tabu list contains the seven most recent moves. If the current neighbourhood does not contain an improved solution, the aspiration function may select one from the tabu list.

Formulating course scheduling as an assignment problem, Hertz developed and applied the TATI tabu algorithm [14], which he later adapted for a more complex and constrained real-life course scheduling problem [15]. The length of a lecture is not fixed in advance and there are ten different types of moves (e.g. moving a lecture to another day, changing the duration of the lecture etc). When the schedule of a particular lecture in a particular day is changed it may be moved to another period (possibly in another day). However, for a given number of iterations it is tabu to move the lecture to a period in the original day.

Corne, Ross and Fang found an intelligent mutation operator to be more successful than two-parent crossover [16]. Their system, GATT, is now being used successfully to timetable courses at the University of Edinburgh and several other institutions.

Paechter, Cumming *et al* have developed “Neeps and Tatties”, a system which is being used to schedule courses at Napier University’s Computer Science department. Its genetic algorithm encodes timetables as an ordering of events, which must be input to a special program which uses the order to produce a timetable [17]. This necessitates a

different sort of recombination operator, which takes elements of the order from each parent to produce a new ordering.

The Automated Scheduling And Planning group at the University of Nottingham, has developed genetic algorithms for examinations scheduling which employ a large degree of heuristic knowledge, both to seed the initial population, and to improve the standard genetic operators [18 – 20].

Fernades, Calldeira, *et al* introduced an new operator, “Bad Genes Mutation, which greatly improved the evolutionary algorithm’s speed. The algorithm was tested on a large high school called D.F.L. using the 1996/1997 school year timetables [21].

UTTSExam is the exam scheduling portion of University Timetable Scheduler (UTTS) software, an automated university timetabling program developed in the National University of Singapore (NUS), which when completed, the program is expected to automatically schedule both the course and examination timetables for all the faculties in the entire university that employ the modular academic course structure. While the exam scheduling portion of UTTS reached the deployment stage and was used to generate the 2001/2002 academic year in NUS the other portion – the course scheduling – is currently still under development [22]. UTTSExam also made use of artificial intelligence technology. It used the Combined Method [23] for solving Constraint Satisfaction Optimization Problem (CSOP) [24]. It also made use of Genetic Algorithm [25] with Tabu Search Post Optimization [26].

An advanced genetic algorithm, which made use of the indirect representation in encoding a timetable solution, was developed and used by Karzalis, Petridis and Fragkou in solving the timetabling problem and was applied to the Technological Educational Institute of Serres in Greece for which the solutions were compared to that of the man made. A similar algorithm has been proposed in [27] where the non-evolutionary heuristic algorithm is proposed for exam timetabling problems.

Perzina designed an optimization model for solving the university timetabling problem that is capable of dealing with individual timetables of every student. A parallel self-adaptive genetic algorithm with self-adaptation of all its parameters was proposed. This algorithm was applied for solving the real university timetabling problem at Silesian University of Czech Republic, and has shown to be effective. An enrollment optimization algorithm when dealing with individual timetables of students was also proposed, which when implemented, has significantly decreased the number of student clash constraints [5].

Kov, aiming to produce “high quality timetables”, presented methods for solving university timetabling exam problems on his doctorate thesis last November 2003. In the course of his thesis, he developed a variant of NSGA for exam timetabling, which employs elitism. He also introduced the idea of a trajectory-based multiobjective approach which enables the search process to move along defined trajectories. [28].

NSGA-II algorithm was used as the core of the course scheduling system (CSS) presented on March 2006, by Gagno *et al* [29] of the University of the Philippines, Diliman, in partial fulfillment of their bachelor's degree. "The team has demonstrated that the CSS project is capable of generating feasible solutions to the course scheduling problem, given a set of courses, resources and constraints to be observed. It is able to reduce the overhead for time, labor and paper by a great scale".

Also on June 2006, NSGA-II-UCTO: NSGA-II as University Class Timetable Optimizer developed by Datta *et al* [30] as a multiobjective EA-based university class timetable optimizer in solving class timetabling problems of the Indian Institute of Technology Kanpur. With the use of NSGA-II-UCTO, a number of trade-off solutions, had been obtained very easily. "Moreover, much better results, than the manually prepared one, have been obtained using NSGA-II-UCTO". [31]

## CHAPTER III

### THEORETICAL FRAMEWORK

#### A. The University Timetabling Problem

Timetabling, as described by de Werra, is the activity of scheduling a set of meetings or events in such a way that certain requirements and constraints are satisfied [32]. Timetabling problems include: educational timetabling, sports timetabling, employee timetabling, transport timetabling and others [3].

The university timetabling problem can be described as follows. There are  $q$  events  $e_1, \dots, e_q$ , a potential set of timeslots or  $p$  periods  $1, \dots, p$ ,  $m$  rooms  $r_1, \dots, r_m$  which the events can occur, and a potential set of agents (or professors) tasked to handle each event  $e_i$ . Each room  $r_j$  has a capacity  $cap_j$ , expressed in terms of number of available seats. There are also  $g$  groups of courses, called *curricula*, such that any two courses of a curriculum have students in common.

For course scheduling, each event (in this case each course)  $c_i$  consists of  $l_i$  lectures to be scheduled in distinct time periods, and it is attended by  $s_i$  students. As for the exam scheduling, each event (in this case each exam)  $e_i$  is also scheduled in time periods (which

does not necessarily be distinct all the time), but is attended by  $\sum s_i$  students from all  $l_i$  lectures

A given constraint for a scheduling problem can be classified either as a *hard constraint* or a *soft constraint*. A hard constraint must be absolutely met by a candidate solution in order to be feasible. An example is the Room Occupancy, where two distinct lectures cannot take place in the same room in the same period. On the other hand, it is not imperative that a solution satisfies a given soft constraint – they are desirable but not essential. However, these constraints evaluate the quality of a candidate solution. In short, they give a quantitative measure of the desirability of a generated schedule. An example of soft constraint is, the number of students that attend a course must be less or equal than the number of seats of all the rooms that host its lectures. The number and variety of constraints (hard or soft) existing in educational timetabling problems makes it impossible to list all of them [3]. An effective timetabling in academic institution is crucial for the satisfaction of educational requirements and efficient utilization of human and space resources [33].

## **B. Operations Research**

Also termed Operational research, or simply OR is an interdisciplinary science. Scientific methods like mathematical modeling, statistics, and algorithms to decision making are deployed in complex real world problems which are concerned with coordination and execution of the operations within an organization. The nature of organization is essentially

immaterial. The eventual intention behind using this science is to elicit a best possible solution to a problem scientifically, which improves or optimizes the performance of the organization [33].

Some of the primary tools used by operations researchers are statistics, optimization, stochastics, queueing theory, game theory, graph theory, and simulation. Because of the computational nature of these fields OR also has ties to computer science, and operations researchers regularly use custom-written or off-the-shelf software [33]

Areas of application include road traffic management, design and layout of computer chips, constructing a telecommunications network, scheduling, etc. [33].

### **C. Multi-objective Optimization**

The general multi objective optimization problem was described by Landa *et al* as follows:

$$\text{Minimize or Maximize } F(x) = (f_1(x), f_2(x), \dots, f_k(x)) \quad \text{s.t. } x \in S \quad (1)$$

where  $x$  is a solution,  $S$  is the set of feasible solutions,  $k$  is the number of objectives in the problem,  $F(x)$  is the image of  $x$  in the  $k$ -objective space and each  $f_i(x)$   $i = 1, \dots, k$  represents one (minimization or maximization) objective.



In many problems, the aim is to obtain the optimal arrangement of a group of discrete entities in such a way that the additional requirements and constraints (if they exist) are satisfied [34, 35].

Steuer described the three ways of combining the search and the decision-making processes [36] – the first decision that has to be made when dealing with a multi-objective optimization problem – and these are summarized as follows. In the *a priori approach*, decision making is done before the search. The preferences for each objective are set by the decision-makers and then, one or various solutions satisfying these preferences have to be found. The inverse is done in the *a posteriori approach*. Various solutions are found and then, the decision-makers select the most adequate. The solutions presented should represent a trade-off between the various objectives. In the last approach, the decision-makers intervene during the search in order to guide it towards promising solutions by adjusting the preferences in the process – **a decision-making with Interactive search**.

Another important decision is how to evaluate the quality of solutions, because the conflicting and incommensurable nature of some of the criteria makes this process more complex. There are several alternatives listed as follows: [37]

- (1) **Combine the objectives**. This is one of the “classical” methods to evaluate the solution fitness in multi-objective optimization. It refers to converting the multi-objective problem into a single-objective one by combining the various criteria into a

single scalar value. The most common way of doing this is by setting weights to each criterion and adds them all together using an aggregating function.

(2) **Alternating the objectives.** This is another “classical” approach. It refers to optimizing one criterion at a time while imposing constraints on the others. The difficulty here is on how to establish the ordering in which the criteria should be optimized, because this can have an effect on the success of the search.

(3) **Pareto-based evaluation.** In this approach, a vector containing all the objective values represents the solution Fitness and the concept of *dominance* is used to establish preference between solutions [36]. A solution  $x$  is said to be non-inferior or non-dominated if there is no other solution that is better than  $x$  in all the criteria. Suppose two distinct vectors  $V = (v_1, v_2, \dots, v_k)$  and  $U = (u_1, u_2, \dots, u_k)$  containing the objective values of two solutions for a  $k$ -objective minimization problem, then:

–  $V$  *strictly dominates*  $U$  if  $v_i < u_i$ , for  $i = 1, 2, \dots, k$ .

–  $V$  *loosely dominates*  $U$  if  $v_i \leq u_i$ , for  $i = 1, 2, \dots, k$  and  $v_i < u_i$ , for at least one  $i$ .

–  $V$  and  $U$  are *incomparable* if neither  $V$  (strictly or loosely) dominates  $U$  nor  $U$  (strictly or loosely) dominates  $V$ .

Minimization is considered here mainly because most of the scheduling problems are of this type (minimize processing time, minimize soft constraints violation, minimize

schedule length, etc.), but the above definition is altered in the obvious way for the case of maximization problems.

Landa *et al* noted that “using strict or loose dominance can have an effect on how the search is performed. This is because if a solution  $x_1$  is strictly dominated, it means that it is outperformed by the other solution  $x_2$  in all criteria. But, if the solution  $x_1$  is loosely dominated it means that it is outperformed in some of the criteria but it is as good as  $x_2$  in at least one of them. Then, finding a new solution that strictly dominates the current one may be more difficult than finding a solution that loosely dominates it” [3].

The aim in Pareto optimization is to find a set of compromise solutions that represent a good approximation to the Pareto optimal front [36, 39]. The Pareto optimal front is the set of all non-dominated solutions in the multi-objective space [36]. Pareto optimization refers to finding the Pareto optimal front or a set that represents a good approximation to that front. Pareto optimization is appealing because in most multi-objective optimization problems there is no such single-best solution and it is also very difficult to establish preferences among the criteria before the search. It has expressed that even if the conflicting nature of the criteria is not proved, Pareto-based metaheuristics would be able to find the ideal solution that is the best in all criteria [38].

#### D. Approaches to the University Timetabling Problem

A large number of diverse methods have been already proposed in the literature for solving timetabling problems. These methods come from a number of scientific disciplines like Operations Research, Artificial Intelligence, and Computational Intelligence [27, 39 – 43] and can be divided into four categories.

Sequential Methods treat timetabling problems as graph problems. After ordering the events with the use of domain-specific heuristics, they assign the events sequentially. Events are assigned into valid timeslots in such a way that no constraints are violated for each timeslot [44]. In 1967, Welsh and Powell [45] pointed out the similarity between timetabling problem and the one of colouring the vertices of a graph. Here, the vertices are taken to be equivalent to courses and the arcs between them represent conflicts. Colouring the graph amounts to placing courses in appropriate periods. The algorithm they present is similar to Broder's [46]. They order the vertices according to degree and attempt to colour the graph without using an upper limit on the number of colours. Since 1967 Welsh and Powell's observation has led to many timetabling algorithms based on graph colouring. Matula, Marble and Isaacson [47] in 1972 presented a *smallest degree last recursive* sequential algorithm. They also presented an interchange which involves looking for a colour swap in vertices adjacent to the one which is currently trying to be coloured when the normal method would introduce a new colour, adding limited search ability to the algorithm. A graph colouring algorithm is an integral part the system presented by Burke and Elliman [48] who have presented graph colouring and room allocation algorithm and show how the two can be

combined to provide the basis of a flexible and widely applicable timetabling system, and in some details, discussed how several common timetabling features can be handled within the system.

In the Cluster Method, problems are divided into a number of event sets. Each set is defined with the intention that it satisfies all hard constraints. These sets are then assigned to real timeslot, satisfying the soft constraints as well [49].

Another method, models the timetabling problem as a set of variables (events). Values or resources (such as teachers and rooms) have to be assigned to these events in order to satisfy a number of constraints. This method is referred to as Constraint Based Method [50]. E. Burke *et al* proposed an approach using case based heuristic selection concerning both university course time tabling and university exam timetabling, motivated by the goal of developing timetabling systems that are fundamentally more general than the current state of the art. Heuristic that worked well in previous similar situations are memorized in a case base and are retrieved for solving the problem in hand. It has been shown that case based reasoning can act effectively as an intelligent approach to learn which heuristics work well for particular timetabling problem [51]. Petrovic, Yang, Dror, Burke, MacCarthy, and Qu [52, 53] among others are those which proposed constraint based approach in solving timetabling problems.

The last method, such as genetic algorithms (GAs), simulated annealing, tabu search, and other heuristic approaches, is called Meta-Heuristics Methods. This method is mostly

inspired by nature, and such applies nature-like processes to solutions, in order to evolve them towards optimality [39 – 41, 54, 55].

Simulated annealing has been successfully applied to the timetabling problem in Swansea's TISSUE examinations scheduling system [11, 12].

### **E. Multi-objective Genetic Algorithm**

The basic principles of Genetic Algorithm (GA) were first proposed by Holland in 1970's. "*Genetic algorithms* are computerized search and optimization methods that work very similar to the principles of natural evolution". [56] These are based on Darwin's survival-of-the-fittest principles. Genetic algorithms are the most popular type of evolutionary algorithms. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure. In GA's, evolution starts from a population of completely random individuals and happens in generations. In each generation, the fitness and constraint values of the whole population are evaluated, multiple individuals are stochastically selected from the current population (based on their fitness and constraint values), modified (mutated or recombined) to form a new population, which becomes current in the next iteration of the algorithm [56].

Professor Kalyanmoy Deb stated on a short course introduction of GA that "GA's intelligent search procedure finds the best and fittest design solutions, which are otherwise

difficult to find using other techniques.” He also added that “GAs are attractive in engineering design and applications because they are easy to use and they are likely to find the *globally* best design or solution, which is superior to any other design or solution.” Aside from some of the GA applications – which include planning, job shop scheduling, pattern recognition, classification problems, neural network design, operations research and the like – GAs are also suitable for multi-objective optimal design problems, involving multiple objectives.

Voss et al. describe a metaheuristic as “*an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high quality solutions*” [57]. Many metaheuristics that were first applied to solve single-objective optimization problems have also been extended to multi-objective variants. Among these, multi-objective evolutionary algorithms have received particular attention because some researchers argue that these methods are well suited to deal with multi-objective optimization problems [54, 58].

Evolutionary algorithms refer to any population-based metaheuristic optimization algorithm that uses mechanisms inspired by biological evolution, such as inheritance, reproduction, mutation, crossover, natural selection and survival of the fittest. Candidate solutions are termed individuals in a population, and the cost function determines the fitness of a solution set. Evolution of the population then takes place after the repeated application of the above operators [59].

Over the years, there have been several approaches used to deal with problems having various objectives. A strategy, which generates the set of compromise solutions in a single execution of the algorithm – rather performing several searches using different preferences each time – has attracted the interest of researchers for investigating the application of Pareto optimization techniques to multi-objective scheduling problems [60 – 64]. The potential of multi-objective or multi-criterion algorithms (MOAs) in optimization problems has been explored by modern researches. These algorithms, which considers several (and at times conflicting) objectives simultaneously, are capable of generating multiple nearly optimal solutions and are powerful than traditional genetic algorithms since the former can implement the latter using just a single objective.

Though relatively young, research using MOA's show promising results for optimization and scheduling problems. Since the principal reason why a problem has a multi-objective formulation is because it is not possible to have a single solution which simultaneously optimizes all objectives, an algorithm that gives a large number of alternative solutions lying on or near the Pareto-optimal front is of great practical value.

#### **F. The Non-Dominated Sorting Genetic Algorithm II**

One of the first multi-objective algorithms was the Non-Dominated Sorting Genetic Algorithm II (NSGA-II). It incorporates the multi-objective approach in using genetic algorithms (GA's), which involves several generations having processes of evaluation,



stochastic selection and modification a population of completely random individuals, and in each generation, the fittest of the solutions are kept in a mating pool until the solutions converge to the Pareto-optimal front.

NSGA-II was proven, by Deb *et al*, to be faster than other multi-objective evolutionary algorithms, with time complexity of  $O(mN^2)$  where  $m$  is the number of objectives and  $N$  is the population size. NSGA-II, being a multi-objective genetic algorithm, is able to discern the fitness of a solution over an assortment of (sometimes conflicting) objectives, instead of using a singular fitness function characterized by weights and variables. It is also able to rank and generate a set of Pareto-optimal solutions, giving the user the best possible alternatives [65].

Simulation results on five difficult test problems show that the proposed fast, non-dominated NSGA-II is able to find much better spread of solutions in all problems compared to PAES (Pareto Archived Evolution Strategy)-another elitist multi-objective EA which pays special attention towards creating a diverse Pareto-optimal front [65].

## **G. Definition of Terms**

1. **Chromosome** – used to refer to a potential solution. It contains all of the necessary information needed to describe one solution.
2. **Clone** – when a duplicate of a chromosome is created;

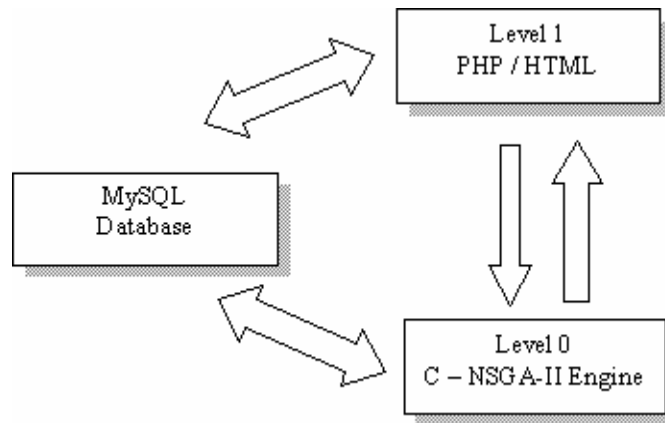
3. **Crossover** – a reproduction operator that create one or more new chromosomes by mixing their solutions.
4. **Elitism** – (or an elitist strategy) is a mechanism which ensures that the chromosome/s of the most highly fit member/s of the population are passed on to the next generation without being altered; ensures that the maximum fitness of the population can never reduce from one generation to the next.
5. **Evolution** – process of change which is assured given a reproductive Population in which there are varieties of Individuals, with some varieties being heritable, of which some varieties differ in fitness
6. **Fitness** – a value assigned to an Individual which reflects how well the individual solves the task in hand.
7. **Fitness Function** – a measure of the quality of a particular chromosome. chromosomes that are better solutions will have better fitness values than those that are less optimal solutions.
8. **Gene** – a subsection of a chromosome which (usually) encodes the value of a single parameter.
9. **Generation** – refers to one round of the Genetic Algorithm Cycle. New chromosomes are created and old ones are removed to make room for them.
10. **Individual** – a single member of a population.
11. **Mutation** – any modification made to the population or to a single Chromosome
12. **Parent** – an individual which takes part in reproduction to generate one or more other individuals, known as Offspring, or children.

13. **Penalty** – a part of the fitness function, it penalizes illegal or undesirable actions of the chromosome in the solution space.
14. **Population** – the collection of available chromosomes that encode the problem solutions. There is normally a limit on the size of the population, and those chromosomes that do poorly are eliminated to make room for better performing chromosomes.
15. **Reproduction** – the creation of a new Individual from two Parents (sexual reproduction). Asexual reproduction is the creation of a new individual from a single parent.

## CHAPTER IV

### DESIGN AND IMPLEMENTATION

Two-level architecture for CASS will be implemented. The highest level is a PHP/HTML user interface level that presents information to, and collects information from, the user. At the next level, a C program translates this information into a linear program (through the use of data structures as arrays of integers), which will then be solved with the help of the core of the system – the NSGA Engine, which will also be implemented in C. All of the data to be used throughout the levels will be stored and retrieved by a MySQL database. This architecture is shown below (in Figure 1).



**Figure 1: System Architecture, CAS Scheduler**

## A. The Algorithm

The genetic algorithm, can be summarized in the flowchart illustrated in Figure 2. The first step is to generate the initial population. Each member of this population will be encoded as a string (binary or not) – sometimes referred to as “*genotype*” or alternatively, a “*chromosome*” – of length  $L$ . These strings are then evaluated and are each assigned a *fitness value*.

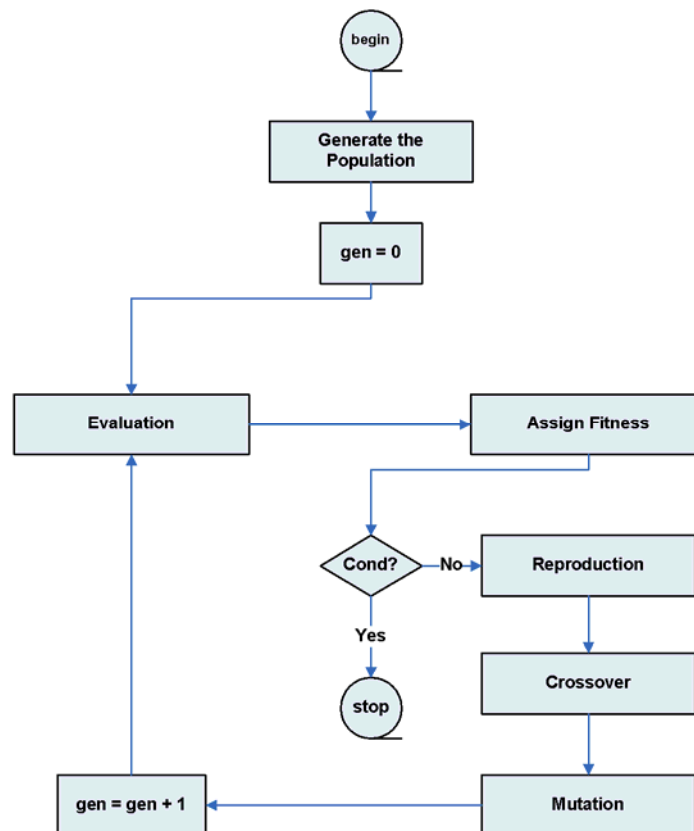


Figure 2: Flowchart of the Genetic Algorithm

In order to make use of the NSGA2 Engine designed by Deb *et al* in [65], the information gathered from all the users of the system (which are stored in MySQL database) will be translated and placed into a data structure of arrays of integers (and/or strings). These data structures will then serve as the encoded “*chromosomes*”. Each chromosome is divided into **n** sets, representing either the **n** sections (for course scheduling) or the **n** subjects (having final examination) to be scheduled. Each sets is divided into three parts – representing the timeslot, the room and the instructor, lecturer or proctor assigned to a section or an exam.

In figure 3, set 0 (colored blue in a), represents a section scheduled in timeslot 1, held at room 3 by instructor 4. The same thing goes to all the other sets (i.e. from set 1 to 3). As for the examination scheduling, set 3 (colored pink in b), represent a course with final exam scheduled during timeslot 1 at room 1. Again, same thing goes for all the other sets.



**Figure 3: Representation**

The evaluation function (objective function), is the measure of performance with respect to a particular set of parameters. The evaluation of a string *i* is independent of other strings. On the other hand, in the fitness function, a sting *i* is always defined with respect other members of the current population. The fitness function transforms the evaluation

function – the measure of performance – into an allocation of reproductive opportunities. It can also be assigned based on a string’s rank in the population or by sampling method of tournament selection.

*CASScheduler* makes use of several evaluation functions. A solution is feasible if it is devoid of conflicts within rooms, instructors, lecturers or proctors, and timeslots. If in case at least one of these conflicts arises, a penalty will be given to a particular candidate solution.

There is a **Room Conflict** if two or more sections are assigned to a same room  $i$ , at a certain timeslot  $j$ , or at overlapping timeslots. For all rooms  $\mathbf{R}$ , there should be no conflict within any of timeslots  $\mathbf{T}$ .

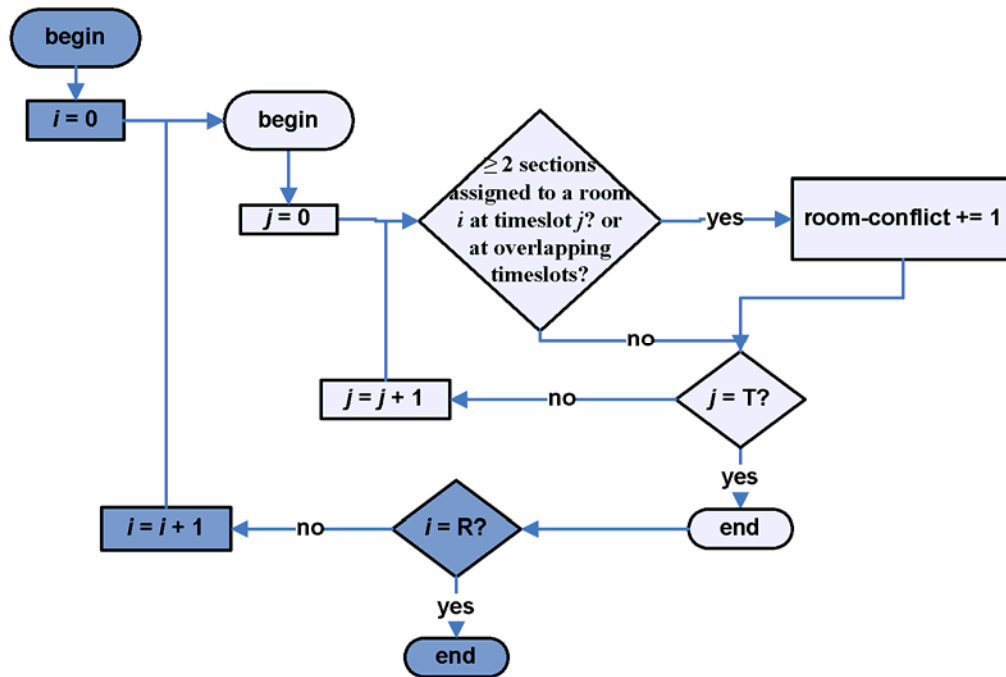


Figure 4: Room Conflict

On the other hand, an **Instructor Conflict** (also called Faculty conflict or Proctor Conflict) arises if an instructor is assigned to teach two distinct sections at overlapping timeslots. Additional penalty will be given if two or more sections are assigned to an Instructor  $i$  at timeslot  $j$ . Each of the  $I$  instructors are checked, to see if there is at least one conflict in the assignments of instructors to each section for all the timeslots  $T$ .

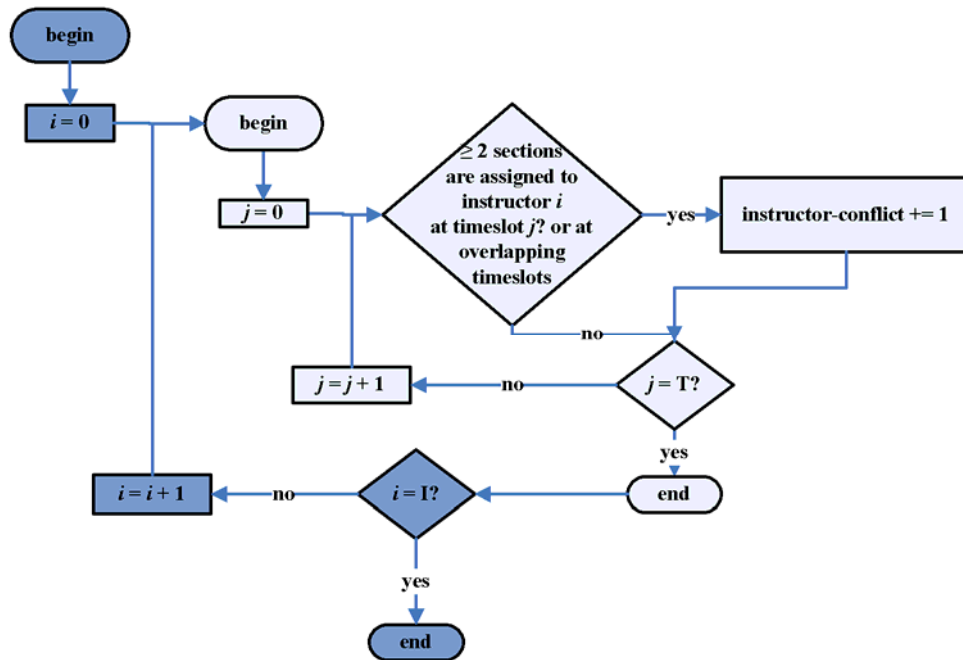
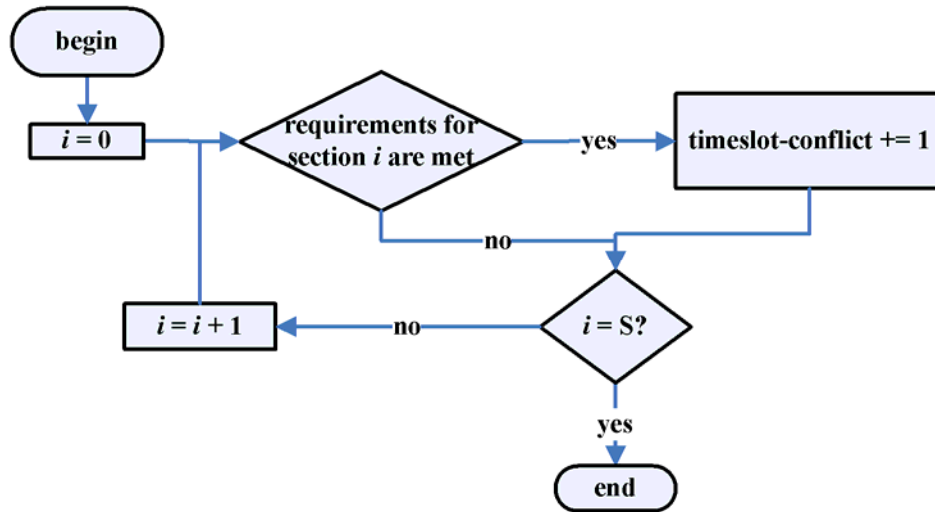


Figure 5: Instructor Conflict

Last among the hard constraints deals with timeslot compatibility; this will ensure that the lecture and laboratory hours (and examination hours, in the case of exam scheduling), and the number of meetings of each section are met by the timeslot to be assigned to it. A penalty would be given if **Timeslot Conflict** arise – if at least one of the necessary requirements of a section  $i$  is not met.





**Figure 6: Timeslot Conflict**

Fitness functions (the soft constraints) are minimization functions, which will be used to ensure the quality of the solution. A corresponding penalty will be added to fitness value whenever a soft constraint is violated. CASS focuses on three main fitness functions. First, a solution must conform with all policies implemented in the College of Arts and Sciences. As illustrated in Figures 7 and 8, a solution must have minimal (or better if no) invalid room assignments. Laboratory subjects must be designated to corresponding laboratory rooms, and subjects of lecture type must be held at lecture rooms (shown in Figure 8). And as is depicted in Figure 9, subjects must be held on their respective departments' rooms.

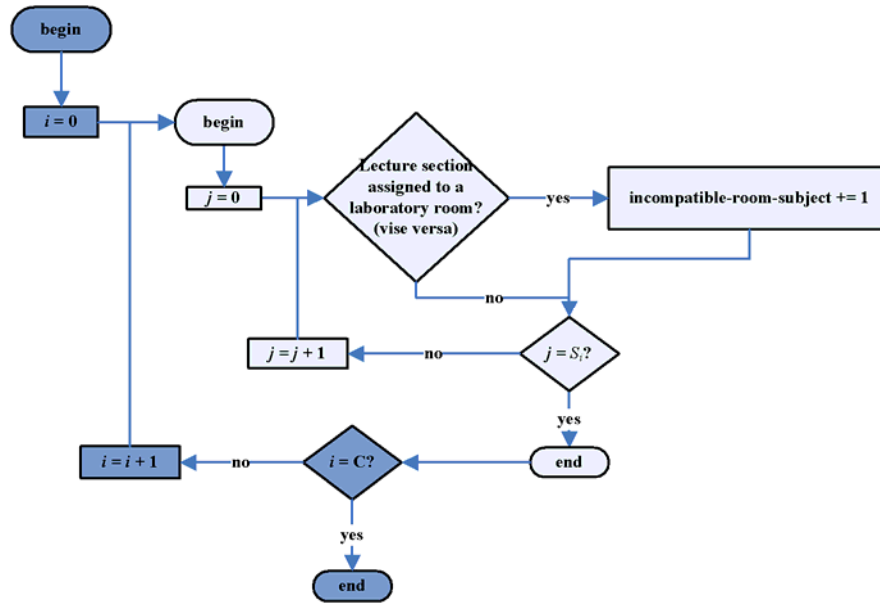


Figure 7: Incompatible Room to Subject Assignment

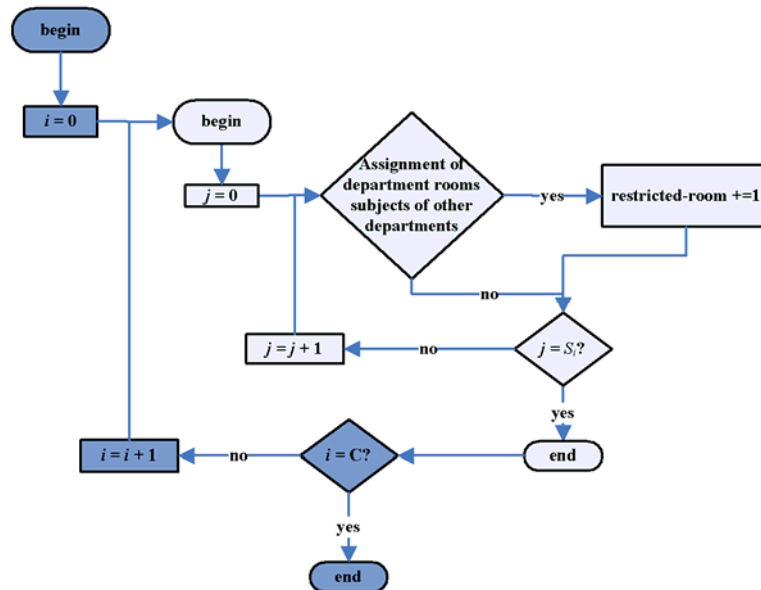


Figure 8: Restricted Room Assignment

Each candidate solution will also be evaluated in such a way that it will satisfy most of the instructors' preferences. A corresponding penalty if an instructor is assigned to any subject not among his expertise (shown in Figure 9), if any of the instructors time preferences

is not met (Figure 10), and if an instructor will have a load greater than the maximum (Figure 11).

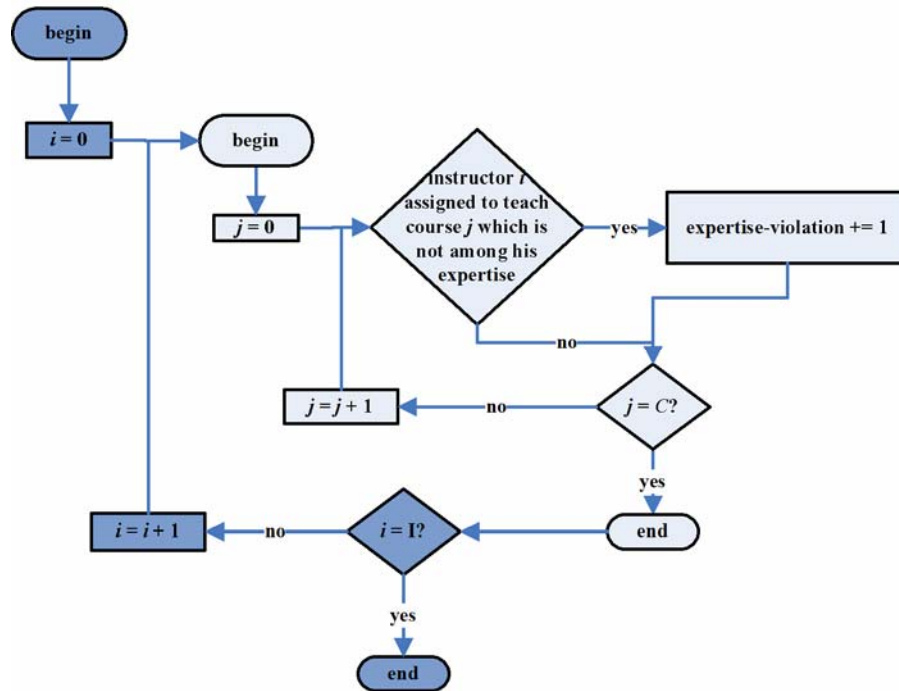


Figure 9: Teaching Expertise

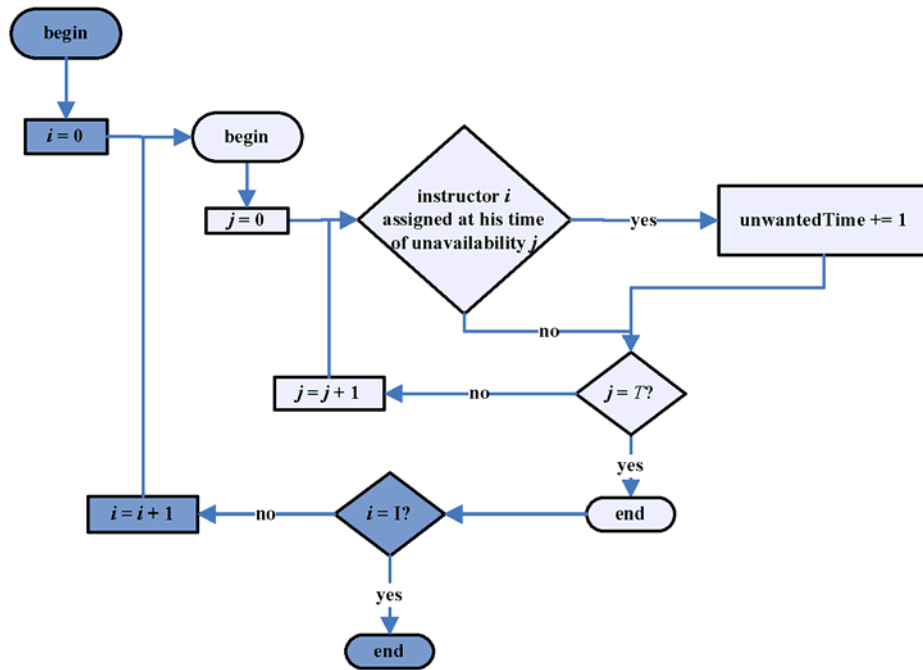


Figure 10: Time Preference

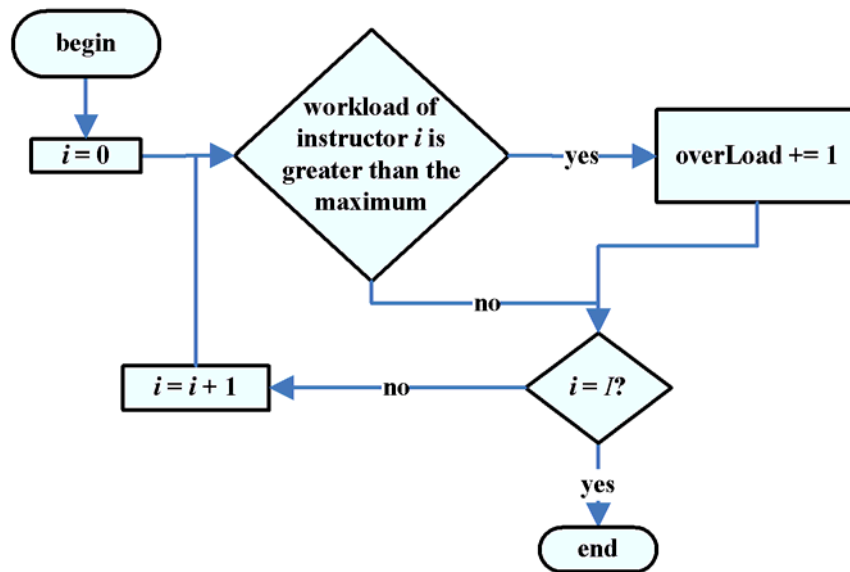


Figure 11: Workload

Last among the set of fitness functions implemented in CASS is such, that a candidate solution can also be “student-friendly” – that no two subjects taken by students of a certain course and a certain year level clashes, as depicted in Figure 12; and student demand for each subject must be satisfied, as shown in Figure 13. Doing so will enable each student to take all the subjects required in his curriculum for that semester; and the allocation of enough space for a student demand for each subject will minimize the addition of slots for the coming semester.

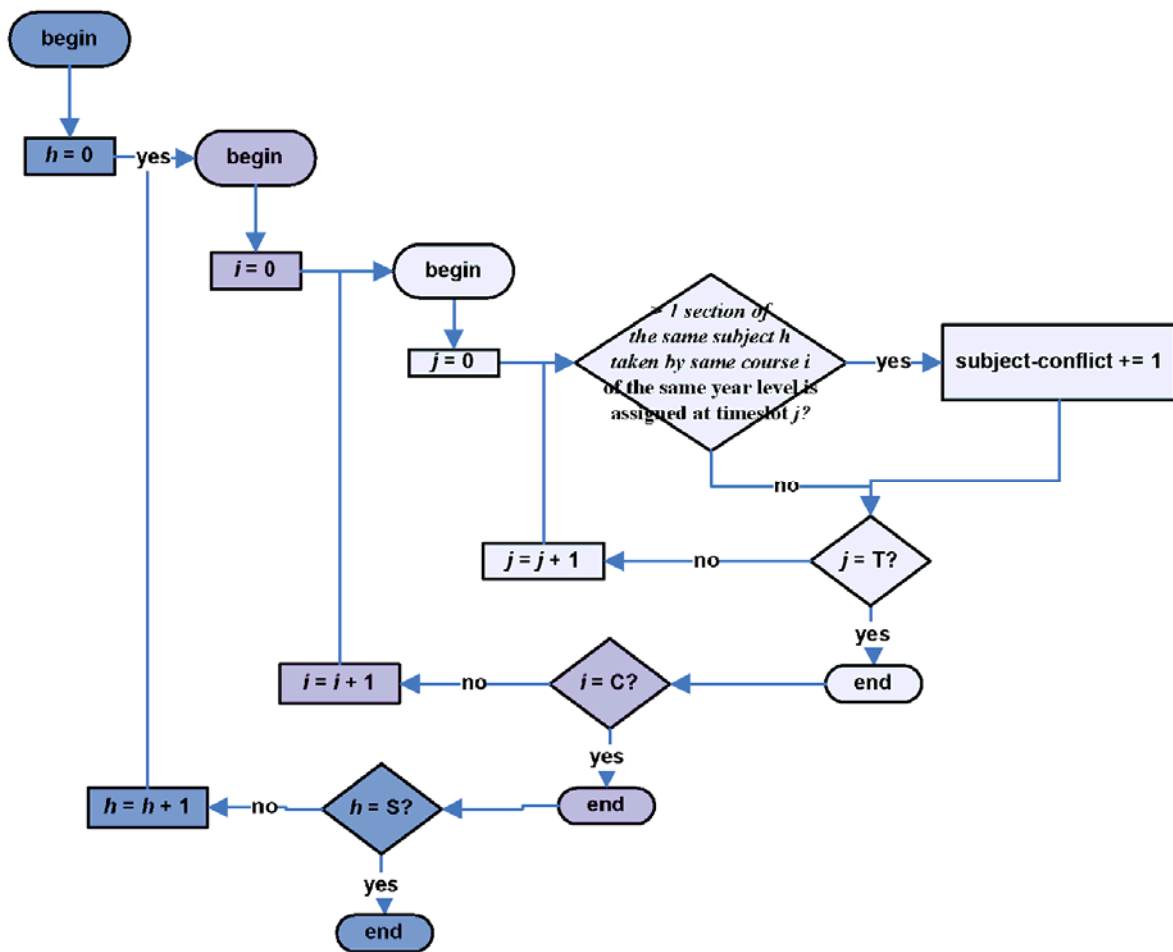
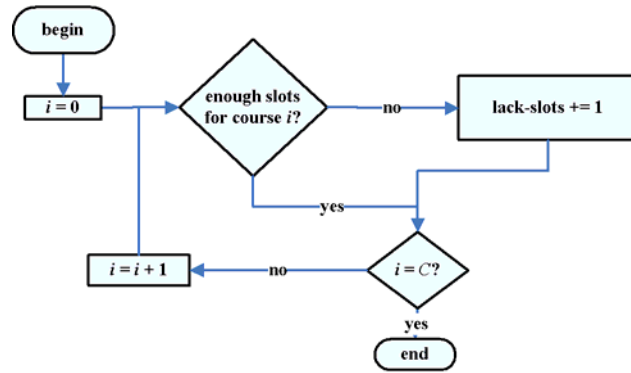


Figure 12: Co-requisite Subject Conflict



**Figure 13: Demand for Slots**

The execution of the algorithm can be viewed in a two stage process. Selection is applied to the current population to form the intermediate population, where the processes of recombination and/or mutation are applied to form the next population. This process – of going from the current population to the next population – constitutes a generation in the execution of the genetic algorithm. As again described in the previous chapter, Non-dominated sorting was the added feature of the NSGA-II designed by Deb *et al.*

Population will consist of the chromosomes described above which encode the problem solution. In the evaluation, corresponding penalty will be given to a solution and is described in the next section. A chromosome can mutate. Mutation happens when genes in a chromosome are combined in another way.

$$134 \mid 123 \mid 111 \mid 142 \rightarrow 133 \mid 124 \mid 111 \mid 142$$

Crossover causes recombination of genetic material of two chromosomes. It leads to rapid combination of patterns from different chromosomes.

$$\begin{array}{ccc}
 134 \mid 123 \mid 111 \mid 142 & \longrightarrow & 134 \mid 123 \mid 152 \mid 142 \\
 126 \mid 135 \mid 152 \mid 143 & & 126 \mid 135 \mid 111 \mid 143
 \end{array}$$

## B. The Entity Relationship

The Entity Relationship Diagram, as illustrated in Figure 14, summarizes the user interface's entities interaction. During a semester, each department offers at least one subject to at least one course (degree program), either belonging to the same department or not. One to many faculty member (instructor or lecturer), teaches at least one subject of his expertise. Also, each department may own rooms (of type lecture or laboratory) which is only exclusive for department use, however some rooms may be shared by all departments. An instructor handling a one of the sections of a subject at a certain room during a certain time consists a schedule. A schedule may be an accepted one (to be implemented for the coming semester) or may be just one of the candidate solutions (which resulted from the scheduling process).

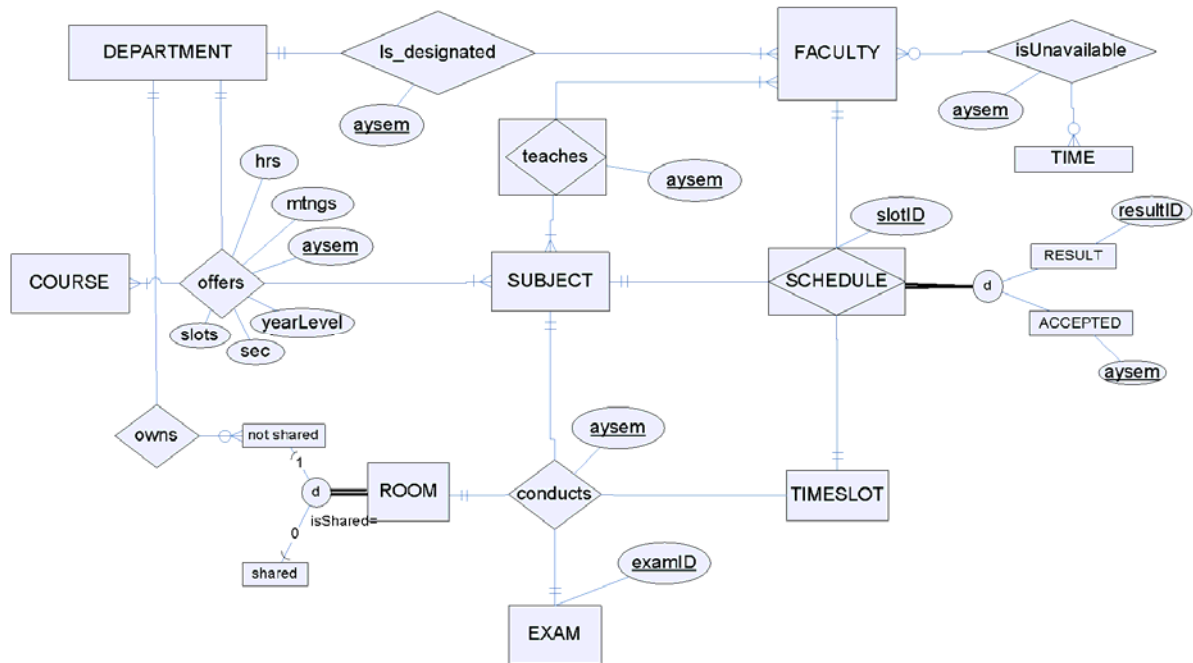


Figure 14: Entity Relationship Diagram, CASScheduler

The DEPARTMENT entity in Figure 15 represents a department under a CAS (College of Arts and Sciences). While the COURSE entity represents the a degree program under a department (Computer Science, Biology, Political Science and the like);

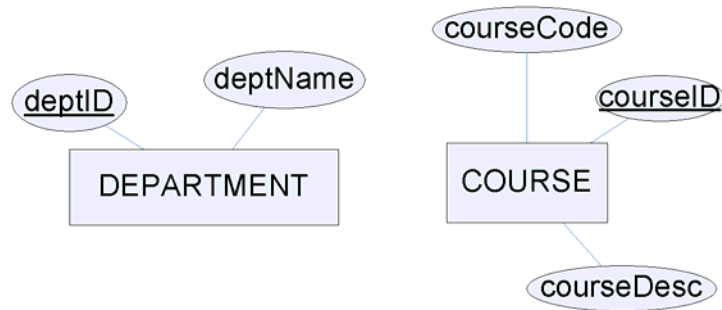


Figure 15: Department and Course Entity with Attributes, *CASScheduler*

Figure 16 shows numerous attributes of the SUBJECT entity. This entity represents a subject offered by a department taken by various students. The ROOM entity on the other hand represents a room where subjects are held. FACULTY entity (illustrated in Figure 17) represents an instructor, lecturer or a proctor designated to a department.

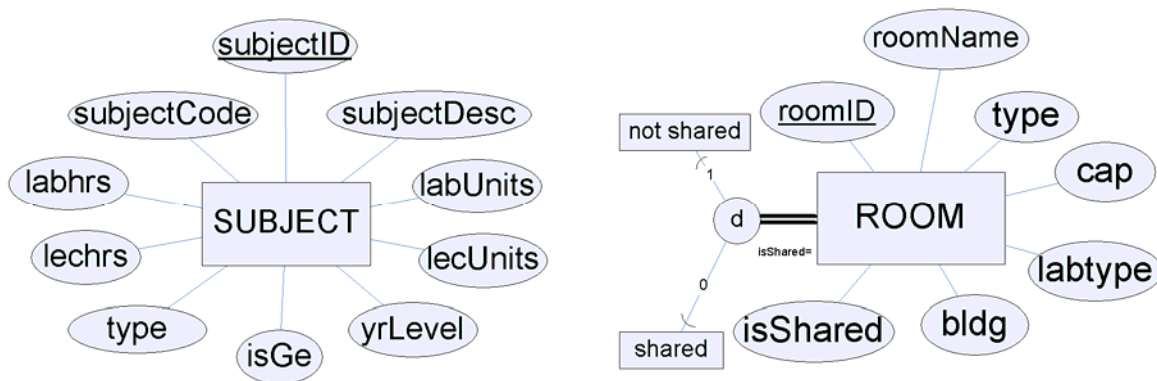


Figure 16: Subject and Room Entity with Attributes, *CASScheduler*



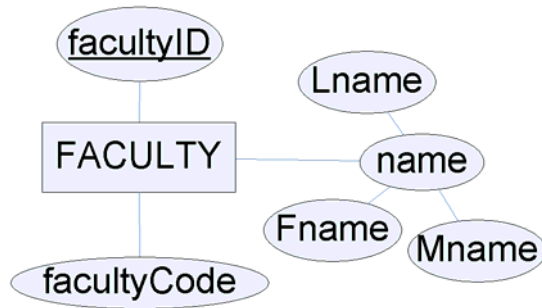


Figure 17: Faculty Entity with Attributes, *CASScheduler*

The TIME entity represents the time of unavailability of an instructor. TIMESLOT entity on the other hand represents the time when a scheduled subject can be held.

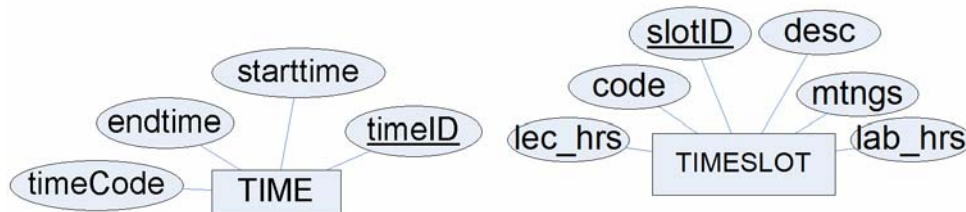


Figure 18: Time and Timeslot Entity, *CASScheduler*

### C. The Data Dictionary

The following tables show how the different entities will be represented as tables in the system's database.

**DEPARTMENTS** – table that stores the college's departments

Field Name	Type	Description
*deptID	int	identification (id) of a department.
deptName	varchar	department name

**ROOMS** – table that stores the lecture rooms and laboratory rooms of the college

Field Name	Type	Description
*roomID	int	identification (id) of the room
bldg	varchar	building where the room is located.
roomName	varchar	name of the room
type	int	room type, i.e. lecture, laboratory or both.
labtype	int	laboratory room type i.e. Chemistry, Physics, etc.
cap	int	number of persons a room can accommodate – capacity.
isShared	int	determines if a room can be shared across departments.
^deptID	int	identification of the department where the room belongs
status	int	tells if the room is active (still exists) or not.

**SEM ROOMS** – table that stores the rooms to be used for the semester

Field Name	Type	Description
*aysem	int	academic year and semester
*roomID	int	identification (id) of the room

**COURSES** – table that stores the degree programs within the college

Field Name	Type	Description
*courseID	int	identification (id) of the course
courseCode	int	course code known to its department.
courseDesc	varchar	description of the course
^deptID	varchar	identification (id) of the department handling the course
status	int	tells if the course is active (still exists) or not.

**SUBJECTS** – table that stores all the subjects the college offers

Field Name	Type	Description
*subjectID	int	academic year and semester
subjectCode	varchar	subject code known to its department.
subjectDesc	varchar	description of the subject
type	int	subject type – lecture, laboratory or both
^labtype	int	laboratory type – Chemistry, Computer, etc.
lecUnits	int	number of lecture units the subject has
labUnits	int	number of laboratory units the subject has
lehrs	float	number of hours (in a week) for the lecture part of the subject

isGE	float	number of hours (in a week) for the laboratory part of the subject
^deptID	int	identification (id) of the department handling the subject
status	int	tells if the course is active (still exists) or not.

**SEM\_SUBJECTS** – table that stores all the subjects the college offers

Field Name	Type	Description
*aysem	int	academic year and semester
*courseID	int	identification (id) of the course taking the subject
*yrLevel	int	year level taking the subject
*subjectID	int	academic year and semester
*type	int	subject type – lecture, laboratory or both
^labtype	int	laboratory type – Chemistry, Computer, etc.
^deptID	int	identification (id) of the department handling the subject
sec	int	number of sections allotted for the course and year level taking the subject
slots	int	number of slots allotted for each section
mtgs	int	number of time each section meets
hrs	float	number of hours (in a week) needed for the subject

**FACULTY** – table that stores the instructors, lecturers and/or proctors

Field Name	Type	Description
*facultyID	int	identification (id) of the course
^deptID	int	identification (id) of the department handling the course
lname	varchar	surname or last name of the faculty
fname	varchar	given or first name of the faculty
mname	varchar	middle name of the faculty

**SEM\_FACULTY** – table which stores the instructors with loads for the semester

Field Name	Type	Description
*aysem	int	academic year and semester
*^facultyID	int	identification (id) of the faculty

**FACULTY\_SUBJECTS** – table which stores instructors and the subjects they teach

Field Name	Type	Description
*aysem	int	academic year and semester
*^facultyID	int	identification (id) of the faculty
*^subjectID	int	identification of the subject
type	int	subject type

**FACULTY\_UNAV** – table that stores the time unavailability of the instructors

Field Name	Type	Description
*aysem	int	academic year and semester
*^facultyID	int	identification (id) of the faculty
*^stimeID	int	start time when the instructor is unavailable

*^etimeID	int	end time when the instructor is unavailable
*^dayID	int	day when the instructor is unavailable

**HALFTIME** – table that stores the half times from 7:00 am to 8:30 pm.

Field Name	Type	Description
*timeID	int	identification (id) of the time
timeCode	int	time code indicating the days and time
startTime	int	start time
endTime	int	end time

**DAYSPOSS** – table that stores possible days when a subject may be scheduled.

Field Name	Type	Description
*dayID	int	identification (id) of the time
dayCode	int	time code used for naming sections (M,T,Th,..MTh...S)
mtgsperwk	int	number of meetings per week

**SECTIONS** – table which store all the possible timeslots (for naming section).

Field Name	Type	Description
*sectionID	int	identification (id) of the section
sectionCode	int	section code / section name
dayID	int	identification (id) of the day/s when the section meets
starttime	int	when timeslot starts
endtime	int	when timeslot ends
nhrs	float	number of hours the timeslot has

**SUBJECT\_RESULTS** – table that stores optimized subject scheduling results

Field Name	Type	Description
*resultID	int	An identification (id) of the result (1 being the best result)
^subjectID	int	An identification (id) of the subject
^type	int	subject type – lecture, or laboratory
^sectionID	int	An identification (id) of the section
^roomID	int	The room where the course section will be held
^facultyID	int	The faculty who will be teaching the class
conflict	int	Non zero if conflict arises.

**EXAM\_RESULTS** – table that stores optimized exam scheduling results

Field Name	Type	Description
*resultID	int	An identification (id) of the result
^subjectID	int	An identification (id) of the subject
^type	int	subject type – lecture, or laboratory
^timeID	int	An identification (id) of the section
^roomID	int	The room where the course section will be held
conflict	int	Non zero if conflict arises.

**SUBJECT\_SCHEDULES** – table of accepted or implemented subject schedules

Field Name	Type	Description
*aysem	int	academic year and semester
*^subjectID	int	An identification (id) of the subject
*^type	int	subject type – lecture, or laboratory
*^sectionID	int	An identification (id) of the section
*^roomID	int	The room where the course section will be held
*^facultyID	int	The faculty who will be teaching the class
conflict	int	Non zero if conflict arises.

**EXAM\_SCHEDULES** – table of accepted or implemented final exam schedules

Field Name	Type	Description
*aysem	int	academic year and semester
*^subjectID	int	An identification (id) of the subject
*^type	int	subject type – lecture, or laboratory
*^timeID	int	An identification (id) of the section
*^roomID	int	The room where the course section will be held
conflict	int	Non zero if conflict arises.

**USER** – table which stores the system users

Field Name	Type	Description
*username	varchar	unique name identifying the a user
password	varchar	password associated with the user
usertype	int	user type which determines the privileges
lname	varchar	surname or last name of the user
fname	varchar	given or first name of the user
mname	varchar	middle name of the user
deptID	int	identification (id) of the department where the user belongs
facultyID	int	identification (id) of the faculty if user is a faculty

---

\*Primary key

^Foreign Key

## D. The Data Flow

Figure 19 illustrates the Context Diagram of CASS. It represents the overall interactions between the users of the system. *CASScheduler* has three users – the instructors / lecturers, the department chairman, and OCS personnel/s who may also serve as the system administrator. Each will input information for the system to process and each may see the results which the system will output. Figures 20-24 illustrated the flow of data on the proposed system.

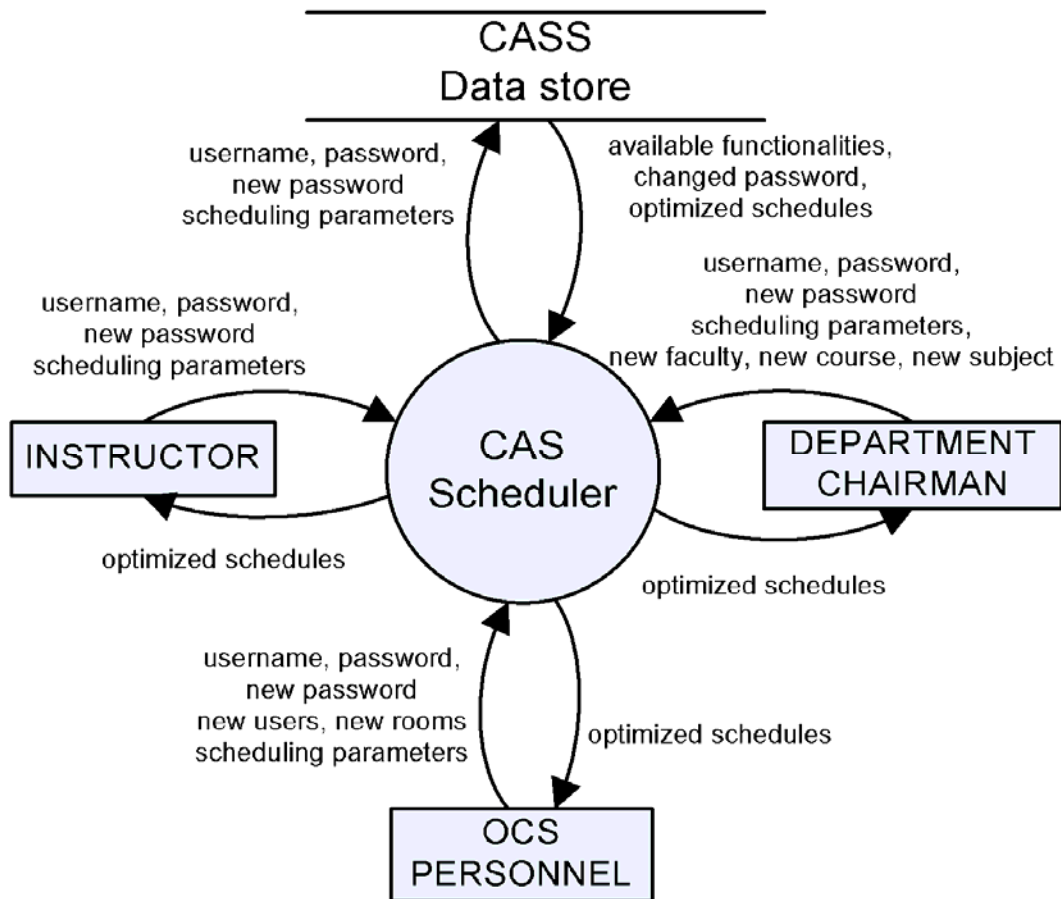
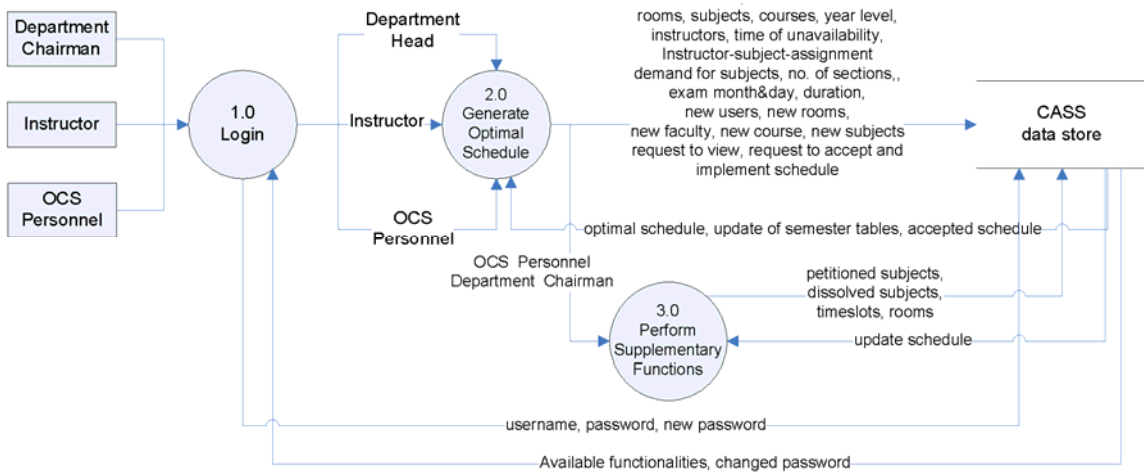
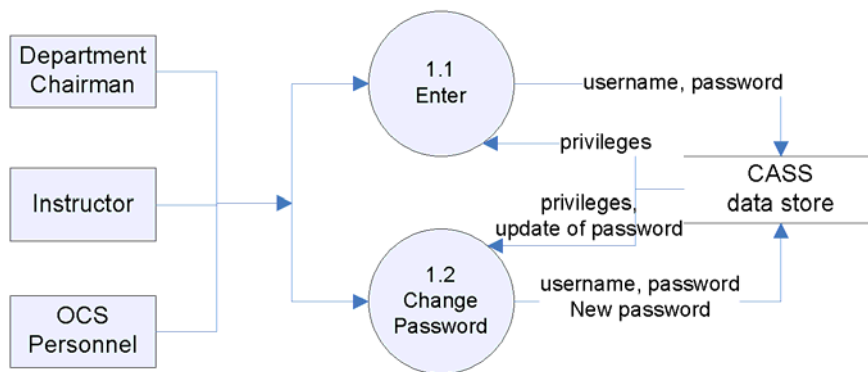


Figure 19: Context Diagram, *CASScheduler*

The user interacts with the system by entering all the scheduling parameters, view the generated and accepted results, or (for users with Administrator privileges) do some modifications to the accepted schedule. These are illustrated in the top level data flow diagram on Figure 20. Upon logging into the system, the user will also have the option to change his password as seen in Figure 21.



**Figure 20: Top Level - Data flow Diagram, *CASScheduler***



**Figure 21: Sub-Explosion of Login, *CASScheduler***

The generation of optimal schedule is comprised of the processes shown in Figure 22. User inputs stored in the data store, are processed and passed to the NSGA-II Engine, being the brain of the system, once a user request (to generate schedule) is made. Solutions generated, are then stored to the data store. Figure 23 shows the basic processes of the scheduling run (Perform Scheduling with NSGA-II Engine).

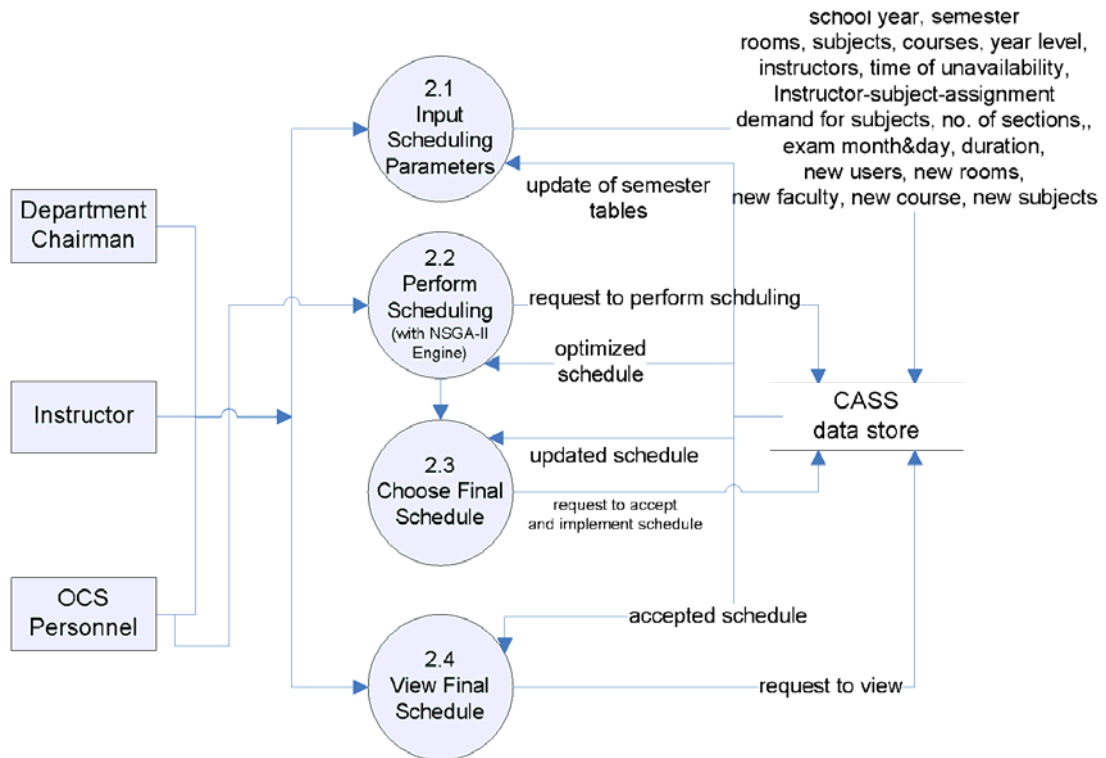


Figure 22: Sub-Explosion of Generate Optimal Schedule, *CASScheduler*



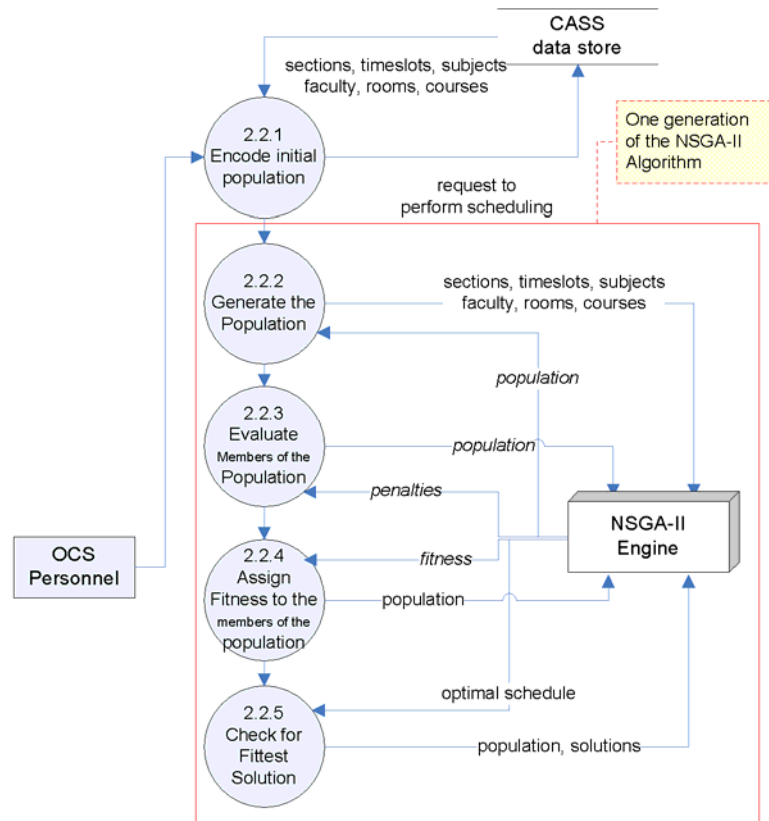


Figure 23: Sub-Explosion of Perform Scheduling, *CASScheduler*

The algorithm is run for several generations, and during its run, it involves chromosome operations. These are depicted in Figures 24 – 25.

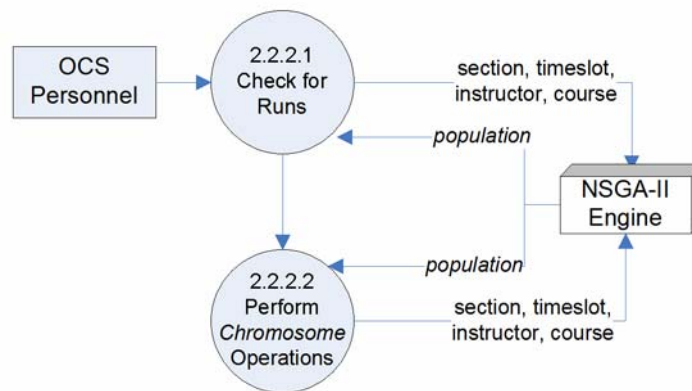
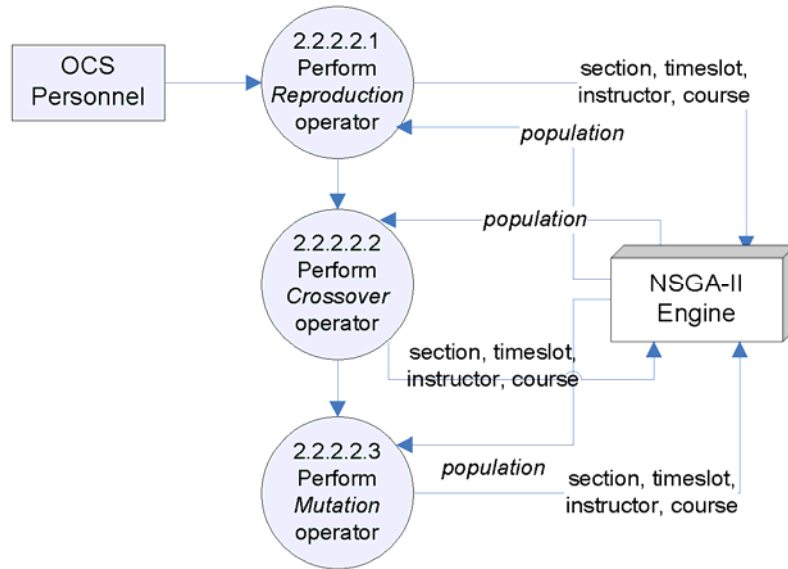


Figure 24: Sub-Explosion of Generate the Population, *CASScheduler*



**Figure 25: Explosion of Perform Chromosome Operations, *CASScheduler***

Supplementary functions are done prior to or after performing the scheduling procedure as shown in Figures 26-28 and 32-33. System users, rooms, course, faculty and subjects must exist first before scheduling can take place, or even before the gathering of scheduling parameters. Such entities may be updated or deleted whenever necessary.

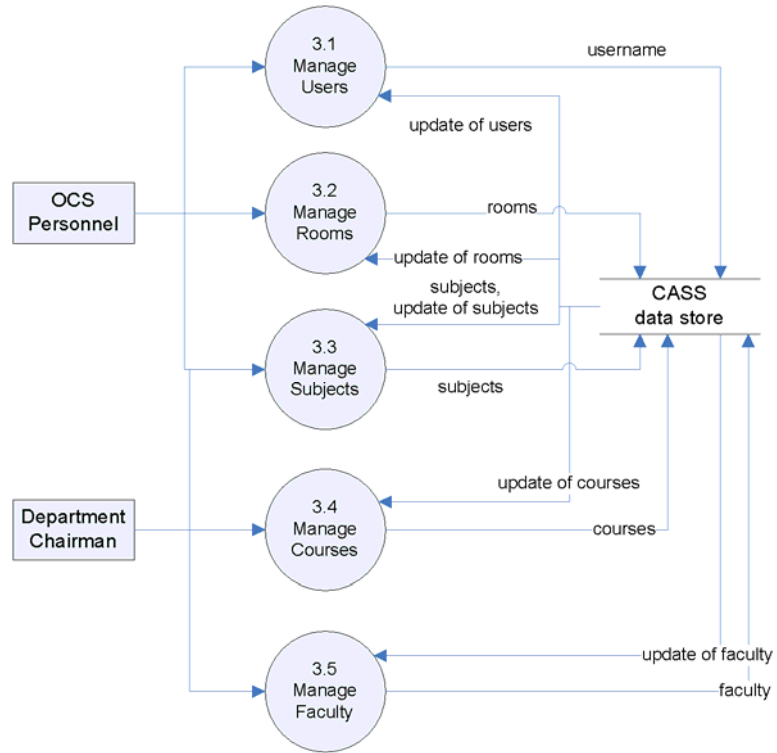


Figure 26: Sub-Explosion of Perform Supplementary Functions, *CASScheduler*

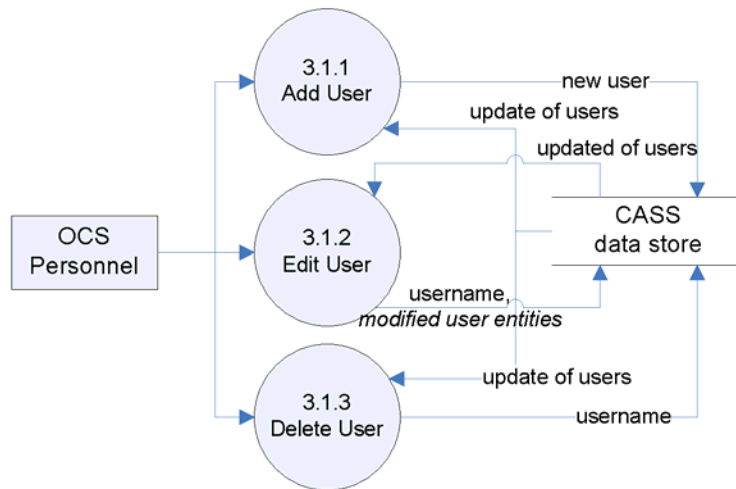
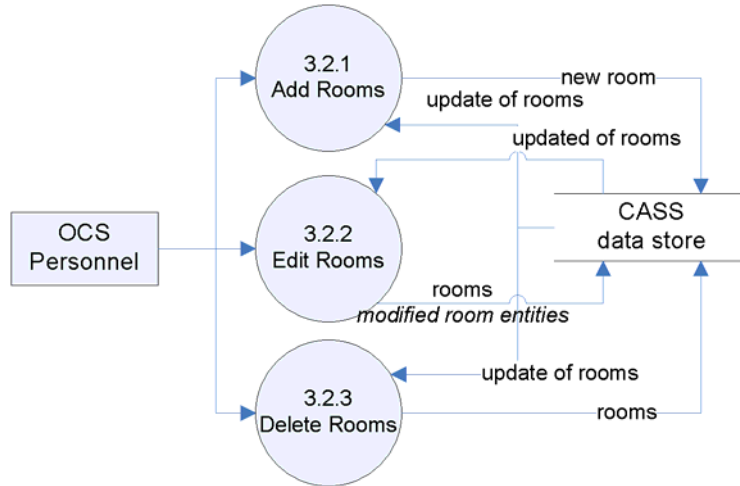
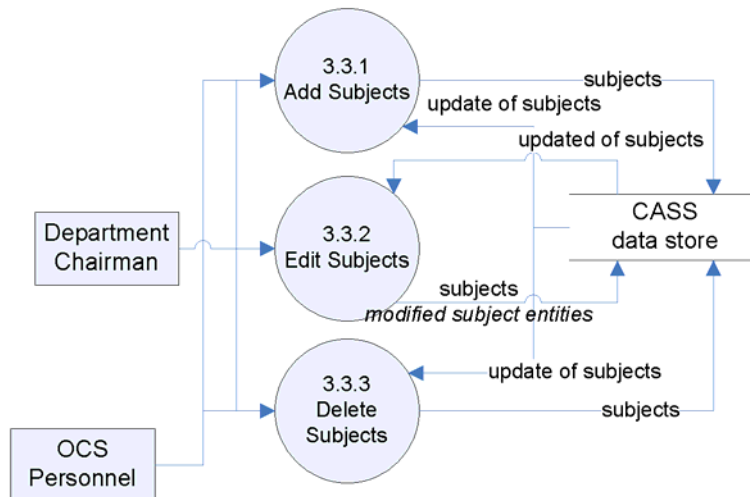


Figure 27: Sub-Explosion of Manage Users, *CASScheduler*

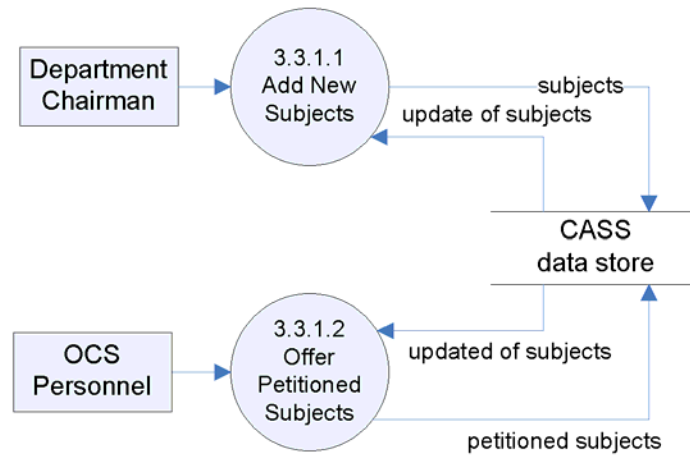


**Figure 28: Sub-Explosion of Manage Rooms, *CASScheduler***

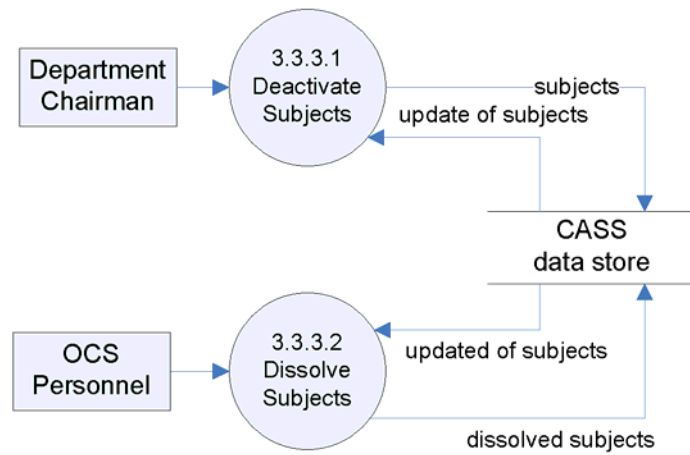
There are instances when there is a need to dissolve an offered subject or offer petitioned subjects. These processes are comprised in the management of subjects. A user having Department Head privileges) adds all the subjects being offered under his department. (see Figure 29-31)



**Figure 29: Sub-Explosion of Manage Subjects, *CASScheduler***

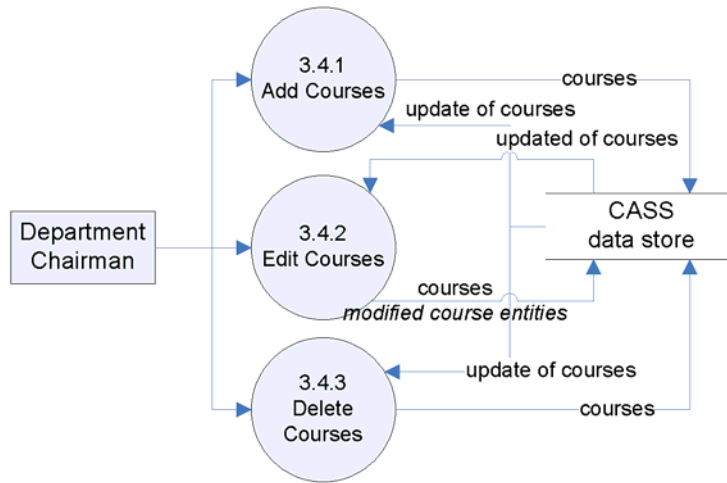


**Figure 30: Sub-Explosion of Add Subjects, *CASScheduler***

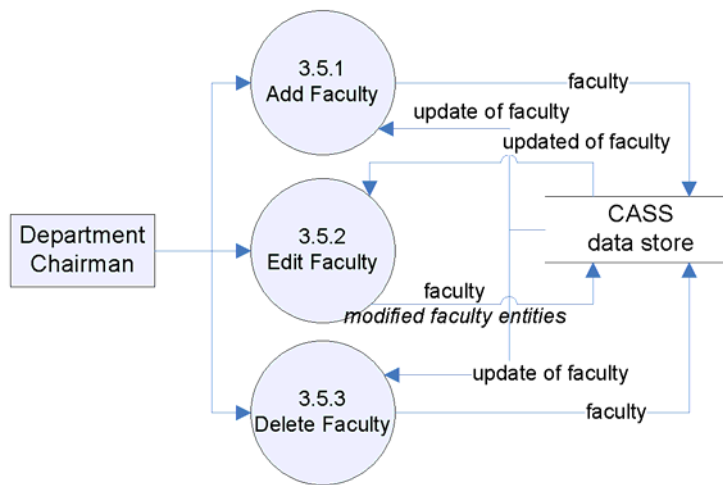


**Figure 31: Sub-Explosion of Delete Subjects, *CASScheduler***

The management of courses and faculty members are also part of the supplementary functions accessible to OCS Personnel users and Department Head users. These are shown in Figures 32- 33.



**Figure 32; Sub-Explosion of process Manage Courses, *CASScheduler***



**Figure 33: Sub-Explosion of Manage Faculty, *CASScheduler***

## E. Technical Architecture

*CASScheduler* makes use of a client-server model, where computer clients request services provided by from computer servers [66]. Particularly LAMP (Linux-Apache-MySQL-PHP) software bundle was used in the development of the system. “This technology allows the user of a web browser to execute a program on the web server and to thereby receive dynamic as well as static content”. [67] Also, it offers completely open source development stack that is lightweight, inexpensive, highly efficient and easy to use

The system was developed and configured using Linux operating system – particularly, Ubuntu. The “next generation of the omnipotent Apache web server” was used. Being a total rewrite, version 2 introduces many new improvements, which includes threading, request responsive filtering and more [68].

As for the data repository of *CASScheduler*, the DBMS (database management system) used was MySQL, which is a “fast, stable and true multi-user, multi-threaded SQL database server”, and of which speed, robustness and ease of use is the main goal.

Among the middleware languages, PHP – an HTML-embedded scripting language – with the goal to allow web developers to write dynamically generated pages quickly, was chosen for the interpretation of the requests.

## CHAPTER V

### RESULTS

Shown below in Figure 34, is the homepage of *CASScheduler*. Positioned on the upper right is where the users login or change their password, as they login.



The screenshot displays the homepage of the *CASScheduler* application. The header features the University of the Philippines - Manila logo on the left, the *CASScheduler* title in a stylized font in the center, and the College of Arts and Sciences logo on the right. Below the header, the main content area contains a login and password management form. The form includes four input fields: Username, Password, New Password, and Verify new password. Below these fields are three buttons: Login, Apply Changes & Login, and Reset.

**Figure 34: Index Page, *CASScheduler***

Users of *CASScheduler* are of three types, first are the heads of each department; another are the instructors or lecturers; and last is/are the OCS Personnel/s. Each of their functionalities differs and is shown at the top of their homepages as they login.



Prior to the generation of schedules, each of the department heads has to decide on what subjects their department will offer, and assign who among their faculty members is eligible in teaching each subject.

In designing the curriculum users having department head privileges may choose to either modify or apply the default curriculum design for a semester. Subjects to be offered, may be chosen from the subject dropdown box. This is seen in Figure 35.



Figure 35: Curriculum Design, *CASScheduler*

Figure 36 shows how the users (OCS Personnel or department head) may edit subject settings – of which duration and meetings per week, number of sections and

number of slots, per course, and year level may be modified. Only the subjects offered or are present in the curriculum of the chosen course and year level is shown and may

University of the Philippines - Manila **CASScheduler** College of Arts and Sciences

Welcome dpsmH, Today is Saturday - March 10, 2007.

[Curricula](#) [Subjects](#) [Instructors](#) [View](#) [Miscellaneous](#) [Logout](#)

### Subject Settings

\*School Year: 2007 - 2008 \*Semester: Second Semester  
 \*Course: BS Biochemistry \*Year Level: 2nd Year

Subject	Class Duration	Exam Duration	Meetings	Sections	Slots	Change Settings
COMMII - Lec	3 : 15	2 : 15	2	1	20	Save Settings
NATSCI5 - Lec	3 : 15	2 : 15	2	1	20	Change Settings
CHEM14 - Lec	3 : 15	2 : 15	2	1	20	Change Settings
CHEM14.1 - Lab	3 : 15	2 : 15	1	1	20	Change Settings
GEO11 - Lec	3 : 15	2 : 15	2	1	20	Change Settings
MATH73 - Lec	3 : 15	2 : 15	2	1	20	Change Settings
MATH73 - Lab	3 : 15	2 : 15	1	1	20	Change Settings

*Duration - number of h:mm per week*  
*Meetings - number of meetings per week*  
*Sections - total number of sections (per course and year level)*  
*Slots - number of slots per section*

Figure 36: Subject Settings, CASScheduler

be edited. The only difference between the two user type's functionality is that, OCS personnel users may modify any department's subject settings, while the latter cannot.

Instructors on leave must also be inputted in the system, to know which among the faculty members of each department may be assigned at least one subject for the semester (see Figure 37).

The screenshot shows the CASScheduler interface. At the top, there is a header with the University of the Philippines - Manila logo on the left, the text 'College of Arts and Sciences' on the right, and the 'CASScheduler' logo in the center. Below the header, a navigation menu includes links for 'Curricula', 'Subjects', 'Instructors' (which is highlighted), 'Miscellaneous', and 'Logout'. Underneath, there are sub-links for 'On Leave', 'Subject Assignments', and 'Unavailability'. The main heading is 'Instructors-on-Leave'. A red message states: 'You already have set faculty on leave for this semester. Would you like to save the changes you made?'. Below this, there are two dropdown menus: 'School Year' set to '2007 - 2008' and 'Semester' set to 'Second Semester'. An 'Instructors' label is followed by a list box containing the names: Solis, Sy, Tuaño, Baes, Bagnol, and Billones, L. At the bottom of the list box are 'Yes' and 'No' buttons.

Figure 37: Faculty-on-Leave, CASScheduler

It is the department head who knows more than anyone else what his subordinates' expertise are. Hence before the scheduling procedure takes place, he must also provide the system of which among the subjects offered by his department can each of his instructors can teach (see Figure 38).

University of the Philippines - Manila

College of Arts and Sciences

CASScheduler

Welcome dpsmH, Today is Saturday - March 10, 2007.

[Curricula](#)
[Subjects](#)
[Instructors](#)
[View](#)
[Miscellaneous](#)
[Logout](#)

[On Leave](#)
[Subject Assignments](#)
[Unavailability](#)

### Faculty -- Subjects

**School Year** 2007 - 2008  
**Semester** Second Semester

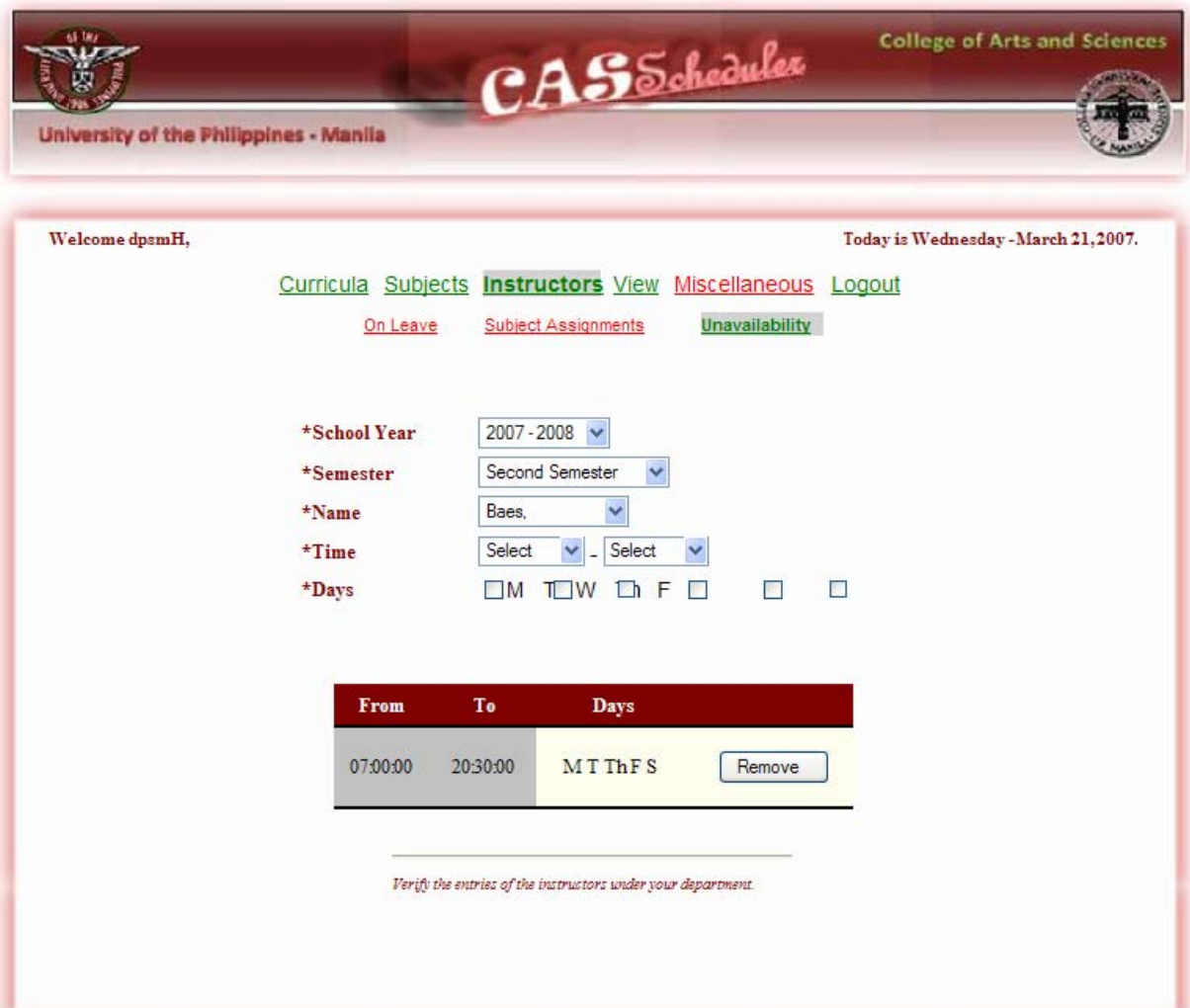
Faculty	Subjects
Bastero,	MATH73 - Lec
Carpio,	MATH73 - Lab
Chua,	CHEM27 - Lec
Co,	CHEM27.1 - Lab
DelaCruz,	MATH101 - Lec
Gonzaga,	MATH101 - Lab
Llarenas,	MATH75 - Lec
Magboo,MS,	MATH75 - Lab
Magboo,VP,	PHYSICS72 - Lec
Nanggan,	PHYSICS72.1 - Lab

Yes No

*Select the subjects the faculty member can teach.*

**Figure 38: Faculty-Subjects, CASScheduler**

Instructors may also input their time of unavailability as shown in Figure 39. Their respective department heads may verify their entries and may modify the inputs made if they think this functionality has been abused.



Welcome dpsmH, Today is Wednesday - March 21, 2007.

[Curricula](#) [Subjects](#) [Instructors View](#) [Miscellaneous](#) [Logout](#)  
[On Leave](#) [Subject Assignments](#) [Unavailability](#)

\*School Year: 2007 - 2008  
 \*Semester: Second Semester  
 \*Name: Baes.  
 \*Time: Select - Select  
 \*Days:  M  T  W  Th  F

From	To	Days	
07:00:00	20:30:00	M T Th F S	<input type="button" value="Remove"/>

*Verify the entries of the instructors under your department.*

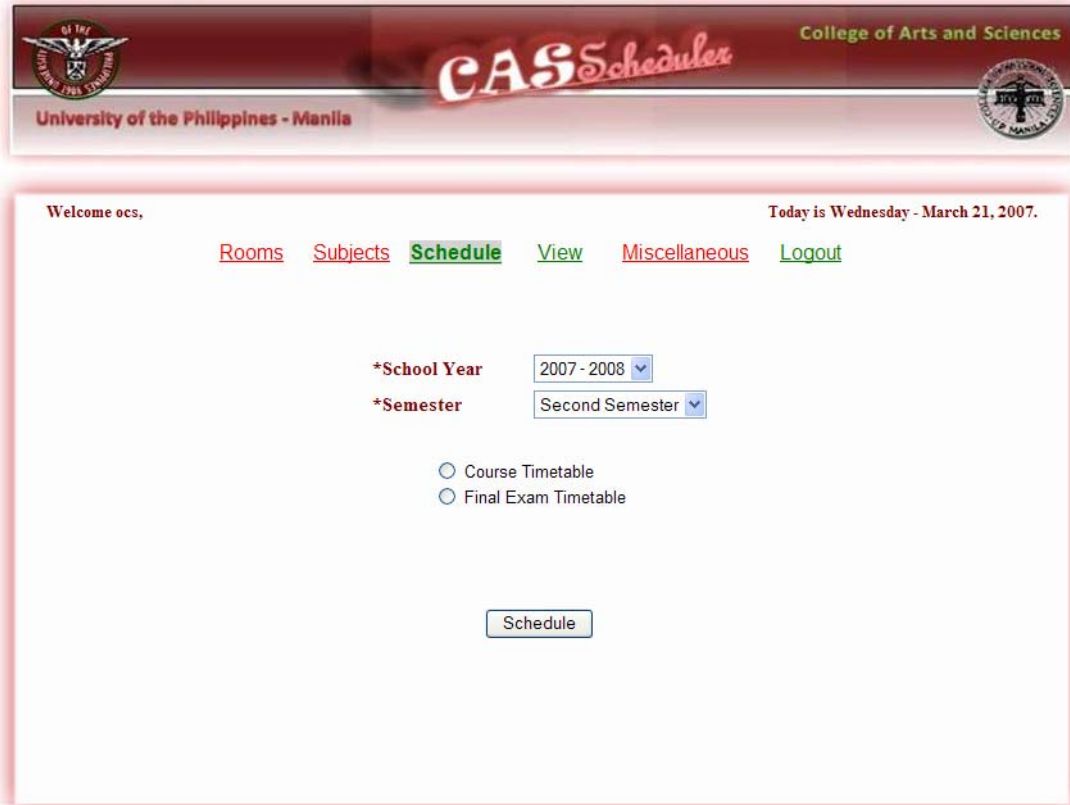
**Figure 39: Time of Unavailability, CASScheduler**

For some time, a room cannot be used due to some reasons. For instance, it may be under construction, or has been reserved to serve as some other purpose. Taking this into consideration, the system asks for which among the rooms are restricted for the semester that is about to be scheduled as illustrated in Figure 40.

The screenshot shows the CASScheduler interface for the University of the Philippines - Manila, College of Arts and Sciences. The page title is "Restrict Rooms". It features a navigation menu with links for "Rooms", "Subjects", "Schedule", "View", "Miscellaneous", and "Logout". The main content area includes several dropdown menus for "School Year" (set to 2007-2008), "Semester" (set to First Semester), and "Building" (set to All). Below these is a list of rooms: GAB-101, GAB-102, GAB-103, GAB-104, GAB-105, and GAB-107. The "Restrict" button is highlighted, indicating that GAB-104 and GAB-105 are selected for restriction. A "None" button is also present. At the bottom, a note states: "Select the rooms which will not be available for this School Year and Semester. The selected room are those which may not be used (restricted, for some reasons) for this semester."

**Figure 40: Restrict Rooms, CASScheduler**

The scheduling proper only involves a mouse click. The authority to perform this function is only given to users having OCS privileges. Scheduling involves both the course and exam schedules for the semester.



**Figure 41: Schedule, CASScheduler**

User having OCS Personnel privileges will be shown the fittest solutions that were produced by the system. If full solutions were found, the at most ten of the best and unique will be shown (see Figure 42), otherwise, only the best among the not fit solutions will be made known where the constraint violated is indicated at beside each scheduled class (refer to Figure 43).

SOLUTION 1			
	room	timeslot	instructor
Fitness	0.000000	0.000000	1.250000
Constraints:	-0.000000	-0.000000	-0.000000
MH 07:00:00-08:30:00			
NATSCIS LEC	RH - 300		JD AGAPITO,
GEO11 LEC	RH - 11C		Aman,
MATH73 LEC	GAB - 303		Engle,
CHEM27 LEC	RH - 220		Billones,J.
PHYSICS72 LEC	RH - 220		Caldo,
PI100 LEC	GAB - 301B		CE JICASIANO,
COMMII LEC	RH - 11C		RL PARAS,
CMSC21 LEC	RH - 220		JJ DELOS REYES,
MATH73 LEC	GAB - 304		EDO VILLAFRANCA,
BIO10 LEC	RH - 300		MM PARUNGAO,
CMSC130 LEC	RH - 117		Carpio,
HUMII LEC	RH - 11A		GP ODAL-DEVORA,
COMMII LEC	RH - 11A		FC EVANGELISTA,
SOCSCI192 LEC	RH - 313		GEC MATEO,

**Figure 42: View All Solutions, CASScheduler**


No full solutions were found.

SOLUTION 1		
	room	timeslot
Fitness	20.000000	142.000000
Constraints:	-0.000000	-4.900000
M 07:00:00-09:00:00		
HUMII LEC	GAB - 202	
r HUMII LEC	RH - 223	
r CHEM27 LEC	RH - 223	
CMSC126 LEC	GAB - 102	
CMSC130 LEC	GAB - 204	
CMSC142 LEC	RH - 119	
CMSC21 LEC	GAB - 104	
GEO11 LEC	GAB - 308	
r GEO11 LEC	RH - 223	
MATH101 LEC	RH - 11B	
MATH121.1 LEC	RH - 221	
MATH73 LEC	RH - 220	
MATH73 LEC	RH - 300	
STS LEC	GAB - 307	
r HISTORYII LEC	GAB - 301B	
POLSCI196 LEC	RH - 313	
r SOCSCIII LEC	GAB - 301B	

**Figure 43: No full Solutions, CASScheduler**

All users may view the accepted or implemented schedule for any semester. It may be a room schedule, a faculty schedule, or schedules sorted by timeslot (shown in Figure 44-46).



University of the Philippines - Manila 

Welcome ocs, Today is Tuesday - March 20, 2007.

[Rooms](#) [Subjects](#) [Schedule](#) [View](#) [Miscellaneous](#) [Logout](#)

### View Schedule


\*School Year:   
 \*Semester:   
 Course Timetable  
 Final Exam Timetable

View

---

SCHEDULE	SUBJECT	INSTRUCTOR
MH 13:00:00-14:30:00	CMSC125	Carpio,
M 07:00:00-10:00:00	MATH75	Bastero,
M 07:00:00-10:00:00	CMSC142	Solano,

**Figure 44: View Room Schedule, CASScheduler**

University of the Philippines - Manila 

Welcome ocs, Today is Tuesday - March 20, 2007.

[Rooms](#) [Subjects](#) [Schedule](#) [View](#) [Miscellaneous](#) [Logout](#)

### View Schedule

\*School Year:   
 \*Semester:   
 Course Timetable  
 Final Exam Timetable

View

---

SCHEDULE	ROOM	SUBJECT
MH 08:30:00-10:00:00	RH-117	MATH121.1
TF 11:30:00-13:00:00	GAB-303	CMSC199(SP)

**Figure 45: View Faculty Schedule, CASScheduler**

Welcome ocs, Today is Tuesday - March 20, 2007.

[Rooms](#)   [Subjects](#)   [Schedule](#)   **[View](#)**   [Miscellaneous](#)   [Logout](#)

### View Schedule

\*School Year:   
 \*Semester:   
 Course Timetable  
 Final Exam Timetable

View:  |

---

SUBJECT	SCHEDULE	ROOM	INSTRUCTOR
COMMII	MH 07:00:00-08:30:00	RH-336	RL PARAS,
COMMII	MH 10:00:00-11:30:00	RH-25B	RL PARAS,
COMMII	MH 12:00:00-13:30:00	RH-300	RL PARAS,
COMMIIIENG	MH 07:00:00-08:30:00	GAB-304	GP ODAL-DEVORA,
COMMIIIENG	MH 08:30:00-10:00:00	RH-11A	GP ODAL-DEVORA,
HUMII	MH 11:00:00-12:30:00	RH-117	GP ODAL-DEVORA,

**Figure 46: View Subject Schedule, CASScheduler**

And to help the users of the system resolve conflicts (if in case they arise), they are also provided of an option to view room, subject and instructor conflicts as depicted in Figure 47.

Welcome ocs, Today is Tuesday - March 20, 2007.

[Rooms](#)   [Subjects](#)   [Schedule](#)   **[View](#)**   [Miscellaneous](#)   [Logout](#)

### View Schedule

\*School Year:   
 \*Semester:   
 Course Timetable  
 Final Exam Timetable

View:  |

---

SUBJECT	SCHEDULE	ROOM
HUMII-Lec	M 07:00:00-09:00:00	RH-223
CHEM27-Lec	M 07:00:00-09:00:00	RH-223
GEO11-Lec	M 07:00:00-09:00:00	RH-223
CMSC191(CT)-Lec	W 07:00:00-09:00:00	RH-229
MATH151-Lec	W 07:00:00-09:00:00	RH-229

**Figure 47: View Conflicts, CASScheduler**

Also, some supplementary functions are provided by the system. Department heads, these includes managing subjects, instructors, and courses as illustrated in Figure

48-50. OCS personnel may manage – add, edit/update, and delete – room and courses as shown in Figure 51-52, and may dissolve, or add petitioned subjects in Figure 53.

University of the Philippines - Manila

College of Arts and Sciences

CAS Scheduler

Welcome dpsmH, Today is Sunday - March 11, 2007.

[Curricula](#) [Subjects](#) [Instructors](#) [View](#) [Miscellaneous](#) [Logout](#)

[Manage Faculty](#) [Manage Courses](#) [Manage Subjects](#)

### Manage Faculty

Name: DelRosario, ▾

Faculty Code: 0095

Figure 48: Manage Faculty, *CASScheduler*

University of the Philippines - Manila

College of Arts and Sciences

CAS Scheduler

Welcome dpsmH, Today is Sunday - March 11, 2007.

[Curricula](#)
[Subjects](#)
[Instructors](#)
[View](#)
[Miscellaneous](#)
[Logout](#)

[Manage Faculty](#)
[Manage Courses](#)
[Manage Subjects](#)

### Manage Courses

Course Code

Course Description

Figure 49: Manage Courses, CAS Scheduler

University of the Philippines - Manila

College of Arts and Sciences

CAS Scheduler

Welcome dpsmH, Today is Sunday - March 11, 2007.

[Curricula](#)
[Subjects](#)
[Instructors](#)
[View](#)
[Miscellaneous](#)
[Logout](#)

[Manage Faculty](#)
[Manage Courses](#)
[Manage Subjects](#)

### Manage Subjects

Subject Code

Description

Subject Type  Lecture  Laboratory

Lecture Units  Hrs/Wk

Laboratory Type

Laboratory Units  Hrs/Wk

GE Subject?  Yes  No

Figure 50: Manage Subjects, CAS Scheduler

Welcome ocs, Today is Sunday - March 11, 2007.

[Rooms](#) [Subjects](#) [Schedule](#) [View](#) [Miscellaneous](#) [Logout](#)  
[Post Message](#) [Manage Rooms](#) [Manage Users](#)

### Manage Rooms

Building:    
 Room Name:    
 Room Type:  Lecture  Laboratory   
 Laboratory Type:    
 Capacity:    
 Is Shared?:  Yes  No   
 Department:

Figure 51: Manage Rooms , CASScheduler

Welcome ocs, Today is Sunday - March 11, 2007.

[Rooms](#) [Subjects](#) [Schedule](#) [View](#) [Miscellaneous](#) [Logout](#)  
[Post Message](#) [Manage Rooms](#) [Manage Users](#)

### Manage Users

Department:    
 Name:    
 Surname:  , First Name:  Middle Name:    
 username:  password:    
 usertype:

Figure 52: Manage Users, CASScheduler

Welcome ocs, Today is Tuesday - March 20, 2007.

[Rooms](#) [Subjects](#) [Schedule](#) [View](#) [Miscellaneous](#) [Logout](#)

**Add / Dissolve Subjects**

\*School Year

\*Semester

\*Subjects

\*Schedule

\*Room

[Back](#)

Figure 53: Add Petitioned Subject, *CASScheduler*

**CHAPTER VI**

**DISCUSSION**

*CASScheduler* is an application that produces optimal timetable the College of Arts and Science of the University of the Philippines, Manila. Users input scheduling parameters like rooms, instructors, and timeslots needed by the system to generate results. Optimal timetable/s is/are produced by applying Non-dominated Sorting Genetic Algorithm II (NSGA-II) to the gathered data. Several constraints are taken into consideration in finding an optimal schedule – it includes avoidance of conflict between rooms, instructors, and timeslot. Minimization functions which include satisfying the demand of students for each subject, avoidance of offering subjects taken by students of the same course and year level at the same timeslot, preventing the assignment of an instructor to a timeslot when he is unavailable, and the like, were also implemented.

This application was designed for use in a client-server scenario, and was developed using Apache 2 as HTTP server, MySQL 5 as DBMS, and PHP 5 as the server scripting language. It also uses client-side JavaScript for some form processing.

As compared to Schedule-EZ [8] and QUICK Scheduler [9], *CASScheduler* is more than just an automation of the manual process of timetabling. It made use of artificial intelligence techniques, like how UTTS and UTTSExam [40], and particularly CSS [29] and NSGA-II-UCTO approached the problem of timetabling. But unlike CSS and that of UCTO, *CASScheduler* made every effort to attend to both course and exam timetabling problem.

The results produced by *CASScheduler* is final which means that if changes in the inputs would like to be made, the whole scheduling process (not including the gathering

of data inputs) would have to be performed again, and it cannot guarantee that the results produced on the previous run would also be achieved in the next run of the algorithm on the data including the additional inputs or changes made in the inputs.

*CASScheduler* produces set of optimal solutions that may guide the OCS Personnel in making class and final exam schedules, the choice of the best solution is not anymore part of the system. Manual override by the user (OCS) is still in place. There may be times when no full solutions will be found. In this case, the OCS Personnel has the option to perform the scheduling process again, ensuring that all the assumptions of the system are met, or he may also try to increase the number of generation (iteration of the NSGA-II algorithm).

*CASScheduler* produces solutions in a relatively lesser time than manual timetabling provided that the specifications of the machine used is the best possible – specially the RAM, which must be no less than 512 MB. Otherwise, it could have a very poor running time.



## CHAPTER VII

### CONCLUSION

*CASScheduler* can produce optimal class and exam timetables for the College of Arts and Science of the University of the Philippines, Manila.

It allows users to input scheduling parameters such as rooms, courses (or degree program), and instructors (or lecturers) by providing a form within the users' browser so that the users can submit the necessary information regarding the parameters. Using these information, the application executes the server-side scripts to produce timetables (either schedule of classes, or final exams schedule), which are optimal. Eligible user may choose from these timetables, the schedule to be implemented for the chosen semester, and this implemented schedule can then be viewed by other users of the system. Moreover, users are provided with different ways to view the implemented schedule, either by room, faculty, or per subject, and if in case conflict arise, room, subject and instructor conflicts may also be viewed separately to further help the user to design appropriate timetables.

## **CHAPTER VIII**

### **RECOMMENDATION**

At present the *CASScheduler* is able to produce timetables for classes, and final exams. It could be improved if timetabling of departmental exams will also be covered. Another interesting improvement would be to modify the engine used in such a way that the accepted results on its first run could be maintained if some changes in the scheduling parameters will be made (for instance, deletion or addition of new faculty, or subject).

It is also recommended to deploy the application on a server that is fast – at least 512 MB RAM – since as parameters increases, the running time of the application also increases, or modifying the mutation operator of the engine would also be helpful.

Additional constrains and fitness functions could also be made to improve the course (class) scheduling, and final exam scheduling, and make it more beneficial for the students; like implementing something that would consider irregular students

**CHAPTER IX**  
**BIBLIOGRAPHY**

- [1] Wren, A., “*Scheduling, Timetabling and Rostering – A Special relationship?*”, in The Practice and Theory of Automated Timetabling: Selected Papers from the 1<sup>st</sup> int’l Conf. on the practice and Theory of Automated Timetabling, Burke, E., Ross, P. (Eds.) Springer Lecture Notes in Computer Science Series, Vol. 1153, 1996, pp. 46-75.
- [2] Kazarlis, S., Petridis, V. and Fragkou, P., “*Solving University Timetabling Problems Using Advance Genetic Algorithms.*”
- [3] Landa Silva J.D., Burke E.K., Petrovic S., “*An Introduction to Multi-objective Metaheuristics for Scheduling and Timetabling*”, in: Metaheuristic for Multi-objective Optimisation, Gandibleux X., Sevaux M., Sorensen K., T’kindt V. (Eds.) Lecture Notes in Economics and Mathematical Systems, Vol. 535, Springer, pp. 91-129, 2004.
- [4] Burke,E.K., Jackson, K.S., Kingston, J.H. and Weare, R.F., “*Automated University Timetabling: The State of the Art*”, The Computer Journal, Vol. 40, No. 9, pp 565-571, 1997
- [5] Perzina, R., “*Solving University Timetabling Problems with Optimized Enrollment of Students by a Parallel Self-Adaptive Genetic Algorithm*”, Practice

and Theory of Automated Timetabling (PATAT), Burke, E., Ross, P. (Eds.) 2006, pp. 264–280. ISBN 80-210-3726-1.

- [6] Garey, M. R. and Johnson, D. S. “*Computers and Intractability: A Guide to the Theory of NP-Completeness*”, (1979) San Francisco, CA W.H. Freeman .
  
- [7] Even, S., Iati, A., Shamir, A. “*On the Complexity of Timetabling and Multicommodity Flow Problems*”. Siam Journal of Computation, vol. 5, no. 4, pp. 691-703.1976.
  
- [8] Siva, S., Chhabra, J., “*Schedule-EZ: A Tool for Scheduling Faculty, Rooms, and Courses*”. Use Services Conference Proceeding of the annual ACM SIGUCCS Conference on User services, 2003, pp. 21-24. ISBN: 1-58113-665-X. San Antonio, Texas.
  
- [9] Gregory, J.M., Carter, W.J., and Gregory, P.S. “*The Student's Handbook for Academic Survival in College*”, McGraw Hill, New York, 1997, p.p. 9-10.
  
- [10] Owen, C.K., “*QUICK Scheduler A Time-saving Tool for Scheduling Class Sections*”, Use Services Conference Proceeding of the 23<sup>rd</sup> annual ACM SIGUCCS Conference on User services, 2005, pp. 294-298. California
  
- [11] Thompson, J. and Dowsland, K.A., “*Variants of Simulated Annealing for the Examination Timetabling Problem*”. Annals of Operations Research, 1995. European Business Management School, University of Wales at Swansea, UK

- [12] Thompson, J. and Dowsland, K.A., “*General Cooling Schedules for a Simulated Annealing based Timetabling System*”. in the Practice and Theory of Automated Timetabling. Burke, E.K. and Ross, P. (Eds.) pp. 345-363, Springer-Verlag (Lecture Notes in Computer Science), 1996. European Business Management School, University of Wales at Swansea, UK
- [13] Boufflet, J.P. and N`egre, S. “*Three Methods used to solve an Examination Timetable Problem*” in The Practice and Theory of Automated Timetabling, ed. Burke, E.K. and Ross, P. pp. 327-344, Springer-Verlag (Lecture Notes in Computer Science), 1996. D`epartement de G`enie Informatique, Universit`e de Technologie de Compi`egne, France
- [14] Hertz, A., “*Tabu Search for Large-Scale Timetabling Problems*”. European Journal of Operations Research, no. 54, pp. 39-47, 1991. D`epartement d’Informatique et de Recherche Op`erationelle, Universit`e de Montr`eal, Canada
- [15] A Hertz, “Finding a Feasible Course Schedule using Tabu Search,” *Discrete Applied Mathematics*, vol. 35, no. 3, pp. 255-270, Elsevier Science Publishers, 1992. D`epartement d’Informatique et de Recherche Op`erationelle, Universit`e de Montr`eal, Canada
- [16] Corne, D., Ross, P., and Fang, H.L., “*Fast Practical Evolutionary Timetabling*” Lecture Notes in Computer Science, vol. 865 (Artificial Intelligence and Simulation of Behaviour (AISB) Workshop on Evolutionary Computing,

University of Leeds, UK, 11th-13th April 1994), pp. 251-263, Springer-Verlag, 1994. Department of Artificial Intelligence, University of Edinburgh, UK

- [17] Paechter\*, B., Cumming\*, A., Luchian†, H. and Petriuc‡, M. “*Two Solutions to the General Timetable Problem using Evolutionary Methods*” proceedings of the IEEE Conference on Evolutionary Computation 1994. \*Computer Studies Department, Napier University, Edinburgh, Scotland, UK; †Faculty of Computer Science, Al I Cuza University of Iasi, Romania; and ‡Technical University of Iasi, Romania
- [18] Burke, E.K., Elliman, D.G., and Weare, R.F. “*The Automation of the Timetabling Process in Higher Education*”. Journal of Educational Technology Systems, vol. 23, no. 4, pp. 257-266, Baywood Publishing Company, 1995. Department of Computer Science, University of Nottingham, UK
- [19] Burke, E.K., Elliman, D.G., and Weare, R.F, “*A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems*”. 6th International Conference on Genetic Algorithms (ICGA’95, Pittsburgh, USA, 15th-19th July 1995), Kaufmann, M.. San Francisco, CA, USA. Department of Computer Science, University of Nottingham, UK
- [20] EK Burke, DG Elliman, and RF Weare, “*Specialised Recombinative Operators for Timetabling Problems*”. proceedings of the AISB (Artificial Intelligence and Simulation of Behaviour) Workshop on Evolutionary Computing (University of

Sheffield, UK, 3rd-7<sup>th</sup> April 1995), pp. 75-85, Springer-Verlag, 1995. Department of Computer Science, University of Nottingham, UK

- [21] Fernandes, C., Paulo, C. J., Fernando, M., and Agostinho, R., “*High School Weekly Timetabling by Evolutionary Algorithms*”. Symposium on Applied Computing Proceedings of the 1999 ACM Symposium on computing, 1999, pp. 344-350. ISBN:1-58113-086-4. San Antonio, Texas.
  
- [22] Lim, A. Ang, J.C., Ho, W.K., Oon, W.C., “*UTTSExam: A Campus-Wide University Exam-Timetabling System,*” Eighteenth National Conference on Artificial Intelligence. 2002. p.p. 838-844. ISBN: 0-262-51129-0. Edmonton, Alberta, Canada.
  
- [23] Ho, W.K. and Lim, A., “*A Hybrid-Based Framework for Constraint Satisfaction Optimization Problems,*” in International Conference on Information Systems (ICIS) 2001, pg. 65-76.
  
- [24] Tsang, E., “*Foundations of Constraint Satisfaction*”, 1993.
  
- [25] Marin, H.T. , “*Combinations of GA and CSP Strategies for Solving the Examination Timetabling Problem*”. PhD thesis, Intituto Tecnológico y de Estudios Superiores de Menterrey, 1998.
  
- [26] Rayward-Smith, V.J. , Osman, I.H., Reeves, C.R. and Smith, G.D., *Modern Heuristics Search Methods*, 1996.

- [27] Burke, E. K. and Newall, J. P., "*A New Adaptive Heuristic Framework for Examination Timetabling Problems*". University of Nottingham, Working Group on Automated Timetabling, TR-2002-1 <http://www.cs.nott.ac.uk/TR/cgi/TR.cgi?tr=2002-1>
- [28] Yuri Bykov. "*Time-Predefined and Trajectory-Based Search: Single and Multiobjective Approaches to Exam Timetabling*", PhD thesis, University of Nottingham, UK, November 2003.
- [29] Gagno, P.A. , Sarmiento, L. and Teroso, S.K.. "*Multiobjective Course Scheduling*". Bachelor thesis, University of the Philippines – Diliman, March 2006.
- [30] Datta, D., Deb, K., & Fonseca, C.M., "*Multi-objective evolutionary algorithm for University Class Timetabling Problem*". in Evolutionary Scheduling, Springer-Verlag (in Press), 2006.
- [31] Datta, D., Deb, K., Fonseca, C. M. "*Solving class timetabling problem of IIT Kanpur using multi-objective evolutionary algorithm*". KanGAL Report No. 2006006. June 2006.
- [32] de Werra D., "*An Introduction to Timetabling*". European Journal of Operational Research, Vol. 19, pp. 151-162, 1985.



- [33] Rankin R.C., “*Automated Timetabling in Practice*”, in: The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT 1995), Burke E.K., Ross P. (eds.), Lecture Notes in Computer Science, Vol. 1153, Springer, pp. 266-279, 1996.
- [34] Papadimitriou C.H., “*Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall”. 1982.
- [35] Reeves C.R. (ed.), “*Modern Heuristic Techniques for Combinatorial Problems*”. McGraw-Hill, 1995.
- [36] Steuer, R.E., “*Multiple Criteria Optimization: Theory, Computation and Application*”. Wiley. 1986.
- [37] Coello C.A., Van Veldhuizen D.A., Lamont G.B., “*Evolutionary Algorithms for Solving Multi-Objective Problems*”, Kluwer Academic Publishers, 2002.
- [38] Fonseca C.M., Fleming P.J., “*An Overview of Evolutionary Algorithms in Multi-objective Optimization*”. Evolutionary Computation, Vol. 3, No. 1, pp. 1-16, 1995.
- [39] Abramson, D., “*Constructing school timetables using simulated annealing: sequential and parallel algorithms*”. Management Science, 37(1), January 1991, pp. 98-113

- [40] A. Hertz, “*Tabu search for large scale timetabling problems*”. European journal of Operations Research, vol. 54, 1991, pp. 39-47.
- [41] Paechter, B., Cumming, A., Norman, M.G. and Luchian,H., “*Extensions to a memetic timetabling system*” in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling, Burke, E.K. and Ross, P.M. (eds.) 1995.
- [42] Schaerf, A., “*A Survey of Automated Timetabling*”. Artificial Intelligence Review, vol 13 (2), 1999, 87-127.
- [43] Tripathy, A., “*A lagrangian relaxation approach to course timetabling*”. Journal of the Operational Research Society, vol. 31,1980, pp. 599-603
- [44] Carter, M.W., “*A Survey of Practical Applications of Examination Timetabling Algorithms*”. Journal of Operations Research vol. 34, 1986, pp. 193-202.
- [45] Welsh D.J.A. and Powell M.B., “*An Upper Bound for the Chromatic Number of a Graph and Its Application to Timetabling Problems*”. Computer Journal vol. 10, 1967, pp. 85-86.
- [46] Broder S. “*Final Examination Scheduling*”. Comm. A.C.M. vol. 7. 1964. 494-498.

- [47] Matula D.W., Marble G. and Isaacson I.D. “*Graph Colouring Algorithms*” in Graph Theory and Computing, R.C.Read (ed.) Academic Press, New York.1972.
- [48] Burke, E.K., Elliman, D.G. and Weare, R.F., “*A University Timetabling System based on Graph Colouring and Constraint Manipulation*”. Journal of Research on Computing in Education Vol. 27, Iss. 1, pp. 1-18, 1994.
- [49] White, G.M. and Chan, P.W., “*Towards the Construction of Optimal Examination Timetables*”. INFOR vol. 17, 1979, p.p. 219-229.
- [50] Brailsford, S.C., Potts, C.N. and Smith, B.M., “*Constraint Satisfaction Problems: Algorithms and Applications*”. European Journal of Operational Research, vol 119, 1999, pp. 557-581. Burke, E.K. and Newall, J.P. (eds.) "A New Adaptive Heuristic Framework"
- [51] Burke, E.K., Petrovic, S. and Qu, R. “*Case Based Heuristic Selection for Timetabling Problems*” . Journal of Scheduling, vol 9, no. 2, pp 115-132, 2006
- [52] Petrovic, S., Yang, Y. and Dror, M., “*Case-based Selection of Initialisation Heuristics for Metaheuristic Examination Timetabling*”. Accepted for publication in Expert Systems With Applications, to appear in vol. 33 iss.3, 2007.
- [53] Burke, E.K., MacCarthy, B., Petrovic, S. and Qu, R., “*Multiple-Retrieval Case Based Reasoning for Course Timetabling Problems*”. Journal of the Operational Research Society, vol. 57, no. 2, pp 148-162, 2006.

- [54] Adamidis, P. and Arapakis, P., “*Evolutionary Algorithms in Lecture Timetabling*” Proceedings of the 1999 IEEE Congress on Evolutionary Computation (CEC '99), IEEE, 1999, pp. 1145-1151.
- [55] Colomi, A., Dorigo, M., and Maniezzo, V., “*Genetic algorithms – A new approach to the timetable problem*” In Lecture Notes in Computer Science - NATO ASI Series, Vol. F 82, Combinatorial Optimization, (Akgul et al eds), Springer-Verlag, 1990, pp. 235-239.
- [56] “*Genetic Algorithm*”, [http://en.wikipedia.org/wiki/Genetic\\_algorithm](http://en.wikipedia.org/wiki/Genetic_algorithm)
- [57] Voss S., Martello S., Osman I.H. and Rucaiol C. (eds.), “*Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*”. Kluwer Academic Publishers, 1999.
- [58] Deb K., “*Multi-Objective Optimization Using Evolutionary Algorithms*”, Wiley, 2001.
- [59] “*Evolutionary Algorithm*”, [http://en.wikipedia.org/wiki/Evolutionary\\_algorithm](http://en.wikipedia.org/wiki/Evolutionary_algorithm)
- [60] Bagchi T.P., “*Multi-objective Scheduling By Genetic Algorithms*”, Kluwer Academic Publishers, 1999.

- [61] Bagchi T.P., “*Pareto-Optimal Solutions for Multi-objective Production Scheduling Problems*”, In: Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001), Lecture Notes in Computer Science, Vol. 1993, Springer, pp. 458-471, 2001.
- [62] Brizuela C., Sannomiya N., Zhao Y., “*Multi-objective Flow-Shop: Preliminary Results*”, In: [61], pp. 443-457, 2001.
- [63] Ishibuchi, H., Yoshida, T., and Murata, T., “*Selection of Initial Solutions for Local Search in Multi-objective Genetic Local Search*”. Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), IEEE Press, pp. 950-955, 2002.
- [64] Murata, T., Ishibuchi, H., Gen, M., “*Specification of Genetic Search Directions in Cellular Multi-objective Genetic Algorithms*”. In: Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001), Lecture Notes in Computer Science, Vol. 1993, Springer, pp. 82-95, 2001.
- [65] Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T. “*A fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II*”. IEEE Transacion in Evolutionary Computation, vol. 6 iss. 2, 181-197.
- [66] Tyson, J. “*How Internet Infrastructure Works*”  
<<http://computer.howstuffworks.com/internet-infrastructure5.htm>>.

[67] “*LAMP Servers on SUSE Linux Enterprise*”. <<http://www.novell.com/products/serve/lamp.html>>.

[68] Synaptic Package Manager. Computer Software. Disk.

# CHAPTER X

## APPENDIX

### nsga2.c (class scheduling)

```
/* This is a Multi-Objective GA program.
*****
*****
This program is the implementation of the
NSGA-2 proposed by
*
* Prof. Kalyanmoy Deb and his students .
*
*
* copyright Kalyanmoy Deb
*****
*****
18.08.2003: The keepaliven.h file is
modified to have normalized
crowding distance calculation. The previous
version of
the code did not have this feature. This way
, maintaining
a good distribution of solutions in problems
having quite
a different range of objective functions
were difficult.
Hopefully, with this modification, such
difficulties will
not appear. -- K. Deb
18.08.2003: Also the dfit.h file is deleted.
It was not needed any way.
The user have to give the input manually or
through a data file.
The user needs to enter objective functions
in func-con.h
The code can also take care of the
constraints. Enter the constraints
in the space provided in the func-con.h file
.
Constraints must be of the following type:
g(x) >= 0.0
Also normalize all constraints (see the
example problem in func-con.h)
If your program asks you to increase the
values of some parameters in the
program come to main program and accordingly
changed the values which are
defined against #define ...
The program generates few output files.
These are described as
1.output.out
* This file has the detailed
record for all the variables,
* the fitness values, constraint
values, overall constraint
violation (penalty) and their ranks for all
the members
* of old population in the left
hand side of the |**|
* and of new population in the
right hand side.
2.all_fitness.out
* This file prints the record of all
the fitness values for
* different individual of new
population created at all
generations.
3.g_rank_record.out
* This file maintains the record of
individuals in global pop-
ulation at different ranks for all
the generations.
4.ranks.out
* This file prints the number of
individual at different ranks
*
* in old and new population and
finds rank ratios
5.final_fitness.out
* This file has the fitness
value of all feasible and
non-dominated individuals at the final
generation
6.final_var.out
* This file has the all the
variables of the feasible
and non-dominated individuals at the final
generation.
The i-th solutions here corresponds to the i
-th solution
in the final_fitness.out file.
7.plot.out This file contains gnuplot
-based file for plotting
the non-dominated feasible solutions
obtained by the code.
*****
*****
* This is recommended to delete or rename
all the *.out files
* obtained from the previous runs as some
files are opened in
* append mode so they give false resemblance
of data if the
* user is not careful
* Compilation procedure: gcc nsga2.c -lm
* Run ./a.out with or without an input file
* Input data files: Three files are included
, but at one time one is needed
* depending on the type of variables used:
* inp-r (template file input-real) : All
variables are real-coded
* inp-b (template file input-binary): All
variables are binary-coded
* inp-rb(template file input-rl+bin): Some
variables are real
* and some are binary
*/

#include <math.h>
#include <mysql/mysql.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>

#define square(x) ((x)*(x))

#define maxpop 500 /*Max population */
#define maxchrom 200 /*Max chromosome
length*/
#define maxvar 500 /*Max no. of variables
*/
#define maxfun 10 /*Max no. of functions
*/
#define maxcons 20 /*Max no. of
Constraints*/

/* USER-DEFINED VARIABLES */
#define MAX_YEAR_LEVEL 5
#define LECTURE 1
#define LABORATORY 2
#define LEC_LAB 3

#define ROOM_CONFLICT 0.5
#define FACULTY_CONFLICT 0.5
#define SLOT_TYPE_CONFLICT 0.6
```

```

#define INCOMPATIBLE_ROOM 0.8
#define CURRICULUM_CONFLICT 0.6
#define FACULTY_SUBJECT_CONFLICT 6
#define ROOM_NOT_SHARED_CONFLICT 3
#define OVERLOAD_INSTRUCTOR 1.5
#define SCHED_PREF_CONFLICT 1.25
#define UNSATISFIED_DEMAND 0.5

/* USER-DEFINED VARIABLES START */
typedef struct{
    int subjID;
    char *subjCode;
    int dept;
    int dept_i;
    int lecunits;
    int labunits;
    int subjType; // 1 - lec, 2- lab, 3-lec&
                 lab
    int subjlabType;
    int GEsubj;
    int nTSec;
    int demand;
    int* fac;
    int numOfFitInsts;
    int* fitInsts;
}subject;

typedef struct{
    int subjID;
    int crseID;
    int yearLevel; // for year standing
                  offered, 0 for elective
    int dept;
    int dept_i;
    int subjType; // 1 - lec, 2- lab, 3-lec&
                 lab
    int subjlabType;
    int units;
    int nSec;
    int nSlots;
    int nMeet;
    float hours;
    int *slotFit;
    int *indexFitSlots;
    int numFitSlots;

    int sub_ind;
    int cor_s_ind;
} sem_subject;

typedef struct{
    int courseID;
    char *courseCode;
    int dept;
    int dept_i;
} course;
typedef struct{
    int subject_i;
    int semsubject_i;
    int course_i;
    int yearLevel;
    int section_i;
    int type;
    int labtype;
} section;
typedef struct{
    int roomID;
    char *bldg;
    char *roomName;
    int type; /* 1 - lec, 2- lab, 3-lec/lab
             */
    int labtype;
    int capacity;

    int isShared;
    int dept;
    int dept_i;
} room;
typedef struct{
    int id;
    char* code;
    int start_time;
    int end_time;
    int day;
    float hrs;
    int num_meetings;
} slot;
typedef struct{
    int facID;
    char* facName;
    int dept;
    int dept_i;
    int* unavTime;
    int load;
} faculty;

typedef struct{
    int faculty_i;
    int subject_i;
    int type;
} subjectFaculty;

typedef struct{
    int id;
    int num_meetings;
    float lec_hrs;
    float lab_hrs;
} slot_type;

typedef struct{
    int deptID;
    char* deptName;
    int numLecRooms;
    int numLabRooms;
    int* lecRooms;
    int* labRooms;
} department;

int slot_n;
int faculty_n;
int room_n;
int subject_n;
int semsubject_n;
int course_n;
int section_n;
int department_n;
int subjfac_n;
int CONST_ADMIN_DEPT_ID = 0;

subject* subject_list;
sem_subject* semsubject_list;
course* course_list;
section* section_list;
room* room_list;
faculty* faculty_list;
slot* timeslots;
department* department_list;
subjectFaculty* subjfac_list;
int **overlaps;
int **consecutive;
int **prev_template;
int *indexTable;
int *numSectionTable;
int clock_start = 0;
int clock_end = 0;
int sfinal, rfinal, ifinal;
time_t tstart, tend;

```



```

FILE *eval_ptr;
FILE *test_ptr;
FILE *compare;
int t;
char* AY_SEM;
/* USER-DEFINED VARIABLES END*/

int gener, /*No of generations*/
nvar,nchrom, /*No of variables*/
ncons, /*No of Constraints*/
vlen[maxvar], /*Array to store no of bits
for each variable*/
nmut, /* No of Mutations */
ncross, /*No of crossovers*/
ans;
float seed, /*Random Seed*/
pcross, /*Cross-over Probability*/
pmut_b, pmut_r, /*Mutation Probability*/
lim_b[maxvar][2],
lim_r[maxvar][2]; /*Limits of variable in
array*/
float di, /*Distribution Index for
the Cross-over*/
dim, /*Distribution Index for
the Mutation*/
delta_fit, /* variables required
for fitness for fitness sharing */
min_fit,
front_ratio;
int optype, /*Cross-over type*/
nfunc, /*No of functions*/
sharespace; /*Sharing space (either
parameter or fitness)*/
double coef[maxvar]; /*Variable used for
decoding*/
static int popsize, /*Population Size*/
chrom; /*Chromosome size*/

typedef struct /*individual properties
*/
{
int genes[maxchrom], /*bianry chromosome*/
rank, /*Rank of the
individual*/
flag; /*Flag for ranking*/
float xreal[maxvar], /*list of real
variables*/
xbin[maxvar]; /*list of decoded
value of the chromosome */
float fitness[maxfun], /*Fitness values */
constr[maxcons], /*Constraints
values*/
cub_len, /*crowding distance
of the individual*/
error; /* overall constraint
violation for the individual*/

int* sched;
}individual; /*Structure defining
individual*/

typedef struct
{
int maxrank; /*Maximum rank
present in the population*/
float rankrat[maxpop]; /*Rank Ratio*/
int rankno[maxpop]; /*Individual at
different ranks*/
individual ind[maxpop], /*Different
Individuals*/
*ind_ptr;
}population ; /*Population

Structure*/
#include "random.h" /*Random Number
Generator*/
#include "input.h" /*File Takes Input
from user*/
#include "realinit.h" /*Random
Initialization of the populaiton*/
#include "init.h" /*Random
Initialization of the population*/
#include "decode.h" /*File decoding
the binary dtrings*/
#include "ranking.h" /*File Creating
the Pareto Fronts*/
#include "rancon.h" /*File Creating
the Pareto Fronts when
Constraints are
specified*/
#include "func-con.h" /*File Having the
Function*/
#include "select.h" /*File for
Tournament Selection*/
#include "crossover.h" /*Binary Cross-
over*/
#include "uniformxr.h" /*Uniform Cross-
over*/
#include "realcross2.h" /*Real Cross-over
*/
#include "mut.h" /*Binary Mutation
*/
#include "realmut1.h" /*Real Mutation*/
#include "keepaliven.h" /*File For Elitism
and Sharing Scheme*/
#include "report.h" /*Printing the
report*/

/* USER-DEFINED HEADER FILES */
#include "parameters.h" /* for parameter
transformations */
#include "misc.h" /* miscellaneous
functions */
#include "mutation.h" /*for room, faculty and
slot mutation */
#include "cassCrossover.h" /* to do real
crossover */
#include "timetableOut.h" /* for output
*/
/* USER-DEFINED HEADER FILES END*/

population oldpop,
newpop,
matepop,
*old_pop_ptr,
*new_pop_ptr,
*mate_pop_ptr;
/*Defining the population Structures*/
main(int argc, char *argv[])
{
AY_SEM = argv[1];

/*Some Local variables to this Problem (
Counters And some other pointers*/
int ind1, ind2;
int i,j,l,f,maxrank1;
float *ptr,tot;
FILE
*rep_ptr,
*gen_ptr,
*rep2_ptr,
*end_ptr,
*g_var,
*lastit;

```

```

/*File Pointers*/
test_ptr = fopen("test.txt", "w");

old_pop_ptr = &(oldpop);
nmult = 0;
ncross = 0;

/*Give the initial seed*/
printf("\nGive random seed(between 0 and 1)
\n");

seed = 0.125;
printf("random seed: %f", seed);
warmup_random(seed);

printf("\nRetrieving input from user...\n");
printf("Calling makeVar... ");
transformParameters();
printf("done.\nDetermining overlaps in
timeslots...");
fflush(stdout);
overlaps = findOverlappingSlots();
printf("done.\nDetermining consecutive
timeslots...");
fflush(stdout);
consecutive = findConsecSlots();
printf("done.\n");
fflush(stdout);

/*Get the input from the file input.h*/
input(rep_ptr);
printf("\nFinished retrieving input from
user.\n\n");
fflush(stdout);

time(&tstart);

printf("Initializing %d variables... ", nvar
);
/*Binary Initializaton*/
if (nchrom > 0)
init(old_pop_ptr);
if (nvar > 0)
realignit(old_pop_ptr);
printf("done.\n");

old_pop_ptr = &(oldpop);

// decode binary strings
decode(old_pop_ptr);

old_pop_ptr = &(oldpop);
new_pop_ptr = &(newpop);

for(j = 0; j < popsize; j++)
{
/*Initializing the Rank array having
different individuals
at a particular rank to zero*/
old_pop_ptr->rankno[j] = 0;
new_pop_ptr->rankno[j] = 0;
}

printf("Evaluating initial population...");
func(old_pop_ptr);
printf(" done.\n");
/*Function Calculation*/

/*****
*****/
/*-----GENERATION STARTS
HERE-----*/
for (i = 0; i < gener; i++)
{
printf("\nGeneration = %d\n", i+1);
old_pop_ptr = &(oldpop);
mate_pop_ptr = &(matepop);

printf("Executing selection phase... ");
fflush(stdout);
/*-----SELECT-----*/
nselect(old_pop_ptr, mate_pop_ptr);
new_pop_ptr = &(newpop);
mate_pop_ptr = &(matepop);
printf(" done.\n");
printf("Executing crossover phase...");
/*CROSSOVER-----*/
if (nchrom > 0)
{
if (optype == 1)
{
crossover(new_pop_ptr, mate_pop_ptr);
/*Binary Cross-over*/
}
if (optype == 2)
{
unicross(new_pop_ptr, mate_pop_ptr);
/*Binary Uniform Cross-over*/
}
}
if (nvar > 0)
//realcross(new_pop_ptr, mate_pop_ptr);
ishcross(new_pop_ptr, mate_pop_ptr);
/*Real Cross-over*/
printf(" done.\n");
printf("Executing mutation phase...");
fflush(stdout);
/*-----MUTATION-----*/
new_pop_ptr = &(newpop);
if (nchrom > 0)
mutate(new_pop_ptr);
/*Binary Mutation */
if (nvar > 0) {
mutate_intel(new_pop_ptr);
}
/*Real Mutation*/

new_pop_ptr = &(newpop);
printf(" done.\n");
fflush(stdout);
printf("Executing decoding phase...");
/*-----DECODING-----*/

if (nchrom > 0)
decode(new_pop_ptr);
printf(" done.\n");
fflush(stdout);
/*Decoding for binary strings*/
new_pop_ptr = &(newpop);

printf("Executing function evaluation...");
/*-----FUNCTION EVALUATION-----
*/
new_pop_ptr = &(newpop);
func(new_pop_ptr);
printf(" done.\n");
fflush(stdout);

printf("Executing selection keep fronts
alive... ");
/*-----SELECTION KEEPING
FRONTS ALIVE-----*/
old_pop_ptr = &(oldpop);

```

```

new_pop_ptr = &(newpop);
mate_pop_ptr = &(matepop);
printf(" done.\n");
/*Elitism And Sharing Implemented*/
keepalive(old_pop_ptr ,new_pop_ptr ,
          mate_pop_ptr,i+1);

mate_pop_ptr = &(matepop);

printf("Executing report printing...");
mate_pop_ptr = &(matepop);
/*-----REPORT PRINTING-----*/
//report(i ,old_pop_ptr ,mate_pop_ptr ,
        rep_ptr ,gen_ptr , lastit );
printf(" done.\n");
/*-----*/
printf("Executing rank ratio calculation
phase...");
/*-----Rank Ratio Calculation-----*/
new_pop_ptr = &(matepop);
old_pop_ptr = &(oldpop);
/*Finding the greater maxrank among the two
populations*/
if(old_pop_ptr->maxrank > new_pop_ptr->
maxrank)
    maxrank1 = old_pop_ptr->maxrank;
else
    maxrank1 = new_pop_ptr->maxrank;
/*fprintf(rep2_ptr,"-----RANK AT
GENERATION %d-----\n",i+1);
fprintf(rep2_ptr,"Rank old ranks   new ranks
rankratio\n");
*/
for(j = 0;j < maxrank1 ; j++)
{
    /*Sum of the no of individuals at any rank
in old population
and the new population*/
    tot = (old_pop_ptr->rankno[j]) + (
        new_pop_ptr->rankno[j]);
    /*Finding the rank ratio for new
population at this rank*/
    new_pop_ptr->rankrat[j] = (new_pop_ptr->
rankno[j])/tot;
    /*Printing this rank ratio to a file
called ranks.dat*/
}
//fprintf(rep2_ptr,"-----Rank
Ratio-----\n");
/*-----*/
printf(" done.\n");
printf("Executing copy of new pop to old pop
...");
/*-----Copying the new population to old
population-----*/
old_pop_ptr = &(oldpop);
new_pop_ptr = &(matepop);
for(j = 0;j < popsize;j++)
{
    old_pop_ptr->ind_ptr = &(old_pop_ptr->ind[
j]);
    new_pop_ptr->ind_ptr = &(new_pop_ptr->ind[
j]);
    if(nchrom > 0)
    {
        /*For Binary GA copying of the
chromosome*/
        for(l = 0;l < chrom;l++)
            old_pop_ptr->ind_ptr->genes[l]=
                new_pop_ptr->ind_ptr->genes[l];
        for(l = 0;l < nchrom;l++)
            old_pop_ptr->ind_ptr->xbin[l] =
                new_pop_ptr->ind_ptr->xbin[l];
    }
    if(nvar > 0)
    {
        /*For Real Coded GA copying of the
chromosomes*/
        for(l = 0;l < nvar;l++)
            old_pop_ptr->ind_ptr->xreal[l] =
                new_pop_ptr->ind_ptr->xreal[l];
    }
    /*Copying the fitness vector */
    for(l = 0 ; l < nfunc ;l++)
        old_pop_ptr->ind_ptr->fitness[l] =
            new_pop_ptr->ind_ptr->fitness[l];
    /*Copying the dummy fitness*/
    old_pop_ptr->ind_ptr->cub_len =
        new_pop_ptr->ind_ptr->cub_len;
    /*Copying the rank of the individuals*/
    old_pop_ptr->ind_ptr->rank = new_pop_ptr->
ind_ptr->rank;
    /*Copying the error and constraints of the
individual*/
    old_pop_ptr->ind_ptr->error = new_pop_ptr-
>ind_ptr->error;
    for(l = 0;l < ncons;l++)
    {
        old_pop_ptr->ind_ptr->constr[l] =
            new_pop_ptr->ind_ptr->constr[l];
    }
    /*Copying the flag of the individuals*/
    old_pop_ptr->ind_ptr->flag = new_pop_ptr->
ind_ptr->flag;
} // end of j
maxrank1 = new_pop_ptr->maxrank ;
/*Copying the array having the record of the
individual
at different ranks */
for(l = 0;l < popsize;l++)
{
    old_pop_ptr->rankno[l] = new_pop_ptr->
rankno[l];
}
/*Copying the maxrank */
old_pop_ptr->maxrank = new_pop_ptr->maxrank;
/*Printing the fitness record for last
generation in a file last*/
if(i == gener-1)
{
    /* for the last generation
old_pop_ptr = &(matepop);
output timetables(old_pop_ptr);
// end of f (printing)
} // for the last generation
printf("...done.\n");
} // end of i

/*
Generation Loop Ends
*/
/*****
*****
printf("NSGA ENGINE TERMINATED.\n");
fclose(test_ptr);
}

```

## nsga2.c (final exam scheduling)

```
/* This is a Multi-Objective GA program.
*****
* This program is the implementation of the
  NSGA-2 proposed by
* Prof. Kalyanmoy Deb and his students .
*
* copyright Kalyanmoy Deb
*****
18.08.2003: The kepaliven.h file is
  modified to have normalized
  crowding distance calculation. The
  previous version of
  the code did not have this feature. This
  way, maintaining
  a good distribution of solutions in
  problems having quite
  a different range of objective functions
  were difficult.
  Hopefully, with this modification, such
  difficulties will
  not appear. -- K. Deb
18.08.2003: Also the dfit.h file is deleted.
  It was not needed any way.
The user have to give the input manually or
through a data file.
The user needs to enter objective functions
in func-con.h
The code can also take care of the
constraints. Enter the constraints
in the space provided in the func-con.h file
.
Constraints must be of the following type:
g(x) >= 0.0
Also normalize all constraints (see the
example problem in func-con.h)
If your program asks you to increase the
values of some parameters in the
program come to main program and accordingly
changed the values which are
defined against #define ...
The program generates few output files.
These are described as
1.output.out
* This file has the detailed record for all
  the variables,
* the fitness values, constraint values,
  overall constraint
  violation (penalty) and their ranks for
  all the members
* of old population in the left hand side
  of the |**|
* and of new population in the right hand
  side.
2.all_fitness.out
* This file prints the record of all the
  fitness values for
* different individual of new population
  created at all
* generations.
3.g_rank_record.out
* This file maintains the record of
  individuals in global pop-
* ulation at different ranks for all the
  generations.
4.ranks.out
* This file prints the number of individual
  at different ranks
* in old and new population and finds rank
  ratios
```

```
5.final_fitness.out
* This file has the fitness value of all
  feasible and
  non-dominated individuals at the
  final generation
6.final_var.out
* This file has the all the variables of the
  feasible
  and non-dominated individuals at the
  final generation.
  The i-th solutions here corresponds
  to the i-th solution
  in the final_fitness.out file.
7.plot.out          This file contains gnuplot
  -based file for plotting
  the non-dominated feasible solutions
  obtained by the code.
*****
* This is recommended to delete or rename
  all the *.out files
* obtained from the previous runs as some
  files are opened in
* append mode so they give false
  resemblance of data if the
* user is not careful
Compilation procedure: gcc nsga2.c -lm
Run ./a.out with or without an input file

Input data files: Three files are included,
  but at one time one is needed
  depending on the type of variables used:
inp-r (template file input-real) : All
  variables are real-coded
inp-b (template file input-binary): All
  variables are binary-coded
inp-rb(template file input-rl+bin): Some
  variables are real and some are binary
*/

#include <mysql/my_global.h>
#include <mysql/mysql.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include <stddef.h>
#include <time.h>
#include <sys/types.h>

#define square(x) ((x)*(x))
#define maxpop 500 /*Max population */
#define maxchrom 200 /*Max chromosome
  length*/
#define maxvar 500 /*Max no. of variables
  */
#define maxfun 10 /*Max no. of functions
  */
#define maxcons 20 /*Max no. of
  Constraints*/

/* USER-DEFINED VARIABLES */
#define MAX_YEAR_LEVEL 5
#define LECTURE 1
#define LABORATORY 2
#define LEC_LAB 3

#define ROOM_CONFLICT 0.5
#define FACULTY_CONFLICT 0.5
#define PENALTY_SUBJECT_CONFLICT 0.7
#define SLOT_TYPE_CONFLICT 0.7
```

```

#define INCOMPATIBLE_ROOM 0.8
#define CURRICULUM_CONFLICT 0.6
#define ROOM_NOT_SHARED_CONFLICT 3
#define UNSATISFIED_DEMAND 0.7
/* USER-DEFINED VARIABLES START */

typedef struct{
    int subjID;
    char *subjCode;
    int dept;
    int dept_i;
    int subjType; // 1 - lec, 2- lab, 3-lec&
                lab
    int subjlabType;
    int nTSec;
    int demand;
    int* fac;
    int numOFFitInsts;
    int* fitInsts;
} subject;

typedef struct{
    int subjID;
} subjects;

typedef struct{
    int subjID;
    int crseID;
    int yearLevel; // for year standing
                  offered, 0 for elective
    int dept;
    int dept_i;
    int subjType; // 1 - lec, 2- lab, 3-lec&
                lab
    int subjlabType;
    int nSec;
    int nSlots;
    float examhours;
    int *slotFit;
    int *indexFitSlots;
    int numFitSlots;
    int sub_ind;
    int subject_i;
    int cors_ind;
} sem_subject;

typedef struct{
    int courseID;
    char *courseCode;
    int dept;
    int dept_i;
} course;

typedef struct{
    int subject_id;
    int subject_i;
    int subject_ii;
    int semsubject_i;
    int course_i;
    int yearLevel;
    int section_i;
    int type;
    int labtype;
} section;

typedef struct{
    int roomID;
    char *bldg;
    char *roomName;
    int type; /* 1 - lec, 2- lab, 3-lec/lab
            */
    int labtype;
    int capacity;
    int isShared;
    int dept;
    int dept_i;
} room;

typedef struct{
    int id;
    char* code;
    int start_time;
    int end_time;
    int day;
    float hrs;
    int num_meetings;
} slot;

typedef struct{
    int facID;
    char* facName;
    int dept;
    int dept_i;
    int* unavTime;
    int load;
} faculty;

typedef struct{
    int faculty_i;
    int subject_i;
    int type;
} subjectFaculty;

typedef struct{
    int id;
    int num_meetings;
    float lec_hrs;
    float lab_hrs;
} slot_type;

typedef struct{
    int deptID;
    char* deptName;
    int numLecRooms;
    int numLabRooms;
    int* lecRooms;
    int* labRooms;
} department;

int slot_n;
int faculty_n;
int room_n;
int subject_n;
int subject_nn;
int semsubject_n;
int course_n;
int section_n;
int department_n;
int subjfac_n;
int CONST_ADMIN_DEPT_ID = 0;

subject* subject_list;
subjects* subjects_list;
sem_subject* semsubject_list;
course* course_list;
section* section_list;
room* room_list;
faculty* faculty_list;
slot* timeslots;
department* department_list;
subjectFaculty* subjfac_list;

int **overlaps;

```

```

int **consecutive;
int **prev_template;
int *indexTable;
int *numSectionTable;

int clock_start = 0;
int clock_end = 0;
int sfinal, rfinal, ifinal;
time_t tstart, tend;
FILE *eval_ptr;
FILE *test_ptr;
FILE *compare;

int t;
char* AY_SEM;
/* USER-DEFINED VARIABLES END*/

int gener, /*No of generations*/
nvar,nchrom, /*No of variables*/
ncons, /*No of Constraints*/
vlen[maxvar], /*Array to store no of bits
for each variable*/
nmut, /* No of Mutations */
ncross, /*No of crossovers*/
ans;
float seed, /*Random Seed*/
pcross, /*Cross-over Probability*/
pmut_b, pmut_r, /*Mutation
Probability*/
lim_b[maxvar][2],
lim_r[maxvar][2]; /*Limits of variable in
array*/
float di, /*Distribution Index for
the Cross-over*/
dim, /*Distribution Index for
the Mutation*/
delta_fit, /* variables required
for fitness for fitness sharing */
min_fit,
front_ratio;
int optype, /*Cross-over type*/
nfunc, /*No of functions*/
sharespace; /*Sharing space (either
parameter or fitness)*/

double coef[maxvar]; /*Variable used for
decoding*/

static int popsize, /*Population Size*/
chrom; /*Chromosome size*/

typedef struct /*individual properties
*/
{
int genes[maxchrom], /*bianry chromosome*/
rank, /*Rank of the
individual*/
flag; /*Flag for ranking*/
float xreal[maxvar], /*list of real
variables*/
xbin[maxvar]; /*list of decoded
value of the chromosome */

float fitness[maxfun], /*Fitness values */
constr[maxcons], /*Constraints
values*/
cub_len, /*crowding distance
of the individual*/
error; /* overall
constraint violation for the
individual*/

int* sched;

}individual; /*Structure defining
individual*/

typedef struct
{
int maxrank; /*Maximum rank
present in the population*/
float rankrat[maxpop]; /*Rank Ratio*/
int rankno[maxpop]; /*Individual at
different ranks*/
individual ind[maxpop], /*Different
Individuals*/
*ind_ptr;
}population; /*Popouation
Structure*/

#include "random.h" /*Random Number
Generator*/
#include "input.h" /*File Takes Input
from user*/
#include "realinit.h" /*Random
Initialization of the populaiton*/
#include "init.h" /*Random
Initialization of the population*/
#include "decode.h" /*File decoding
the binary dtrings*/
#include "ranking.h" /*File Creating
the Pareto Fronts*/
#include "rancon.h" /*File Creating
the Pareto Fronts when
Constraints are
specified*/
#include "func-con.h" /*File Having the
Function*/
#include "select.h" /*File for
Tournament Selection*/
#include "crossover.h" /*Binary Cross-
over*/
#include "uniformxr.h" /*Uniform Cross-
over*/
#include "realcross2.h" /*Real Cross-over
*/
#include "mut.h" /*Binary Mutation
*/
#include "realmut1.h" /*Real Mutation*/
#include "keepaliven.h" /*File For Elitism
and Sharing Scheme*/
#include "report.h" /*Printing the
report*/

/* USER-DEFINED HEADER FILES */
#include "parameters.h" /* for parameter
transformations */
#include "misc.h" /* miscellaneous
functions */
#include "mutation.h" /*for room, faculty and
slot mutation */
#include "cassCrossover.h" /* to do real
crossover */
#include "timetableOut.h" /* for output
*/
/* USER-DEFINED HEADER FILES END*/

population oldpop,
newpop,
matepop,
*old_pop_ptr,
*new_pop_ptr,
*mate_pop_ptr;
/*Defining the population Structures*/

main(int argc, char *argv[])
{

```

```

AY_SEM = argv[1];
/* USER-DEFINED LOCAL VARS */
float tempProb = 0;
/* USER-DEFINED LOCAL VARS END*/
/*Some Local variables to this Problem (
    Counters And some other pointers*/
int ind1, ind2;
int i,j,l,f,maxrank1;
float *ptr,tot;
FILE
    *rep_ptr,
    *gen_ptr,
    *rep2_ptr,
    *end_ptr,
    *g_var,
    *lastit;
/*File Pointers*/
test_ptr = fopen("test.txt", "w");
old_pop_ptr = &(oldpop);
nmute = 0;
ncross = 0;
/*Give the initial seed*/
printf("\nGive random seed(between 0 and 1
)\n");
seed = 0.125;
printf("random seed: %f", seed);
warmup_random(seed);
printf("\nRetrieving input from user...\n"
);
printf("Calling makeVar... ");
transformParameters();
printf("done.\nDetermining overlaps in
timeslots...");
fflush(stdout);
overlaps =findOverlappingSlots();
printf("done.\nDetermining consecutive
timeslots...");
fflush(stdout);
consecutive = findConseSlots();
printf("done.\n");
fflush(stdout);
/*Get the input from the file input.h*/
input(rep_ptr);
printf("\nFinished retrieving input from
user.\n\n");
fflush(stdout);
time(&tstart);
printf("Initializing %d variables... ",
nvar);
/*Binary Initializaton*/
if (nchrom > 0)
    init(old_pop_ptr);
if (nvar > 0)
    realinit(old_pop_ptr);
//ish_init(old_pop_ptr);
printf("done.\n");
old_pop_ptr = &(oldpop);
// decode binary strings
decode(old_pop_ptr);
old_pop_ptr = &(oldpop);
new_pop_ptr = &(newpop);
for(j = 0;j < popsize;j++)
{
    /*Initializing the Rank array having
        different individuals
        at a particular rank to zero*/
    old_pop_ptr->rankno[j] = 0;
    new_pop_ptr->rankno[j] = 0;
}
printf("Evaluating initial population...")
;
func(old_pop_ptr);
printf(" done.\n");
/*Function Calculation*/
/*****
*****
*****GENERATION STARTS
HERE-----*/
for (i = 0;i < gener;i++)
{
    printf("\nGeneration = %d\n",i+1);
    old_pop_ptr = &(oldpop);
    mate_pop_ptr = &(matepop);
    printf("Executing selection phase... ");
    fflush(stdout);
    /*-----SELECT-----*/
    nselect(old_pop_ptr ,mate_pop_ptr );
    new_pop_ptr = &(newpop);
    mate_pop_ptr = &(matepop);
    printf(" done.\n");
    printf("Executing crossover phase...");
    /*CROSSOVER-----*/
    if (nchrom > 0)
    {
        if(optype == 1)
        {
            crossover(new_pop_ptr ,mate_pop_ptr
            );
            /*Binary Cross-over*/
        }
        if(optype == 2)
        {
            unicross(new_pop_ptr ,mate_pop_ptr )
            ;
            /*Binary Uniform Cross-over*/
        }
    }
    if (nvar > 0)
        cross(new_pop_ptr , mate_pop_ptr);
    /*Real Cross-over*/
    printf(" done.\n");
    printf("Executing mutation phase...");
    fflush(stdout);
    /*-----MUTATION-----*/
    new_pop_ptr = &(newpop);
    if (nchrom > 0)
        mutate(new_pop_ptr );
    /*Binary Mutation */
    if (nvar > 0){
        mutate_intel(new_pop_ptr);
    }
    /*Real Mutation*/
    new_pop_ptr = &(newpop);
    printf(" done.\n");
    fflush(stdout);
    printf("Executing decoding phase...");
    /*-----DECODING-----*/
    if(nchrom > 0)
        decode(new_pop_ptr );
    printf(" done.\n");
    fflush(stdout);
    /*Decoding for binary strings*/
    new_pop_ptr = &(newpop);
    printf("Executing function evaluation...
    ");
    /*-----FUNCTION EVALUATION-----
    ---*/
    new_pop_ptr = &(newpop);
    func(new_pop_ptr);
    printf(" done.\n");
    fflush(stdout);
    printf("Executing selection keep fronts
    alive... ");
    /*-----SELECTION KEEPING FRONTS
    ALIVE-----*/

```

```

old_pop_ptr = &(oldpop);
new_pop_ptr = &(newpop);
mate_pop_ptr = &(matepop);
printf(" done.\n");
/*Elitism And Sharing Implemented*/
kepalive(old_pop_ptr ,new_pop_ptr ,
        mate_pop_ptr,i+1);
mate_pop_ptr = &(matepop);
printf("Executing report printing...");
mate_pop_ptr = &(matepop);
/*-----REPORT PRINTING-----
-----*/
//report(i ,old_pop_ptr ,mate_pop_ptr ,
        rep_ptr ,gen_ptr, lastit );
printf(" done.\n");
/*-----*/
printf("Executing rank ration
        calculation phase...");
/*-----Rank Ratio Calculation---
-----*/
new_pop_ptr = &(matepop);
old_pop_ptr = &(oldpop);
/*Finding the greater maxrank among the
        two populations*/
if(old_pop_ptr->maxrank > new_pop_ptr->
        maxrank)
    maxrank1 = old_pop_ptr->maxrank;
else
    maxrank1 = new_pop_ptr->maxrank;
for(j = 0;j < maxrank1 ; j++)
{
    /*Sum of the no of individuals at any
        rank in old population
        and the new population*/
    tot = (old_pop_ptr->rankno[j])+ (
        new_pop_ptr->rankno[j]);
    /*Finding the rank ratio for new
        population at this rank*/
    new_pop_ptr->rankrat[j] = (new_pop_ptr
        ->rankno[j])/tot;
}
//fprintf(rep2_ptr,"-----
        Rank Ratio-----\n");
/*-----*/
printf(" done.\n");
printf("Executing copy of new pop to old
        pop...");
/*=====Copying the new population to
        old population=====*/
old_pop_ptr = &(oldpop);
new_pop_ptr = &(matepop);
for(j = 0;j < popsize;j++)
{
    old_pop_ptr->ind_ptr = &(old_pop_ptr->
        ind[j]);
    new_pop_ptr->ind_ptr = &(new_pop_ptr->
        ind[j]);
    if(nchrom > 0)
    {
        /*For Binary GA copying of the
            chromosome*/
        for(l = 0;l < chrom;l++)
            old_pop_ptr->ind_ptr->genes[l]=
                new_pop_ptr->ind_ptr->genes[l]
                ;

        for(l = 0;l < nchrom;l++)
            old_pop_ptr->ind_ptr->xbin[l] =
                new_pop_ptr->ind_ptr->xbin[l];
    }
}

if(nvar > 0)
{
    /*For Real Coded GA copying of the
        chromosomes*/
    for(l = 0;l < nvar;l++)
        old_pop_ptr->ind_ptr->xreal[l] =
            new_pop_ptr->ind_ptr->xreal[l]
            ;

    /*Copying the fitness vector */
    for(l = 0 ; l < nfunc ;l++)
        old_pop_ptr->ind_ptr->fitness[l] =
            new_pop_ptr->ind_ptr->fitness[l]
            ;

    /*Copying the dummy fitness*/
    old_pop_ptr->ind_ptr->cub_len =
        new_pop_ptr->ind_ptr->cub_len;
    /*Copying the rank of the individuals
        */
    old_pop_ptr->ind_ptr->rank =
        new_pop_ptr->ind_ptr->rank;
    /*Copying the error and constraints of
        the individual*/
    old_pop_ptr->ind_ptr->error =
        new_pop_ptr->ind_ptr->error;
    for(l = 0;l < ncons;l++)
    {
        old_pop_ptr->ind_ptr->constr[l] =
            new_pop_ptr->ind_ptr->constr[l];
    }
    /*Copying the flag of the individuals
        */
    old_pop_ptr->ind_ptr->flag =
        new_pop_ptr->ind_ptr->flag;
} // end of j
maxrank1 = new_pop_ptr->maxrank ;
/*Copying the array having the record of
        the individual
        at different ranks */
for(l = 0;l < popsize;l++)
{
    old_pop_ptr->rankno[l] = new_pop_ptr->
        rankno[l];
}
/*Copying the maxrank */
old_pop_ptr->maxrank = new_pop_ptr->
    maxrank;
/*Printing the fitness record for last
        generation in a file last*/
if(i == gener-1)
{
    // for the last generation
    old_pop_ptr = &(matepop);
    output_timetables(old_pop_ptr);
    // end of f (printing)
} // for the last generation
printf("...done.\n");
} // end of i

/*          Generation Loop Ends
*/
/*****
*****/
printf("NSGA ENGINE TERMINATED.\n");
fclose(test_ptr);
}

```



## random.h

```
#include <math.h>

/* variables are declared static so that
   they cannot conflict
   with names of */
/* other global variables in other files.
   See K&R, p 80, for
   scope of static */
static double oldrand[55]; /* Array of 55
   random numbers */
static int jrand; /* current random
   number */
static double rndx2; /* used with random
   normal deviate */
static int rndcalclflag; /* used with random
   normal deviate */

advance_random()
/* Create next batch of 55 random numbers */
{
    int j1;
    double new_random;

    for(j1 = 0; j1 < 24; j1++)
    {
        new_random = oldrand[j1] - oldrand[
            j1+31];
        if(new_random < 0.0) new_random =
            new_random + 1.0;
        oldrand[j1] = new_random;
    }
    for(j1 = 24; j1 < 55; j1++)
    {
        new_random = oldrand [j1] - oldrand
            [j1-24];
        if(new_random < 0.0) new_random =
            new_random + 1.0;
        oldrand[j1] = new_random;
    }
}

int flip(prob)
/* Flip a biased coin - true if heads */
float prob;
{
    float randomperc();

    if(randomperc() <= prob)
        return(1);
    else
        return(0);
}

initrandomnormaldeviate()
/* initialization routine for
   randomnormaldeviate */
{
    rndcalclflag = 1;
}

double noise(mu ,sigma)
/* normal noise with specified mean & std
   dev: mu & sigma */
double mu, sigma;
{
    double randomnormaldeviate();

    return((randomnormaldeviate()*sigma) +
        mu);
}

}

double randomnormaldeviate()
/* random normal deviate after ACM algorithm
   267 / Box-Muller Method */
{
    double sqrt(), log(), sin(), cos();
    float randomperc();
    double t, rndx1;

    if(rndcalclflag)
    {
        rndx1 = sqrt(- 2.0*log((double)
            randomperc()));
        t = 6.2831853072 * (double)
            randomperc();
        rndx2 = sin(t);
        rndcalclflag = 0;
        return(rndx1 * cos(t));
    }
    else
    {
        rndcalclflag = 1;
        return(rndx2);
    }
}

float randomperc()
/* Fetch a single random number between 0.0
   and 1.0 - Subtractive Method */
/* See Knuth, D. (1969), v. 2 for details */
/* name changed from random() to avoid
   library conflicts on some machines*/
{
    jrand++;
    if(jrand >= 55)
    {
        jrand = 1;
        advance_random();
    }
    return((float) oldrand[jrand]);
}

int rnd(low, high)
/* Pick a random integer between low and
   high */
int low,high;
{
    int i;
    float randomperc();

    if(low >= high)
        i = low;
    else
    {
        i = (randomperc() * (high - low + 1)
            ) + low;
        if(i > high) i = high;
    }
    return(i);
}

float rndreal(lo ,hi)
/* real random number between specified
   limits */
float lo, hi;
{
    return((randomperc() * (hi - lo)) + lo);
}

```

```

warmup_random(random_seed)
/* Get random off and running */
float random_seed;
{
    int j1, ii;
    double new_random, prev_random;

    oldrand[54] = random_seed;
    new_random = 0.000000001;
    prev_random = random_seed;
    for(j1 = 1 ; j1 <= 54; j1++)
    {
        ii = (21*j1)%54;
        oldrand[ii] = new_random;
        new_random = prev_random-new_random
        if(new_random<0.0) new_random =
            new_random + 1.0;
        prev_random = oldrand[ii];
    }

    advance_random();
    advance_random();
    advance_random();

    jrand = 0;
}

```

## mut.h

```

/* This is the module used to formulate the
mutation routine*/

void mutate(population *new_pop_ptr);

void mutate(population *new_pop_ptr)
{
    int i,*ptr,j;
    float randl;

    randl=randomperc();
    new_pop_ptr->ind_ptr = &(new_pop_ptr->ind[
0]);

    for(j = 0;j < popsize;j++)
    {
        ptr= &(new_pop_ptr->ind_ptr->genes[0])
        ;
        new_pop_ptr->ind_ptr =&(new_pop_ptr->
ind[j+1]);

        /*Select bit */
        for (i = 0;i < chrom;i++)
        {
            randl = randomperc();

            /*Check whether to do mutation or not*/
            if(randl <= pmut_b)
            {
                if(*ptr == 0)
                *ptr =1;
                else
                *ptr=0;
                nmut++;
            }
            ptr++;
        }
    }
    return;
}

```

## realinit.h

```

/*This is the file which initializes the
population*/
void realinit(population *pop_ptr);

void realinit(population *pop_ptr)
{
    int i,j,r,d2;
    float d,d1;

    for (i = 0 ; i < popsize ; i++)
    {
        for (j = 0; j < nvar; j++)
        {
            d = randomperc();
            d1 = 2*d - 1;
            /*if limits are not specified then
generates any number between
zero and infinity*/
            if(ans != 1)
            {
                pop_ptr->ind[i].xreal[j] = 1/d1 ;
            }

            /*if limits are specified it generates
the value in
range of minimum and maximum value of
the variable*/
            else
            {
                pop_ptr->ind[i].xreal[j] = d*(lim_r[j]
[1] - lim_r[j][0])+lim_r[j][0];
            }
        }
        /* pop_ptr->ind_ptr = &(pop_ptr->ind[i+1
]); */
    }
    /*pop_ptr->ind_ptr = &(pop_ptr->ind[0]);
*/

    return;
}

```

## init.h

```

/*This is the file which initializes the
population*/
void init(population *pop_ptr);

void init(population *pop_ptr)
{
    int i,j;
    float d;
    pop_ptr->ind_ptr = &(pop_ptr->ind[0]);

    /*Loop Over the population size*/
    for (i = 0 ; i < popsize ; i++)
    {
        /*Loop over the chromosome length*/
        for (j = 0;j < chrom;j++)
        {
            /*Generate a Random No. if it is less
than 0.5 it
generates a 0 in the string otherwise
1*/
            d = randomperc();
            if(d >= 0.5)
            {
                pop_ptr->ind_ptr->genes[j] = 1;
            }
            else
            {
                pop_ptr->ind_ptr->genes[j] = 0;
            }
        }
        pop_ptr->ind_ptr = &(pop_ptr->ind[i+1
]);
    }
    pop_ptr->ind_ptr = &(pop_ptr->ind[0]);
    return;
}

```

## input.h

```
/*This is a file to get the input for the GA
program*/

void input(FILE *rep_ptr);

void input(FILE *rep_ptr)
{
    int i;
    float cc;
    int vlen_c, vlen_i, div;

    nchrom = 0;
    nvar = section_n*3;

    /* SPECIFY NUMBER OF FITNESS FUNCTIONS AND
    CONSTRAINTS */
    nfunc = 3;
    ncons = 3;

    printf("\nInitializing parameters:\n");
    printf("\tNumber of Objective Functions:
\t%d\n", nfunc);
    printf("\tNumber of Constraints: \t%d\n",
ncons);
    printf("\tNumber of Subjects to be
Scheduled: \t%d\n", subject_n);
    printf("\tNumber of Timeslots: \t%d\n",
slot_n);
    printf("\tNumber of Rooms: \t%d\n", room_n
);
    printf("\tNumber of Instructors: \t%d\n",
faculty_n);
    printf("\tNumber of Sections: \t%d\n",
section_n);

    /* GA PARAMETERS */

    printf("\nInitializing GA parameters: \n")
;
    fflush(stdin);

    /* SPECIFY POPULATION SIZE */
    printf("Give Population size (an even no.)
\n");

    if (section_n * 0.6 <= 50) {
        popsize = 50;
    } else if (section_n * 0.6 >= 100){
        popsize = 100;
    } else {
        popsize = section_n * 0.6;
    }

    printf("popsize: %d\n", popsize);

    if(popsize > maxpop)
    {
        printf("Increase Population size.
Currently set %d\n",maxpop);
        exit(1);
    }

    /* SPECIFY NUMBER OF GENERATIONS */
    printf("Give the no.of generations \n");

    if (section_n % 2 == 0) {
        gener = section_n*12;
    } else {
        gener = (section_n+1)*12;
    }

    printf("num generations: %d\n", gener);

    /* Crossover PROBABILITY */
    pcross = 0.1;

    if (nvar > 0)
    {
        /* mutation probability for real-coded
        vectors (between 0 and %f)\n",cc);
        */
        pmut_r = 0.1;

        /* Distribution Index for Real-Coded
        Cross-over (Default = 20)*/
        //di = randomperc() * 25;
        di = 20;

        /* Distribution Index for real-coded
        mutation between 0.5 to 500\n"); */
        dim = 0.5 + (randomperc() * 495.5);

        /*Specify the limits of the variables
        */
        for(i = 0; i < nvar; i++)
        {
            lim_r[i][0] = 0;
            if(i%3==0)
                lim_r[i][1] = slot_n - 0.01;
            else if (i%3==1)
                lim_r[i][1] = room_n - 0.01;
            else
                lim_r[i][1] = faculty_n - 0.01;
        }

        /* If limits are rigid, ans=1 */
        ans = 1;
    }

    chrom = 0;
    if (nchrom > 0)
    {
        /* SPECIFY CROSSOVER TYPE: 1 for simple,
        2 for uniform crossover */
        optype = 1;

        /* SPECIFY NUMBER OF BITS & LIMITS FOR
        EACH VARIABLE */
        div = subject_n;
        for(vlen_c=0, div=subject_n; div>=1; div
=div/2)
            vlen_c++;
        for(vlen_i=0, div=faculty_n; div>=1; div
=div/2)
            vlen_i++;

        printf("vlen_c: %d, vlen_i: %d\n",
vlen_c, vlen_i);
        for (i = 0; i < nchrom; i++)
        {
            lim_b[i][0] = 0;
            if(i%2==0){
                vlen[i] = vlen_c;
                lim_b[i][1] = subject_n;
            } else {
                vlen[i] = vlen_i;
                lim_b[i][1] = faculty_n;
            }
            chrom += vlen[i];
        }
    }
}
```

```

if(chrom > maxchrom)
{
    printf("Increase chromosome size.
           Currently set %d.\n",maxchrom);
    exit(1);
}

/* Ask for mutation probablity */
cc = 1.0/chrom;
pmut_b = cc * 0.5;
}

printf("-----\n");
-----\n");

// end of reading parameters
return;
}

```

## decode.h

```

/*This is the program to decode the
  chromosome to get real values*/

void decode(population *pop_ptr);

void decode(population *pop_ptr)
{
    float *real_ptr;
    int i,sum,b,k,c,d,*gene_ptr,m,x;

    pop_ptr->ind_ptr = &(pop_ptr->ind[0]);

    for(i = 0; i < popsize; i++)
    {
        real_ptr = &(pop_ptr->ind_ptr->xbin[0]
        );
        gene_ptr = &(pop_ptr->ind_ptr->genes[
        ]);
        for(m = 0; m < nchrom; m++)
        {
            /*finding out the co-efficient 2 to the
              power of
              (l-1) where l is the no of bits
              assigned to this variable

              For More Info Study DEB's Book*/
            sum = 0;
            for(k = 0; k < vlen[m]; k++)
            {
                b = *gene_ptr;
                d = vlen[m] - k - 1;
                c = pow(2,d);
                sum =sum + c * b;
                gene_ptr++;
            }

            x = vlen[m];
            coef[m] = pow(2,x) - 1;
            *real_ptr =lim_b[m][0] + (sum/coef[m])*(
                lim_b[m][1]-lim_b[m][0]);
            real_ptr++;
        }
        pop_ptr->ind_ptr = &(pop_ptr->ind[i+1]
        );
    }
    return ;
}

```

## ranking.h

```

/*This also demarkates the different Pareto
  Fronts*/

void ranking(population *pop_ptr);

int indcmp(float *ptr1,float *ptr2);

void ranking(population *pop_ptr)
{
    int i,j,k,          /*counters*/
    rnk,              /*rank*/
    val,              /*value obtained after
                      comparing two individuals*/
    nondom,          /*no of non dominated
                      members*/
    maxrank1,        /*Max rank of the
                      population*/
    rankarr[maxpop], /*Array storing the
                      individual number at a rank*/
    q;

    float *ptr1,*ptr2;

    /*-----* RANKING
      *-----*/

    /*Initializing the ranks to zero*/
    rnk = 0 ;

    nondom = 0 ;
    maxrank1 = 0;

    /*min_fit is initialize to start
      distributing the dummy fitness =
      popsize to the rank one individuals and
      keeping the record such
      that the minimum fitness of the better
      rank individual is always
      greater than max fitness of the
      relatively worse rank*/

    /*Difference in the fitness of minimum
      dummy fitness of better rank
      and max fitness of the next ranked
      individuals*/

    /*Initializing all the flags to 2*/
    for( j = 0 ;j < popsize; j++)
    {
        pop_ptr->ind[j].flag = 2;
    }

    q = 0;

    for(k = 0;k < popsize;k++,q=0)
    {
        for(j = 0;j < popsize;j++){
            if (pop_ptr->ind[j].flag != 1)break;
            /*Break if all the individuals are
              assigned a rank*/
        }
        if(j == popsize)break;

        rnk = rnk + 1;

        for( j = 0 ;j < popsize; j++)
        {
            if(pop_ptr->ind[j].flag == 0)
                pop_ptr->ind[j].flag = 2;
        }
    }
}

```

```

        }
        } /*Loop over flag check ends*/
    } /*Loop over i ends */
    pop_ptr->rankno[rank-1] = q;
}
maxrank1 = rank;

/* Find Max Rank of the population */
for(i = 0;i < popsize;i++)
{
    rank = pop_ptr->ind[i].rank;
    if(rank > maxrank1)maxrank1 = rank;
}

pop_ptr->maxrank = maxrank1;

return;
}

/*Routine Comparing the two individuals*/
int indcmp(float *ptr1,float *ptr2)
{
    float fit1[maxfunc],fit2[maxfunc];
    int i,value,m,n;
    for(i = 0;i < nfunc ;i++)
    {
        fit1[i] = *ptr1++;
        fit2[i] = *ptr2++;
    }
    m = 0;
    n = 0;
    while(m < nfunc && fit1[m] <= fit2[m])
    {
        if((fit2[m] - fit1[m]) < 1e-7) n++;
        m++;
    }
    if(m == nfunc)
    {
        if(n == nfunc) value = 3;
        else value = 1; /*value = 1 for
        domination*/
    }
    else
    {
        m = 0;
        n = 0;
        while(m < nfunc && fit1[m] >= fit2[m])
        {
            if((fit1[m] - fit2[m]) < 1e-7) n++;
            m++;
        }
        if(m == nfunc)
        {
            if(n != nfunc)
            value = 2; /*value = 2 for
            dominated */
            else value = 3;
        }
        else value = 3; /*value = 3 for
        incomparable */
    }
}

return value;
}

/*Set the flag of dominated
individuals to 2*/
}

for(i = 0;i < popsize ; i++)
{
    /*Select an individual which rank to
    be assigned*/
    pop_ptr->ind_ptr = &(pop_ptr->ind[i]);
    if(pop_ptr->ind_ptr->flag != 1 &&
    pop_ptr->ind_ptr->flag != 0)
    {
        ptr1 = &(pop_ptr->ind_ptr->fitness[0
        ]);
        for(j = 0;j < popsize ; j++){
            /*Select the other individual
            which has not got a rank*/
            if( i != j)
            {
                if(pop_ptr->ind[j].flag != 1)
                {
                    pop_ptr->ind_ptr = &(pop_ptr->
                    ind[j]);
                    ptr2 = &(pop_ptr->ind_ptr->
                    fitness[0]);

                    /*Compare the two individuals
                    for fitness*/
                    val = indcmp(ptr1,ptr2);

                    /* VAL = 1 if individual i
                    dominates j */
                    /*VAL = 2 if individual i is
                    dominated by ind j */
                    /*VAL = 3 for non comparable
                    individuals*/

                    if( val == 2)
                    {
                        pop_ptr->ind[i].flag = 0; /*
                        individual 1 is
                        dominated */
                        break;
                    }

                    if(val == 1)
                    {
                        pop_ptr->ind[j].flag = 0; /*
                        individual 2 is
                        dominated */
                    }

                    if(val == 3)
                    {
                        nondom++; /* individual 1 & 2
                        are non dominated */
                        if(pop_ptr->ind[j].flag != 0
                        )
                            pop_ptr->ind[j].flag = 3;
                    }
                } /*if loop ends*/
            } /* i != j loop ends*/
        } /*loop over j ends*/
    }
    if( j == popsize)
    {
        /*Assign the rank and set the flag
        */
        pop_ptr->ind[i].rank = rank;
        pop_ptr->ind[i].flag = 1;
        rankarr[q] = i;
        q++;
    }
}

```

## ranc-con.h

```
/*This demarkates the different Pareto
Fronts*/

void rankcon(population *pop_ptr);

int indcmp3(float *ptr1,float *ptr2);

void rankcon(population *pop_ptr)
{
int i,j,k,          /*counters*/
  rnk,              /*rank*/
  val,              /*value obtained after
                    comparing two individuals*/
  nondom,          /*no of non dominated
                    members*/
  maxrank1,        /*Max rank of the
                    population*/
  rankarr[maxpop], /*Array storing the
                    individual number at a rank*/
  q;

float *ptr1,*ptr2,*err_ptr1,*err_ptr2;

/*-----* RANKING *-----*/

/*Initializing the ranks to zero*/
rnk = 0 ;

nondom = 0 ;
maxrank1 = 0;
/*min_fit is initialize to start
distributing the dummy fitness =
popsize to the rank one individuals and
keeping the record such
that the minimum fitness of the better
rank individual is always
greater than max fitness of the relatively
worse rank*/

min_fit = popsize;

/*Difference in the fitness of minimum dummy
fitness of better rank
and max fitness of the next ranked
individuals*/

delta_fit = 0.1 * popsize;

/*Initializing all the flags to 2*/
for( j = 0 ;j < popsize; j++)
{
  pop_ptr->ind[j].flag = 2;
}

q = 0;

for(k = 0;k < popsize;k++,q=0)
{
  for(j = 0;j < popsize;j++)
  {
    if (pop_ptr->ind[j].flag != 1)break;
    /*Break if all the individuals are
    assigned a rank*/
  }

  if(j == popsize)break;

  rnk = rnk + 1;

  for( j = 0 ;j < popsize; j++)
  {
    if(pop_ptr->ind[j].flag == 0) pop_ptr->
ind[j].flag = 2;
    /*Set the flag of dominated individuals
to 2*/
  }

  for(i = 0;i < popsize ; i++)
  {
    /*Select an individual which rank to be
assigned*/

    pop_ptr->ind_ptr = &(pop_ptr->ind[i]);

    if(pop_ptr->ind_ptr->flag != 1 &&
pop_ptr->ind_ptr->flag != 0)
    {
      ptr1 = &(pop_ptr->ind_ptr->fitness[0])
;
      err_ptr1 = &(pop_ptr->ind_ptr->error);

      for(j = 0;j < popsize ; j++)
      {
        /*Select the other individual which
has not got a rank*/
        if( i!= j)
        {
          if(pop_ptr->ind[j].flag != 1)
          {
            pop_ptr->ind_ptr = &(pop_ptr->
ind[j]);
            ptr2 = &(pop_ptr->ind_ptr->
fitness[0]);
            err_ptr2 = &(pop_ptr->ind_ptr->
error);

            if(*err_ptr1 < 1.0e-6 && *
err_ptr2 > 1.0e-6)
            {
              /*first ind is feasible second
individaul is infeasible*/
              pop_ptr->ind[j].flag = 0;
            }
            else
            {
              if(*err_ptr1 > 1.0e-6 && *
err_ptr2 < 1.0e-6)
              {
                /*first individual is
infeasible and second is
feasible*/
                pop_ptr->ind[i].flag = 0;
                break;
              }
              else
              {
                /*both are feasible or both
are infeasible*/
                if(*err_ptr1 > *err_ptr2)
                {
                  pop_ptr->ind[i].flag = 0;
                  /*first individual is more
infeasible*/
                  break;
                }
                else
                {
                  if(*err_ptr1 < *err_ptr2)
                  {

```

```

pop_ptr->ind[j].flag = 0
;
/*second individual is
more infeasible*/
}
else
{
/*Compare the two
individuals for
fitness*/
val = indcmp3(ptr1,ptr2)
;

/*VAL = 2 for dominated
individual which
rank to be given*/
/*VAL = 1 for dominating
individual which
rank to be given*/
/*VAL = 3 for non
comparable
individuals*/

if( val == 2)
{
pop_ptr->ind[i].flag =
0;
/* individual 1 is
dominated */
break;
}

if(val == 1)
{
pop_ptr->ind[j].flag =
0;
/* individual 2 is
dominated */
}

if(val == 3)
{
nondom++;
/* individual 1 & 2
are non dominated
*/
if(pop_ptr->ind[j].
flag != 0)
pop_ptr->ind[j].flag
= 3;
}
} /*if loop ends*/
} /* i != j loop ends
*/
}
}
} /*loop over j ends*/
if( j == popsize)
{
/*Assign the rank and set the flag*/
pop_ptr->ind[i].rank = rnk;
pop_ptr->ind[i].flag = 1;
rankarr[q] = i;
q++;
}
} /*Loop over flag check ends*/
} /*Loop over i ends */
pop_ptr->rankno[rnk-1] = q ;
}
maxrank1 = rnk;
}

/* Find Max Rank of the population */
for(i = 0;i < popsize;i++)
{
rnk = pop_ptr->ind[i].rank;

if(rnk > maxrank1)maxrank1 = rnk;
}

pop_ptr->maxrank = maxrank1;
return;
}

/*Routine Comparing the two individuals*/
int indcmp3(float *ptr1,float *ptr2)
{
float fit1[maxfun],fit2[maxfun];
int i,value,m,n;
for(i = 0;i < nfunc ;i++)
{
fit1[i] = *ptr1++;
fit2[i] = *ptr2++;
}
m = 0;
n = 0;
while(m < nfunc && fit1[m] <= fit2[m])
{
if(fit1[m]== fit2[m]) n++;
m++;
}
if(m == nfunc)
{
if(n == nfunc) value = 3;
else value = 1; /*value = 1
for domination*/
}
else
{
m = 0;
n = 0;
while(m < nfunc && fit1[m] >= fit2[m])
{
if(fit1[m]== fit2[m]) n++;
m++;
}
if(m == nfunc)
{
if(n != nfunc)
value = 2; /*
value = 2 for dominated */
else value =3;
}
else value = 3; /*value
= 3 for incomparable*/
}
}
return value;
}

```

## func-con.h (class scheduling)

```

/*This is the program used to evaluate the
value of the function & errors
*****
*****
void func(population *pop_ptr);

#include "evaluation.h" /*for population
evaluation - determine penalties*/

void func(population *pop_ptr)
{
    /*File ptr to the file to store the value
    of the g for last iteration
    g is the parameter required for a
    particular problem
    Every problem is not required*/

    float fsum = 0, min_f[3], min_err,
          min_fsum, ave_fsum;
    int **schedOverlap = overlaps;

    float *realx_ptr, /*Pointer to the array
    of x values*/
          *binx_ptr, /* Pointer to the binary
    variables */
          *fitn_ptr, /*Pointer to the array
    of fitness function*/
    x[2*maxvar], /* problem variables */
    f[maxfun], /*array of fitness values
    */
    *err_ptr, /*Pointer to the error */
    cstr[maxcons];

    int i,j,k;
    float error, cc;

    /*float fsum = 0, ave_fsum = 0, min_f[
    maxfun], min_err, min_fsum; */

    eval_ptr = fopen("eval_values.out", "w");
    pop_ptr->ind_ptr = &(pop_ptr->ind[0]);

    /*Initializing the max rank to zero*/
    pop_ptr->maxrank = 0;
    for(i = 0; i < popsize; i++)
    {
        fprintf(eval_ptr, "EVALUATION AT
        INDIVIDUAL %d\n", (i+1));
        pop_ptr->ind_ptr = &(pop_ptr->ind[i]);
        realx_ptr = &(pop_ptr->ind_ptr->xreal[0]
        );
        binx_ptr = &(pop_ptr->ind_ptr->xbin[0]);

        for(j = 0; j < nvar; j++)
        { // Real-coded variables
            x[j] = *realx_ptr++;
        }

        for(j = 0; j < nchrom; j=j+3)
        { // Binary-coded variables
            x[nvar+j] = *binx_ptr++;
        }

        for(j=0; j<nvar; j++){
            k = (int) x[j];
            fprintf(eval_ptr, "(%d/%d/%d) ", k, (
            int) x[j+1], (int) x[j+2]);
        }
        fprintf(eval_ptr, "\n");

        fitn_ptr = &(pop_ptr->ind_ptr->fitness[0
        ]);
        err_ptr = &(pop_ptr->ind_ptr->error);

        /* DO NOT CHANGE ANYTHING ABOVE */
        /*-----CODE YOUR
        OBJECTIVE FUNCTIONS HERE-----
        */
        /*All functions must be of minimization
        type, negate maximization
        functions
        */
        /*=====Start Coding Your Function
        From This Point=====*/

        f[0] = determineAllowedRooms (x, nvar); +
        getSubjectRoomCompatibility (x, nvar)
        ;
        f[1] = getDemandFitness (x, nvar); +
        getYearOfferedClash (x, nvar,
        schedOverlap);
        f[2] = getFacultyPreferredSched (x, nvar)
        ; /*+ getConsecutiveSlots (x, nvar);
        */ + getFacultyLoad (x, nvar); +
        getSubjectFacultyFitness (x, nvar);

        /******
        *****
        Put The Constraints Here
        */
        /******
        *****
        // g(x) >= 0 type (normalize g(x) as in
        the cstr[1] below)
        /*=====Start Coding Here=====
        ==*/
        cstr[0] = -getRoomClash (x, nvar,
        schedOverlap);
        cstr[1] = -getFacultyClash (x, nvar,
        schedOverlap);
        cstr[2] = -evaluateSlotType (x, nvar);
        /* cstr[3] = -getSubjectRoomCompatibility
        (x, nvar);
        cstr[4] = -getYearOfferedClash (x, nvar,
        schedOverlap);*/
        /*=====End Your Coding Upto This
        Point=====*/

        if(nfunc>0){
            fprintf(eval_ptr, "FITNESS: ");
            for(k = 0; k<nfunc; k++)
                fprintf(eval_ptr, "%f ", f[k]);
            fprintf(eval_ptr, "\n");
        }
        if(ncons>0){
            fprintf(eval_ptr, "CONSTRAINT: ");
            for(k = 0; k<ncons; k++)
                fprintf(eval_ptr, "%f ", cstr[k]);
            fprintf(eval_ptr, "\n");
        }

        for(k = 0 ; k < nfunc ; k++)
        {
            *fitn_ptr++ = f[k];
            //pop_ptr->ind_ptr->fitness[k] = f[k];
        }

        for (k = 0; k < ncons; k++)
        {
            pop_ptr->ind_ptr->constr[k] = cstr[k];
        }
        error = 0.0;
    }
}

```



```

for (k = 0; k < ncons; k++)
{
    cc = cstr[k];
    if(cc < 0.0)
        error = error - cc;
}
*err_ptr = error;

/*
fprintf(eval_ptr, "\tError: %f, %f\n",
    error, *err_ptr);
fprintf(eval_ptr, "-----
-----\n\n");
fsum = (f[0] + f[1] + f[2]) / 3;
fprintf(plot_ptr, "%d,%f,%f,%f,%f,%f\n",
    i, f[0],f[1],f[2], fsum, error);
*/

fsum = 0;
if(i==0){
    for(j=0; j<nfunc; j++){
        min_f[j] = f[j];
        fsum = fsum + f[j];
    }
    ave_fsum = fsum;
    min_fsum = fsum;
    min_err = error;
} else {
    for(j=0; j<nfunc; j++){
        if(f[j]<min_f[j])
            min_f[j] = f[j];
        fsum = fsum + f[j];
    }
    if(fsum < min_fsum)
        min_fsum = fsum;
    if(error<min_err)
        min_err = error;
    ave_fsum = ave_fsum + fsum;
}
}

fclose(eval_ptr);
/*-----* RANKING *--
-----*/

printf(" (Min Error: %f ) ", min_err);

ave_fsum = ave_fsum/popsize;
for(j=0; j<nfunc; j++){
    fprintf(test_ptr, "%f ", min_f[j]);
}
fprintf(test_ptr, "%f %f %f\n", min_fsum,
    ave_fsum, min_err);

if(ncons == 0)
    ranking(pop_ptr);
else
    rankcon(pop_ptr);

return;
}

```

## func-con.h (final exam scheduling)

```

/*This is the program used to evaluate the
value of the function & errors
*****
******/
void func(population *pop_ptr);
#include "evaluation.h" /*for population
evaluation - determine penalties*/

void func(population *pop_ptr)
{
    /*File ptr to the file to store the value
of the g for last iteration
g is the parameter required for a
particular problem
Every problem is not required*/

```

```

float fsum = 0, min_f[3], min_err,
    min_fsum, ave_fsum;
int **schedOverlap = overlaps;

float *realx_ptr, /*Pointer to the array
of x values*/
*binx_ptr, /* Pointer to the binary
variables */
*fitn_ptr, /*Pointer to the array
of fitness function*/
x[2*maxvar], /* problem variables */
f[maxfun], /*array of fitness values
*/
*err_ptr, /*Pointer to the error */
cstr[maxcons];

int i,j,k;
float error, cc;

/*float fsum = 0, ave_fsum = 0, min_f[
maxfun], min_err, min_fsum; */

eval_ptr = fopen("eval_values.out", "w");
pop_ptr->ind_ptr = &(pop_ptr->ind[0]);

/*Initializing the max rank to zero*/
pop_ptr->maxrank = 0;
for(i = 0; i < popsize; i++)
{
    fprintf(eval_ptr, "EVALUATION AT
INDIVIDUAL %d\n", (i+1));
    pop_ptr->ind_ptr = &(pop_ptr->ind[i]);
    realx_ptr = &(pop_ptr->ind_ptr->xreal[0]);
    binx_ptr = &(pop_ptr->ind_ptr->xbin[0]);

    for(j = 0; j < nvar; j++)
    { // Real-coded variables
        x[j] = *realx_ptr++;
    }

    for(j = 0; j < nchrom; j=j+2)
    { // Binary-coded variables
        x[nvar+j] = *binx_ptr++;
    }

    for(j=0; j<nvar; j++){
        k = (int) x[j];
        fprintf(eval_ptr, "%d/%d ", k, (int)
            x[j+1]);
    }
    fprintf(eval_ptr, "\n");

    fitn_ptr = &(pop_ptr->ind_ptr->fitness[0
    ]);

    err_ptr = &(pop_ptr->ind_ptr->error);

    /* DO NOT CHANGE ANYTHING ABOVE */
    /*-----CODE YOUR
OBJECTIVE FUNCTIONS HERE-----
*/
    /*All functions must be of minimization
type, negate maximization functions
*/
    /*=====Start Coding Your Function
From This Point=====*/

    f[0] = determineAllowedRooms(x, nvar); +
        getSubjectRoomCompatibility(x, nvar)
        ;
    f[1] = getDemandFitness(x, nvar) +
        getYearOfferedClash(x, nvar,
        schedOverlap); + getSubjectClash(x,
        nvar, schedOverlap);

```

```

/*****
*****
***** Put The Constraints Here
*/
/*****
*****
// g(x) >= 0 type (normalize g(x) as in
the cstr[1] below)
/*****Start Coding Here*****
*****/
cstr[0] = -getRoomClash(x, nvar,
schedOverlap);
cstr[1] = -evaluateSlotType(x, nvar);

/*****End Your Coding Upto This
Point*****/

if(nfunc>0){
fprintf(eval_ptr, "FITNESS: ");
for(k = 0; k<nfunc; k++){
fprintf(eval_ptr, "%f ", f[k]);
fprintf(eval_ptr, "\n");
}
}
if(ncons>0){
fprintf(eval_ptr, "CONSTRAINT: ");
for(k = 0; k<ncons; k++){
fprintf(eval_ptr, "%f ", cstr[k]);
fprintf(eval_ptr, "\n");
}
}

for(k = 0 ; k < nfunc ;k++)
{
*fitn_ptr++ = f[k];
//pop_ptr->ind_ptr->fitness[k] = f[k];
}

for (k = 0;k < ncons; k++)
{
pop_ptr->ind_ptr->constr[k] = cstr[k];
}
error = 0.0;
for (k = 0;k < ncons;k++)
{
cc = cstr[k];
if(cc < 0.0)
error = error - cc;
}
*err_ptr = error;

/*
fprintf(eval_ptr, "\tError: %f, %f\n",

error, *err_ptr);
fprintf(eval_ptr, "-----
-----\n\n");
fsum = (f[0] + f[1] + f[2]) / 3;
fprintf(plot_ptr, "%d,%f,%f,%f,%f,%f\n",
i, f[0],f[1],f[2], fsum, error);
*/

fsum = 0;
if(i==0){
for(j=0; j<nfunc; j++){
min_f[j] = f[j];
fsum = fsum + f[j];
}
ave_fsum = fsum;
min_fsum = fsum;
min_err = error;
}

```

```

} else {
for(j=0; j<nfunc; j++){
if(f[j]<min_f[j])
min_f[j] = f[j];
fsum = fsum + f[j];
}
if(fsum < min_fsum)
min_fsum = fsum;
if(error<min_err)
min_err = error;
ave_fsum = ave_fsum + fsum;
}
}

fclose(eval_ptr);
/***** RANKING *****/
-----*/

printf(" (Min Error: %f ) ", min_err);

ave_fsum = ave_fsum/popsize;
for(j=0; j<nfunc; j++){
fprintf(test_ptr, "%f ", min_f[j]);
}
fprintf(test_ptr, "%f %f %f\n", min_fsum,
ave_fsum, min_err);

if(ncons == 0)
ranking(pop_ptr);
else
rankcon(pop_ptr);

return;
}

```

## select.h

```

/*This is the file to get the different
individuals selected*/

void nselect(population *old_pop_ptr,
population *pop2_ptr);

void nselect(population *old_pop_ptr,
population *pop2_ptr)
{
int *fit_ptr1,*fit_ptr2;

float rnd2,*f1_ptr,*f2_ptr;

int *sl_ptr,*s2_ptr,*select_ptr;
float *select_ptr_r, *sl_ptr_r, *s2_ptr_r;

void *j,*j1;

int i,rnd,rnd1,k,n,j2,r,s;

old_pop_ptr->ind_ptr = &(amp;old_pop_ptr->ind[0]);

pop2_ptr->ind_ptr= &(amp;pop2_ptr->ind[0]);

j = &(old_pop_ptr->ind[popsize-1]);

old_pop_ptr->ind_ptr = &(amp;old_pop_ptr->ind[0]);

j2 = 0;
r = popsize;
s = chrom;
}

```

```

for(n = 0, k = 0; n < popsize; n++, k++)
{
    pop2_ptr->ind_ptr = &(pop2_ptr->ind[k]);
    select_ptr = &(pop2_ptr->ind_ptr->genes[
        0]);
    select_ptr_r = &(pop2_ptr->ind_ptr->
        xreal[0]);

    rnd2 = randomperc();

    rnd2 = popsize* rnd2;

    rnd = floor(rnd2);

    if(rnd == 0)
        rnd = popsize - k;

    if(rnd == popsize)
        rnd = (popsize-2)/2;

    /*Select first parent randomly*/
    j = &(old_pop_ptr->ind[rnd-1]);

    rnd2 = randomperc();

    rnd2 = popsize * rnd2;

    rnd1 = floor(rnd2);

    if (rnd1 == 0)
        rnd1 = popsize - n;

    if(rnd1 == popsize)
        rnd1 = (popsize - 4)/2;

    /*Select second parent randomly*/

    j1 = &(old_pop_ptr->ind[rnd1-1]);

    old_pop_ptr->ind_ptr = j;

    s1_ptr = &(old_pop_ptr->ind_ptr->genes[0
        ]);
    s1_ptr_r = &(old_pop_ptr->ind_ptr->xreal
        [0]);
    fit_ptr1 = &(old_pop_ptr->ind_ptr->rank)
        ;
    f1_ptr = &(old_pop_ptr->ind_ptr->cub_len
        );

    old_pop_ptr->ind_ptr = j1;
    s2_ptr = &(old_pop_ptr->ind_ptr->genes[0
        ]);
    s2_ptr_r = &(old_pop_ptr->ind_ptr->xreal
        [0]);
    fit_ptr2 = &(old_pop_ptr->ind_ptr->rank)
        ;
    f2_ptr = &(old_pop_ptr->ind_ptr->cub_len
        );

    /*-----
        */

    /*-----SELECTION PROCEDURE-----
        */

    /*Comparing the fitnesses*/

    if(*fit_ptr1 > *fit_ptr2)
    {
        for(i = 0; i < chrom; i++)
            *select_ptr++=*s2_ptr++;
        for(i = 0; i < nvar; i++)
            *select_ptr_r++=*s2_ptr_r++;
    }
    else
    {

```

```

        if(*fit_ptr1 < *fit_ptr2)
        {
            for(i = 0; i < chrom; i++)
                *select_ptr++=*s1_ptr++;
            for(i = 0; i < nvar; i++)
                *select_ptr_r++=*s1_ptr_r++;
        }
        else
        {
            if(*f1_ptr < *f2_ptr)
            {
                for(i = 0; i < chrom; i++)
                    *select_ptr++=*s2_ptr++;
                for(i = 0; i < nvar; i++)
                    *select_ptr_r++=*s2_ptr_r++;
            }
            else
            {
                for(i = 0; i < chrom; i++)
                    *select_ptr++=*s1_ptr++;
                for(i = 0; i < nvar; i++)
                    *select_ptr_r++=*s1_ptr_r++;
            }
        }
    }
}

return;
}

```

## crossover.h

```

/*This is the file for formulating the
crossover process*/

void crossover(population *new_pop_ptr,
    population *mate_pop_ptr) ;

void crossover(population *new_pop_ptr,
    population *mate_pop_ptr)
{
    int i, k, n, y, mating_site, *par1, *par2, *chld1
        , *chld2, c;
    float rnd;
    rnd=randomperc();

    new_pop_ptr->ind_ptr=&(new_pop_ptr->ind[0]
        );

    mate_pop_ptr->ind_ptr=&(mate_pop_ptr->ind[
        0]);

    for (i = 0, y = 0, n = 0; i < popsize/2; i++)
    {
        new_pop_ptr->ind_ptr = &(new_pop_ptr->
            ind[n]);
        chld1=&(new_pop_ptr->ind_ptr->genes[0]);
        n = n+1;

        new_pop_ptr->ind_ptr = &(new_pop_ptr->
            ind[n]);
        chld2=&(new_pop_ptr->ind_ptr->genes[0]);
        n = n+1;
    }
}

```

```

mate_pop_ptr->ind_ptr = &(mate_pop_ptr->
ind[y]);
par1 = &(mate_pop_ptr->ind_ptr->genes[0]
);
y = y+1;

mate_pop_ptr->ind_ptr = &(mate_pop_ptr->
ind[y]);
par2 = &(mate_pop_ptr->ind_ptr->genes[0]
);
y = y+1;

rnd = randomperc();
if (rnd < pcross)
{
ncross++;
rnd = randomperc();
c = floor(rnd*(chrom+10));
mating_site = c;
if(mating_site >= chrom)
{
mating_site = mating_site/2;
}

for(k = 0;k < chrom;k++)
{
if(k > mating_site-1)
{
*chld1++ = *par2++;
*chld2++ = *par1++;
}
else
{
*chld1++ = *par1++;
*chld2++ = *par2++;
}
}
}
else
{
for (k = 0;k < chrom;k++)
{
*chld1++ = *par1++;
*chld2++ = *par2++;
}
}
}
return;
}

```

## uniformxr.h

```

/* This is the header file to do the uniform
crossover */

void unicross(population *new_pop_ptr,
population *mate_pop_ptr);

void unicross(population *new_pop_ptr,
population *mate_pop_ptr)
{
int i, j, *gene, y, n, *par1, *par2, *chld1, *
chld2;
float rnd;
for(i = 0, y = 0, n = 0; i < popsize; i++)
{
for(j = 0; j < chrom; j++)
{
/*Select a bit for doing cross-over*/
new_pop_ptr->ind_ptr = &(new_pop_ptr->
ind[y]);
chld1 = &(new_pop_ptr->ind_ptr->genes[
j]);

new_pop_ptr->ind_ptr = &(new_pop_ptr->
ind[y+1]);
chld2 = &(new_pop_ptr->ind_ptr->genes[
j]);
}
}
}

```

```

mate_pop_ptr->ind_ptr = &(mate_pop_ptr
->ind[n]);
par1 = &(mate_pop_ptr->ind_ptr->genes[
j]);

mate_pop_ptr->ind_ptr = &(mate_pop_ptr
->ind[n+1]);
par2 = &(mate_pop_ptr->ind_ptr->genes[
j]);

rnd = randomperc();

/*Checking whether to do cross-over or
not*/
if(rnd <= pcross)
{
ncross++;
*chld1 = *par2;
*chld2 = *par2;
}
else
{
*chld1 = *par1;
*chld2 = *par2;
}

y = y+2;
n = n+2;

}

for(i = 0; i < popsize; i++)
{
new_pop_ptr->ind_ptr = &(new_pop_ptr->
ind[i]);
gene = &(new_pop_ptr->ind_ptr->genes[0])
;
for(j = 0; j < chrom; j++)
{
gene = &(new_pop_ptr->ind_ptr->genes[j
]);
}
}
return;
}

```

## realcross2.h

```

/*This is the file used for crossover for
Real Coded GA*/

void realcross(population *new_pop_ptr,
population *mate_pop_ptr);

void realcross(population *new_pop_ptr,
population *mate_pop_ptr)
{
int i, j, y, n;
float rnd, par1, par2, chld1, chld2, betaq, beta
, alpha;
float y1, y2, yu, yl, exp;

y=0; n=0;

for(i = 0; i < popsize/2; i++)
{
rnd = randomperc();

/*Check Whether the cross-over to be
performed*/
if(rnd <= pcross)
{

```

```

/*Loop over no of variables*/
for(j = 0; j < nvar; j++)
{
    /*Selected Two Parents*/
    par1 = mate_pop_ptr->ind[y].xreal[j]
    ;
    par2 = mate_pop_ptr->ind[y+1].xreal[
    j];

    y1 = lim_r[j][0];
    yu = lim_r[j][1];

    rnd = randomperc();

    /* Check whether variable is
    selected or not*/
    if(rnd <= 0.5)
    {
        /*Variable selected*/
        ncross++;

        if(fabs(par1 - par2) > 0.000001)
            // changed by Deb (31/10/01)
        {
            if(par2 > par1)
            {
                y2 = par2;
                y1 = par1;
            }
            else
            {
                y2 = par1;
                y1 = par2;
            }

            /*Find beta value*/
            if((y1 - y1) > (yu - y2))
            {
                beta = 1 + (2*(yu - y2)/(y2 -
                y1));
                //printf("more beta = %f\n",
                beta);
            }
            else
            {
                beta = 1 + (2*(y1-y1)/(y2-y1))
                ;
                //printf("less beta = %f\n",
                beta);
            }

            /*Find alpha*/
            expp = di + 1.0;
            beta = 1.0/beta;
            alpha = 2.0 - pow(beta, expp);

            if (alpha < 0.0)
            {
                printf("ERRORRORR %f %d %d %f %f
                \n", alpha, y, n, par1, par2);
                exit(-1);
                //alpha = -alpha;
            }

            rnd = randomperc();

            if (rnd <= 1.0/alpha)
            {
                alpha = alpha*rnd;
                expp = 1.0/(di+1.0);
                betaq = pow(alpha, expp);
            }
            else
            {
                alpha = alpha*rnd;
                alpha = 1.0/(2.0-alpha);
            }

            if (alpha < 0.0)
            {
                printf("ERRORRORR \n");
                exit(-1);
                //alpha = -alpha;
            }
            betaq = pow(alpha, expp);
        }

        /*Generating two children*/
        chld1 = 0.5*((y1+y2) - betaq*(y2
        -y1));
        chld2 = 0.5*((y1+y2) + betaq*(y2
        -y1));
    }
    else
    {
        betaq = 1.0;
        y1 = par1; y2 = par2;

        /*Generation two children*/
        chld1 = 0.5*((y1+y2) - betaq*(y2
        -y1));
        chld2 = 0.5*((y1+y2) + betaq*(
        y2-y1));
    }

    // added by deb (31/10/01)
    if (chld1 < y1) chld1 = y1;
    if (chld1 > yu) chld1 = yu;
    if (chld2 < y1) chld2 = y1;
    if (chld2 > yu) chld2 = yu;
}
else
{
    /*Copying the children to parents
    */
    chld1 = par1;
    chld2 = par2;
}
new_pop_ptr->ind[n].xreal[j] = chld1
;
new_pop_ptr->ind[n+1].xreal[j] =
chld2;
}
}
else
{
    for(j = 0; j < nvar; j++)
    {
        par1 = mate_pop_ptr->ind[y].xreal[j]
        ;
        par2 = mate_pop_ptr->ind[y+1].xreal[
        j];
        chld1 = par1;
        chld2 = par2;
        new_pop_ptr->ind[n].xreal[j] = chld1
        ;
        new_pop_ptr->ind[n+1].xreal[j] =
        chld2;
    }
}
n = n+2; y=y+2;
}
return;
}

```

## realmut1.h

```
/* This is the module used to formulate the
mutation routine*/

void real_mutate(population *new_pop_ptr);

void real_mutate(population *new_pop_ptr)
{
    int i,j;
    float rnd,delta,indi,deltaq;
    float y,y1,yu,val,xy;

    for(j = 0;j < popsize;j++)
    {
        for (i = 0;i < nvar; i++)
        {
            rnd = randomperc();

            /*For each variable find whether to do
            mutation or not*/
            if(rnd <= pmut_r)
            {
                y = new_pop_ptr->ind[j].xreal[i];
                y1 = lim_r[i][0];
                yu = lim_r[i][1];

                if(y > y1)
                {
                    /*Calculate delta*/

                    if((y-y1) < (yu-y))
                        delta = (y - y1)/(yu - y1);
                    else
                        delta = (yu - y)/(yu-y1);

                    rnd = randomperc();

                    indi = 1.0/(dim + 1.0);

                    if(rnd <= 0.5)
                    {
                        xy = 1.0-delta;
                        val = 2*rnd+(1-2*rnd)*(pow(xy, (dim
                        +1)));
                        deltaq = pow(val,indi) - 1.0;
                    }
                    else
                    {
                        xy = 1.0-delta;
                        val = 2.0*(1.0-rnd)+2.0*(rnd-0.5)*
                        (pow(xy, (dim+1)));
                        deltaq = 1.0 - (pow(val,indi));
                    }

                    /*Change the value for the parent */
                    // *ptr = *ptr + deltaq*(yu-y1);
                    // Added by Deb (31/10/01)
                    y = y + deltaq * (yu-y1);
                    if (y < y1) y=y1;
                    if (y > yu) y=yu;
                    new_pop_ptr->ind[j].xreal[i] = y;
                }
                else // y == y1
                {
                    xy = randomperc();
                    new_pop_ptr->ind[j].xreal[i] = xy*(yu
                    - y1) + y1;
                }

                nmut++;
            }
            // ptr++;
        }
    }

    return ;
}
```

## keepaliven.h

```
/*This is a routine to keep the fronts alive
(caring the end problem)*/

void keepalive(population *pop1_ptr,
population *pop2_ptr,population *
pop3_ptr,int gen);

typedef struct
{
    int maxrank, /*Max rank of the global
    population*/
    rankar[2*maxpop][2*maxpop], /*record of
    array of individual numbers at
    a particular rank */
    rankno[2*maxpop]; /*record of
    no. of individuals at a particular
    rank*/

    int genes[2*maxpop][maxchrom],
    rank[2*maxpop], /*rank of
    different individuals*/
    flag[2*maxpop]; /*Setting the
    flag */

    float fitness[2*maxpop][maxfun], /*Fitness
    function values for the different
    individuals*/
    cub_len[2*maxpop], /*
    Dummyfitness*/
    xreal[2*maxpop][maxvar], /*value
    of the decoded variables for
    different individuals */
    xbin[2*maxpop][maxvar], /* binray-
    coded variables */
    error[2*maxpop], /*Error
    Values of the individuals*/
    constr[2*maxpop][maxcons];
}globpop;

/*Population structure for the pool having
both the old as well as new
population*/

globpop globalpop,*global_pop_ptr;

void grank(int gen);
/*Ranking the global pool*/

void grankc(int gen);
/*Ranking the global pool when the
constraints are there*/

int indcmp1(float *ptr1,float *ptr2);
/*Comparison of the variables*/

void gsort(int rnk,int sel);
/*Sorting for the function values in
ascending order*/

void gshare(int rnk);
/*Sharing the fitness*/

void sort(int rnk);

int left,Lastrank;
float fpara1[2*maxpop][2];

void keepalive(population *pop1_ptr,
population *pop2_ptr,population *
pop3_ptr,int gen)
{
    int i,j,jj,k,m,a1,l,front_pop[maxpop],rec;
```

```

int sum,st, str,pool,poolf, sel, r1;
int *gene1_ptr, *gene2_ptr, leftsum, x;
float rnd, a, *gene3_ptr, x3, *gene4_ptr, *
xbin1_ptr, *xbin2_ptr;

/*Forming the global mating pool*/
for(i = 0; i < popsize; i++)
{
    if(nchrom > 0)
    {
        /*Binary Coded GA genes are copied*/
        for(k = 0; k < chrom; k++)
        {
            globalpop.genes[i][k] = pop1_ptr->ind[
            i].genes[k];
            globalpop.genes[i+popsize][k] =
            pop2_ptr->ind[i].genes[k];
        }
        for(k=0; k < nchrom; k++)
        {
            globalpop.xbin[i][k] = pop1_ptr->ind
            [i].xbin[k];
            globalpop.xbin[i+popsize][k] =
            pop2_ptr->ind[i].xbin[k];
        }
        if (nvar > 0)
        {
            /*For Real Coded GA x values are copied
            */
            for(k = 0; k < nvar; k++)
            {
                globalpop.xreal[i][k] = pop1_ptr->
                ind[i].xreal[k];
                globalpop.xreal[i+popsize][k] =
                pop2_ptr->ind[i].xreal[k];
            }
        }

        /*Fitness is copied to the global pool
        */
        for(l = 0; l < nfunc; l++)
        {
            globalpop.fitness[i][l] = pop1_ptr->ind
            [i].fitness[l];
            globalpop.fitness[i+popsize][l] =
            pop2_ptr->ind[i].fitness[l];
        }

        /*Initial;ising the dummyfitness to
        zero */
        globalpop.cub_len[i] = 0;
        globalpop.cub_len[i+popsize] = 0;
        globalpop.error[i] = pop1_ptr->ind[i]
        .error;
        globalpop.error[i+popsize] = pop2_ptr
        ->ind[i].error;
        for (jj=0; jj<ncons; jj++)
        {
            globalpop.constr[i][jj] = pop1_ptr->ind
            [i].constr[jj];
            globalpop.constr[i+popsize][jj] =
            pop2_ptr->ind[i].constr[jj];
        }
    }
}

global_pop_ptr = &(globalpop);

/*Finding the global ranks */
if(ncons == 0)
    grank(gen);
else
    grankc(gen);

m = globalpop.maxrank;

/* Sharing the fitness to get the dummy
fitness */
for(i = 0; i < m; i++)
{
    gshare(i+1);
}

poolf = popsize;
pool = 0;

/*Initializing the flags of population to
zero */
for(i = 0; i < 2*popsize; i++)
{
    globalpop.flag[i] = 0;
}

// decide which all solutions belong to
the pop3
rec = 0;
st = 0;
for(i = 0 ; i < m ; i++)
{
    /* Elitism Applied Here */
    st = pool;
    pool += globalpop.rankno[i];

    if(pool <= popsize)
    {
        for(k = 0; k < 2*popsize ; k++)
        {
            if(globalpop.rank[k] == i+1)
                globalpop.flag[k] = 1;
        }
        pop3_ptr->rankno[i] = globalpop.rankno[i
        ];
    }
    else
    {
        sel = popsize - st;
        Lastrank = i+1;
        pop3_ptr->rankno[i] = sel;
        qsort(i+1, sel);
        break;
    }
}

k = 0;
for(i = 0, k = 0; i < 2*popsize && k <
popsize; i++)
{
    if(nchrom > 0)
    {
        if(globalpop.flag[i] == 1)
        {
            gene1_ptr = &(globalpop.genes[i][0])
            ;
            xbin1_ptr = &(globalpop.xbin[i][0]);
            pop3_ptr->ind_ptr = &(pop3_ptr->ind[
            k]);
            gene2_ptr = &(pop3_ptr->ind_ptr->
            genes[0]);
            xbin2_ptr = &(pop3_ptr->ind_ptr->

```

```

        xbin[0]);
        for(j = 0 ; j < chrom; j++)
        {
            *gene2_ptr++ = *gene1_ptr++;
        }
        for (j=0; j < nchrom; j++)
        *xbin2_ptr++ = *xbin1_ptr++;
    }
    if (nvar > 0)
    {
        if(globalpop.flag[i] == 1)
        {
            gene3_ptr = &(globalpop.xreal[i][0])
            ;
            pop3_ptr->ind_ptr = &(pop3_ptr->ind[
            k]);
            gene4_ptr = &(pop3_ptr->ind_ptr->
            xreal[0]);

            for(j = 0 ;j < nvar;j++)
            {
                *gene4_ptr++ = *gene3_ptr++;
            }
        }
        if(globalpop.flag[i] == 1)
        {
            for(j = 0;j < nfunc;j++)
                pop3_ptr->ind[k].fitness[j] =
                globalpop.fitness[i][j];
            pop3_ptr->ind[k].cub_len = globalpop.
            cub_len[i];
            if(ncons != 0)
                pop3_ptr->ind[k].error = globalpop.
                error[i];
            for (jj=0; jj<ncons; jj++)
                pop3_ptr->ind[k].constr[jj] =
                globalpop.constr[i][jj];
            pop3_ptr->ind[k].rank = globalpop.rank[i
            ];
            k++; // increment the pop3 counter
        }
    }
    pop3_ptr->maxrank = Lastrank;
    return;
}
void grank(int gen)
{
    int i,j,k,rnk,val,nondom,popsize1,gflg[2*
    maxpop],q;
    float *ptr1,*ptr2;
    FILE *gr;
    gr = fopen("g_rank_record.out","a");
    fprintf(gr,"Generation no. = %d\n",gen);
    /*-----* RANKING *-----*/
    /*
    rnk = 0;
    nondom = 0;
    popsize1 = 2*popsize;

    for(i = 0; i < popsize1; i++)
    {
        gflg[i] = 2;
    }

    for(k = 0; k < popsize1; k++)
    {
        q = 0;
        for(j = 0; j < popsize1; j++)
        {
            if (gflg[j] != 1) break;
        }
        if(j == popsize1) break;
        rnk = rnk +1;
        for( j = 0 ;j < popsize1; j++)
        {
            if(gflg[j] == 0) gflg[j] = 2;
        }
        for(i = 0; i < popsize1 ; i++)
        {
            if(gflg[i] != 1 && gflg[i] != 0)
            {
                ptr1 = &(global_pop_ptr->fitness[i][
                0]);
                for(j = 0; j < popsize1 ; j++)
                {
                    if( i!= j)
                    {
                        if(gflg[j] != 1)
                        {
                            ptr2 = &(global_pop_ptr->fitness[j][
                            0]);
                            val = indcmp1(ptr1,ptr2);
                            if( val == 2)
                            {
                                gflg[i] = 0; /* individual 1 is
                                dominated */
                                break;
                            }
                            if(val == 1)
                            {
                                gflg[j] = 0; /* individual 2 is
                                dominated */
                            }
                            if(val == 3)
                            {
                                nondom++; /* individual 1 & 2 are
                                non dominated */
                                if(gflg[j] != 0)gflg[j] = 3;
                            }
                        }
                    }
                }
            }
            if( j == popsize1)
            {
                global_pop_ptr->rank[i] = rnk;
                gflg[i] = 1;
                global_pop_ptr->rankar[rnk-1][q] = i;
                q++;
            }
        }
        global_pop_ptr->rankno[rnk-1] = q;
    }
    global_pop_ptr->maxrank = rnk;

    /*
    fprintf(gr," RANK      No Of Individuals
    \n");
    for(i = 0; i < rnk; i++)
        fprintf(gr, "\t%d\t%d\n", i+1, globalpop.
        rankno[i]);

    fclose(gr);
    */
    return;
}
void grankc(gen)

```



```

int i,j,k, rnk, val, nondom, popsize1, gflg[2*
maxpop],q;
float *ptr1,*ptr2;
float *err_ptr1,*err_ptr2;
FILE *gr;

/*gr = fopen("g_rank_record.out","a");
fprintf(gr,"Generation no. = %d\n",gen);
*/
/*-----* RANKING *-
-----*/
rnk = 0;
nondom = 0;
popsize1 = 2*popsize;
min_fit = popsize1;
delta_fit = 0.1 *popsize1;
for(i=0;i<popsize1;i++)
{
gflg[i] = 2;
}
for(k = 0;k < popsize1;k++)
{
q = 0;
for(j = 0;j < popsize1;j++)
{
if (gflg[j] != 1) break;
}
if(j == popsize1) break;
rnk = rnk +1;
for( j = 0 ; j < popsize1; j++)
{
if(gflg[j] == 0) gflg[j] = 2;
}
for(i = 0;i< popsize1 ; i++)
{
if(gflg[i] != 1 && gflg[i] != 0)
{
ptr1 = &(global_pop_ptr->fitness[i][
0]);
err_ptr1 = &(global_pop_ptr->error[i
]);
for(j = 0;j < popsize1 ; j++)
{
if( i!= j)
{
if(gflg[j] != 1)
{
ptr2 = &(global_pop_ptr->fitness[j][
0]);
err_ptr2 = &(global_pop_ptr->error[j
]);

if(*err_ptr1 < 1.0e-6 && *err_ptr2 >
1.0e-6)
{
/* first feasible second
individaul is infeasible*/
gflg[j] = 0;
}
else
{
if(*err_ptr1 >1.0e-6 && *
err_ptr2 < 1.0e-6)
/*first individual is infeasible
and second is feasible*/
gflg[i] = 0;
break;
}
else
/*both feasible or both infeasible
*/
if(*err_ptr1 > *err_ptr2)
{
gflg[i] = 0;
/*first individual is more
infeasible*/
break;
}
else
{
if(*err_ptr1 < *err_ptr2)
gflg[j] = 0;
/*second individual is more
infeasible*/
}
else
{
val = indcmp1(ptr1,ptr2);
if( val == 2)
{
gflg[i] = 0;
/* individual 1 is dominated
*/
break;
}
if(val == 1)
{
gflg[j] = 0;
/* individual 2 is dominated
*/
}
if(val == 3)
{
nondom++;/* individual 1 & 2
are non dominated */
if(gflg[j] != 0) gflg[j] = 3
;
}
}
}
}
}
}
}
}
}
}

if( j == popsize1)
{
global_pop_ptr->rank[i] = rnk;
gflg[i] = 1;
global_pop_ptr->rankar[rnk-1][q] = i;
q++;
}
}
global_pop_ptr->rankno[rnk-1] = q;
}
global_pop_ptr->maxrank = rnk;
return;
}

int indcmp1(float *ptr1,float *ptr2)
{
float fit1[maxfun],fit2[maxfun];
int i,value,m,n;
for(i = 0;i < nfunc ;i++)
{
fit1[i] = *ptr1++;
fit2[i] = *ptr2++;
}
m = 0;n=0;
while(m < nfunc && fit1[m] <= fit2[m])
{

```

```

        if((fit2[m] - fit1[m]) < 1e-7) n++;
        m++;
    }
    if(m == nfunc)
    {
        if(n == nfunc) value = 3;
        else value = 1; /*
            value = 1 for dominating*/
    }
    else
    {
        m = 0; n = 0;
        while(m < nfunc && fit1[m] >= fit2[m])
        {
            if((fit1[m] - fit2[m]) < 1e-7) n++;
            m++;
        }
        if(m == nfunc)
        {
            if(n != nfunc)
                value = 2;
                value = 2 for dominated */ /*
            else value = 3;
        }
        else value = 3;
        value = 3 for incomparable*/ /*
    }
    return value;
}

/* This is the file used to sort the
dummyfitness arrays */
void gsort(int rnk, int sel)
{
    int i, j, a, q;
    float array[2*maxpop][2], temp, temp1;

    q = globalpop.rankno[rnk-1];

    for(i = 0; i < q; i++)
    {
        array[i][0] = globalpop.rankar[rnk-1][i];
        a = globalpop.rankar[rnk-1][i];
        array[i][1] = globalpop.cub_len[a];
    }
    for(i = 0; i < q; i++)
    {
        for(j = i+1; j < q; j++)
        {
            if(array[i][1] < array[j][1])
            {
                temp = array[i][1];
                temp1 = array[i][0];
                array[i][1] = array[j][1];
                array[i][0] = array[j][0];

                array[j][1] = temp;
                array[j][0] = temp1;
            }
        }
    }
    for(i = 0; i < sel; i++)
    {
        a = array[i][0];
        globalpop.flag[a] = 1;
    }
    return;
}

/*=====
=====*/
void gshare(int rnk)
{
    float length[2*maxpop][2], max;
    int i, j, m1, a;
    float min, Diff; // Added 18.08.2003

    m1 = globalpop.rankno[rnk-1];

    for(j = 0; j < nfunc; j++)
    {
        for(i = 0; i < m1; i++)
        {
            fparal[i][0] = 0;
            fparal[i][1] = 0;
        }

        for(i = 0; i < m1; i++)
        {
            a = globalpop.rankar[rnk-1][i];
            fparal[i][0] = (float)a;
            fparal[i][1] = globalpop.fitness[a][j];
        }

        sort(m1); /*Sort the arrays in
ascending order of the fitness*/

        max = fparal[m1-1][1];
        min = fparal[0][1]; // Added
            18.08.2003
        Diff = max-min; // Added
            18.08.2003 and 5 subsequent lines
        if (Diff < 0.0)
        {
            printf("Something wrong in keepaliven.h
\n");
            exit(1);
        }
        for(i = 0; i < m1; i++)
        {
            if(i == 0 || i == (m1-1))
            {
                length[i][0] = fparal[i][0];
                length[i][1] = 100*max;
            }
            else
            {
                length[i][0] = fparal[i][0];
                length[i][1] = fabs(fparal[i+1][1]-
fparal[i-1][1])/Diff; //
crowding distances are
normalized 18.08.2003
            }
        }
        for(i = 0; i < m1; i++)
        {
            a = length[i][0];
            globalpop.cub_len[a] += length[i][1];
        }
    }
    return;
}

void sort(int m1)
{
    float temp, temp1;
    int i1, j1, k1;
    for(k1 = 0; k1 < m1-1; k1++)

```

```

    {
        for(il = k1+1;il < m1;il++)
        {
            if(fparal[k1][1] > fparal[il][1])
            {
                temp = fparal[k1][1];
                temp1 = fparal[k1][0];
                fparal[k1][1] = fparal[il][1];
                fparal[k1][0] = fparal[il][0];
                fparal[il][1] = temp;
                fparal[il][0] = temp1;
            }
        }
    }
    return;
}

```

## cassCross.h

```

/* This is the header file to do the uniform
   crossover */

void cross(population *new_pop_ptr,
           population *mate_pop_ptr)
{
    int i,j,y,n;
    float *gene,*par1,*par2,*chld1,*chld2;
    float rnd;

    for(i = 0,y = 0,n = 0;i < popsize;i++)
    {
        for(j = 0;j < nvar;j++)
        {
            /*Select a bit for doing cross-over*/
            new_pop_ptr->ind_ptr = &(new_pop_ptr->
            ind[y]);
            chld1 = &(new_pop_ptr->ind_ptr->xreal[
            j]);

            new_pop_ptr->ind_ptr = &(new_pop_ptr->
            ind[y+1]);
            chld2 = &(new_pop_ptr->ind_ptr->xreal[
            j]);

            mate_pop_ptr->ind_ptr = &(mate_pop_ptr
            ->ind[n]);
            par1 = &(mate_pop_ptr->ind_ptr->xreal[
            j]);

            mate_pop_ptr->ind_ptr = &(mate_pop_ptr
            ->ind[n+1]);
            par2 = &(mate_pop_ptr->ind_ptr->xreal[
            j]);

            rnd = randomperc();

            /*Checking whether to do cross-over or
            not*/
            if(rnd <= pcross)
            {
                ncross++;
                *chld1 = *par2;
                *chld2 = *par1;
            }
            else
            {
                *chld1 = *par1;
                *chld2 = *par2;
            }
        }
        y = y+2;
        n = n+2;
    }
}

```

```

for(i = 0;i < popsize;i++)
{
    new_pop_ptr->ind_ptr = &(new_pop_ptr->
    ind[i]);
    gene = &(new_pop_ptr->ind_ptr->xreal[0])
    ;
    for(j = 0;j < chrom;j++)
    {
        gene = &(new_pop_ptr->ind_ptr->xreal[j
        ]);
    }
}

return;
}

```

## parameters.h (class scheduling)

```

void transformParameters();
void countTimeslotSize();
void transformTimeslots(char *filename);
void fillTimeslotsTable();
void transformDepartments(char *filename);
void transformDepartmentRooms(char *filename
);
void transformRooms(char *filename);
void transformCourses(char *filename);
void transformInstructors(char *filename);
void transformSubjects(char *filename);
void createFittedSlots(char *filename);
void transformPreferredSchedules(char *
filename);
void InstructorsExpertise(char *filename);
void transformSections(char *filename);

int numSlotTypes = 0;
int faculty_n_db = 0;
int subject_n_db = 0;
int semsubject_n_db = 0;
int course_n_db = 0;
int department_n_db = 0;
int slot_n_db = 0;
int room_n_db = 0;

void transformParameters(){
    transformDepartments("deptFile.in");
    countTimeslotSize();
    if (slot_n==0){
        //printf("No timeslots yet, building
        timeslots..\n");
        fillTimeslotsTable();
    }
    countTimeslotSize();
    transformTimeslots("slotFile.in");
    transformCourses("courseFile.in");
    transformSubjects("subjectFile.in");
    transformSections("sectionFile.in");
    transformInstructors("facultyFile.in");
    InstructorsExpertise("facSubjFile.in");
    transformRooms("roomFile.in");
    transformDepartmentRooms("deptsFile.in");
    transformPreferredSchedules("
    schedulePreference.in");
    createFittedSlots("fitslotsFile.in");
}

void fillTimeslotsTable() {
    int i,days,daysL,daysU;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *endt, *query, *secCode, *secCode2;
    char str1[10], str2[10],strID[10],strHrs[
    10],strDayID[10];
    int qry, id=1, plus=1,minus=1;
    int hours;
}

```

```

mysql_init(&mysql);

if (!mysql_real_connect(&mysql, "localhost",
    "root", NULL, "CASScheduling", 0, NULL, 0)
    ) {
    printf("Error connecting to database: %s\n",
        mysql_error(&mysql));
    } else {
    //printf("Connected...\n");
    }

secCode = (char*)malloc(5000*sizeof(char));
;
secCode2 = (char*)malloc(1000*sizeof(char));
;
query = (char*)malloc(1000*sizeof(char));

for (hours=1; hours<=6; hours++){
    for(i=1;i<=(27-plus);i++){
        if ((hours>=1)&&(hours<=6)){
            if ((hours>=2)&&(hours<=3)){
                daysL = 1;
                daysU = 9;
            }else{
                daysL = 1;
                daysU = 6;
            }
        }
        for (days=daysL;days<=daysU;days++){

            strcpy(query, "SELECT dayCode FROM
                DAYSSPOSS WHERE dayID =");
            sprintf(strDayID, "%d", days);
            strcat(query, strDayID);
            strcat(query, "\0");

            qry = mysql_real_query (&mysql, query, (
                unsigned int) strlen(query));
            if (qry) {
                printf("Error making query: %s\n",
                    mysql_error (&mysql));
            } else {
                //printf("Query %s made...\n", query);
            }

            res = mysql_store_result (&mysql);

            if (res != NULL) {
                row = mysql_fetch_row(res);
                strcpy(secCode, row[0]);
                strcat(secCode, "\0");
            }
            //mysql_free_result (res);

            strcpy(query, "SELECT startTime FROM
                HALFTIME WHERE timeID =");
            sprintf(str1, "%d", (i));
            strcat(query, str1);
            strcat(query, "\0");

            qry = mysql_real_query (&mysql, query, (
                unsigned int) strlen(query));
            if (qry) {
                printf("Error making query: %s\n",
                    mysql_error (&mysql));
            } else {
                //printf("Query %s made...\n", query);
            }

            res = mysql_store_result (&mysql);

            if (res != NULL) {
                row = mysql_fetch_row(res);
                strcat(secCode, row[0]);
                strcat(secCode, "-");
            }

            strcpy(query, "SELECT endTime FROM
                HALFTIME WHERE timeID =");
            sprintf(str2, "%d", (i+plus));
            strcat(query, str2);

            strcat(query, "\0");

            qry = mysql_real_query (&mysql, query, (
                unsigned int) strlen(query));
            if (qry) {
                printf("Error making query: %s\n",
                    mysql_error (&mysql));
            } else {
                //printf("Query %s made...\n", query);
            }

            res = mysql_store_result (&mysql);

            if (res != NULL) {
                row = mysql_fetch_row(res);
                strcat(secCode2, row[0]);

                strcat(secCode2, "\0");
            }

            strcpy(query, "SELECT startTime FROM
                HALFTIME WHERE timeID =");
            sprintf(str1, "%d", (i));
            strcat(query, str1);
            strcat(query, "\0");

```

```

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);

if (res != NULL) {
    row = mysql_fetch_row(res);
    strcat(secCode2,row[0]);
    strcat(secCode2,"-");
}
//mysql_free_result(res);

strcpy(query,"SELECT endTime FROM
    HALFTIME WHERE timeID =");
if (days==10){
    sprintf(str2,"%d", (i+1));
} else{
    sprintf(str2,"%d", (i+2));
}
strcat(query,str2);
strcat(query," \0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);

if (res != NULL) {
    row = mysql_fetch_row(res);
    strcat(secCode2,row[0]);
    strcat(secCode2," \0");
}

sprintf(strID,"%d", (id));
if (days==10){
    sprintf(strHrs,"%d",1);
} else{
    sprintf(strHrs,"%f", (1.5));
}

strcpy(query,"INSERT INTO SECTIONS
    VALUES ");
strcat(query,"(");
strcat(query,strID);strcat(query,",");
strcat(query,secCode2);strcat(query
    ,",");strcat(query,strDayID);
strcat(query,",");strcat(query,str1
    );strcat(query,",");strcat(query,
    str2);strcat(query,",");strcat(
    query,strHrs);
strcat(query,") \0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);
id++;
}
}
}

strcpy(query, "DELETE FROM SECTIONS
    WHERE nhrs in (1,2,4,5)");
strcat(query," \0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

strcpy(query, "DELETE FROM SECTIONS
    WHERE nhrs in (3,6) AND dayID>6");
strcat(query," \0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

strcpy(query, "DELETE FROM SECTIONS
    WHERE dayID=9");
strcat(query," \0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

strcpy(query, "DELETE FROM SECTIONS
    WHERE endTime>20");
strcat(query," \0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

strcpy(query, "DELETE FROM SECTIONS
    WHERE dayID=6");
strcat(query," \0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

strcpy(query, "DELETE FROM SECTIONS
    WHERE dayID in (2,4) AND nhrs in (3
    ,6)");
strcat(query," \0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

```

```

free(query);
free(secCode);
free(secCode2);
}

void countTimeslotSize() {
MYSQL mysql;
MYSQL_RES *res = NULL;
MYSQL_ROW row;
char *query;
int qry;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"localhost"
,"root",NULL,"CASScheduling",0,NULL,0)
) {
printf("Error connecting to database: %s
\n", mysql_error(&mysql));
} else {
//printf("Connected...\n");
}

query = "SELECT COUNT(sectionID) FROM
SECTIONS";

qry = mysql_real_query(&mysql,query,(
unsigned int) strlen(query));
if (qry) {
printf("Error making query: %s\n",
mysql_error(&mysql));
} else {
//printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);
row = mysql_fetch_row(res);
slot_n_db = atoi(row[0]);
slot_n = slot_n_db;
//printf("slot %d\n", slot_n);

mysql_free_result(res);
mysql_close(&mysql);
}

void transformTimeslots(char *filename){
FILE *slot_file;
int i;

MYSQL mysql;
MYSQL_RES *res = NULL;
MYSQL_ROW row;
char *query;
int qry,num,index=-1;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"localhost"
,"root",NULL,"CASScheduling",0,NULL,0)
) {
printf("Error connecting to database: %s
\n", mysql_error(&mysql));
} else {
//printf("Connected...\n");
}

query = "SELECT a.*,b.mtgspwrk FROM
SECTIONS AS a JOIN DAYSPoss AS b USING
(dayID)";
qry = mysql_real_query(&mysql,query,(
unsigned int) strlen(query));
if (qry) {
printf("Error making query: %s\n",
mysql_error(&mysql));
} else {
//printf("Query %s made...\n", query);
}

res = mysql_store_result (&mysql);
if (res != NULL) {
num = mysql_num_rows (res);
slot_n = slot_n_db;

timeslots = (slot*) malloc (slot_n*sizeof
(slot));

for (index=0;index<slot_n;index++) {
row = mysql_fetch_row(res);
timeslots [index].id = atoi(row[0]);

timeslots [index].code = (char*) malloc
(1000*sizeof(char));
strcpy(timeslots [index].code, row[1]);
strcat (timeslots [index].code,"\0");
timeslots [index].day = atoi(row[2]);
timeslots [index].start_time = atoi(row
[3]);
timeslots [index].end_time = atoi(row[4
]);
timeslots [index].hrs = atof(row[5]);
timeslots [index].num_meetings = atoi(
row[6]);
}
}
//mysql_free_result (res);
mysql_close (&mysql);

slot_file = fopen(filename, "w");
fprintf(slot_file, "%d\n", slot_n_db);
fprintf(slot_file, "| sectionID |
sectionCode | dayID | starttime |
endtime | nhrs | mtgspwrk |\n");
for(i=0; i<slot_n; i++){
fprintf(slot_file, "%d\qry%s\qry%d\qry%d
\qry%d\qry%d\qry%f\n", timeslots[i].
id, timeslots [i].code, timeslots [i].
day, timeslots [i].start_time,
timeslots [i].end_time, timeslots [i].

hrs, timeslots [i].num_meetings);
}

fclose(slot_file);
}

void transformDepartments(char *filename){
int i;
FILE *department_file;

MYSQL mysql;
MYSQL_RES *res = NULL;
MYSQL_ROW row;
char *query;

int qry,num;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"localhost"
,"root",NULL,"CASScheduling",0,NULL,0)
) {
printf("Error connecting to database: %s
\n", mysql_error(&mysql));
} else {
//printf("Connected...\n");
}

query = (char*)malloc(1000*sizeof(char));
strcpy(query,"SELECT deptID, deptName FROM
DEPTS");

```

```

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}
fflush(stdout);

res = mysql_store_result(&mysql);
if (res != NULL){
    num = mysql_num_rows(res);
    department_n = num;

    department_list = (department*) malloc(
        department_n*sizeof(department));

    for(i=0; i<department_n; i++){
        row = mysql_fetch_row(res);

        department_list[i].deptID = atoi(row[0]
        );
        department_list[i].deptName = (char*)
            malloc(50*sizeof(int));
        strcpy(department_list[i].deptName,row
            [1]);
        strcat(department_list[i].deptName,"\0
            ");
        }

    free(query);
    mysql_free_result(res);
    mysql_close(&mysql);

    department_file = fopen(filename, "w");
    fprintf(department_file, "%d\n",
        department_n);
    fprintf(department_file, "| deptID |
        deptName |\n");
    for(i=0; i<department_n; i++){
        fprintf(department_file, "%d\qry%s\n",
            department_list[i].deptID,
            department_list[i].deptName);
    }
    fclose(department_file);
}

void transformCourses(char *filename){
    int i;
    FILE *course_file;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;

    int qry,num,j,found=0,dpt_i;

    mysql_init(&mysql);
    if (!mysql_real_connect(&mysql,"localhost"
        ,"root",NULL,"CASScheduling",0,NULL,0)
        ) {
        printf("Error connecting to database: %s
            \n", mysql_error(&mysql));
    } else {
        //printf("Connected...\n");
    }

    query = (char*)malloc(1000*sizeof(char));
    strcpy(query,"SELECT courseID, courseCode,
        deptID FROM COURSES WHERE status=1 ");
    strcat(query," ORDER BY deptID, courseCode
        \0");

    qry = mysql_real_query(&mysql,query,(
        unsigned int) strlen(query));
    if (qry) {
        printf("Error making query: %s\n",
            mysql_error(&mysql));
    } else {
        //printf("Query %s made...\n", query);
    }

    res = mysql_store_result(&mysql);
    if (res != NULL){
        num = mysql_num_rows(res);
        course_n_db = num;
        course_n = course_n_db;
        //printf("courses n: %d\n", course_n);

        course_list = (course*) malloc(num*
            sizeof(course));

        for(i=0; i<num; i++){
            row = mysql_fetch_row(res);

            course_list[i].courseID = atoi(row[0])
                ;

            course_list[i].courseCode = (char*)
                malloc(45*sizeof(int));
            strcpy(course_list[i].courseCode,row[1]
                );
            strcat(course_list[i].courseCode,"\0")
                ;

            course_list[i].dept = atoi(row[2]);

            found=0;
            for (j=0;j<department_n && found==0;j+
                ){
                if (department_list[j].deptID ==
                    atoi(row[2])){
                    dpt_i = j;
                    found=1;
                }
            }
            course_list[i].dept_i = dpt_i;

            fflush(stdout);
        }

        free(query);
        mysql_free_result(res);
        mysql_close(&mysql);

        course_file = fopen(filename, "w");
        fprintf(course_file, "%d\n", course_n);
        fprintf(course_file, "| courseID |
            courseCode | deptID |\n");
        for(i=0; i<course_n; i++){
            fprintf(course_file, "%d\qry%s\qry%d\n",
                course_list[i].courseID, course_list
                [i].courseCode, course_list[i].dept)
                ;
        }
        fclose(course_file);
    }

    void transformSubjects(char *filename){
        int i, j, totalSec = 0;
        int cmtg, smtg;
        int chours, shours;
        FILE *subject_file;
        int totSlotFit = 0;

        MYSQL mysql;
        MYSQL_RES *res = NULL;
        MYSQL_ROW row;
        char *query;
        int qry,num, found, sub, k, cors, dpt_i;

```

```

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"localhost"
    ,"root",NULL,"CASScheduling",0,NULL,0)
    ) {
    printf("Error connecting to database: %s
    \n", mysql_error(&mysql));
} else {
    //printf("Connected...\n");
}

query = (char*)malloc(1000*sizeof(char));

strcpy(query,"SELECT DISTINCT(a.subjectID)
    , b.subjectCode, a.type, a.labtype, b.
    deptID, b.lectUnits, b.labUnits, b.isGE
    FROM SEM SUBJECTS AS a JOIN SUBJECTS
    AS b USING (subjectID) WHERE status=1
    AND a.aysem = ");
strcat(query,AY_SEM);
strcat(query," ORDER BY deptID, subjectID,
    type, labtype\0");

qry = mysql_real_query (&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result (&mysql);
if (res != NULL){
    num = mysql_num_rows (res);
    subject_n_db = num;
    subject_n = subject_n_db;
    printf("subject n: %d\n", subject_n);

    subject_list = (subject*) malloc(
        subject_n*sizeof(subject));

    for(i=0; i<num; i++){
        row = mysql_fetch_row(res);
        subject_list[i].subjID = atoi(row[0]);
        subject_list[i].subjCode = (char*)
            malloc(45*sizeof(int));
        strcpy(subject_list[i].subjCode,row[1]
            );
        strcat(subject_list[i].subjCode,"\0");
        subject_list[i].subjType = atoi(row[2]
            );
        subject_list[i].lectunits = atoi(row[5]
            );
        subject_list[i].labunits = atoi(row[6]
            );

        subject_list[i].subjlabType = atoi(row
            [3]);
        subject_list[i].dept = atoi(row[4]);

        found=0;
        for (j=0;j<department_n && found==0;j+
            ){
            if (department_list[j].deptID ==
                atoi(row[4])){
                dpt_i = j;
                found=1;
            }
        }

        subject_list[i].dept_i = dpt_i;

        subject_list[i].GESubj = atoi(row[7]);
        subject_list[i].nTSec=0;
        subject_list[i].demand=0;
        fflush(stdout);
    }
}

subject_file = fopen(filename, "w");
fprintf(subject_file, "%d\n", subject_n);
fprintf(subject_file, "subjectID |
    subjectCode | Type |\n");
for(i=0; i<num; i++){
    fprintf(subject_file, "%d\qry%s\qry%d\n"
        , subject_list[i].subjID,
        subject_list[i].subjCode,
        subject_list[i].subjType);
}

strcpy(query,"SELECT a.courseID, a.yrLevel
    , a.subjectID, a.type, a.labtype, a.
    sec, a.slots, a.mtgs, a.hrs, b.deptID,
    b.lectUnits, b.labUnits FROM
    SEM SUBJECTS AS a JOIN SUBJECTS AS b
    USING (subjectID) WHERE status=1 AND
    aysem = ");
strcat(query,AY_SEM);
strcat(query," ORDER BY courseID, yrLevel,
    subjectID, type, labtype\0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);
if (res != NULL){
    num = mysql_num_rows(res);
    semsubject_n_db = num;
    semsubject_n = semsubject_n_db;
    printf("semsubject n: %d\n",
        semsubject_n);

    semsubject_list = (sem_subject*) malloc(
        semsubject_n*sizeof(sem_subject));

    for(i=0; i<num; i++){
        row = mysql_fetch_row(res);
        found=0;
        for(j=0;j<course_n && found==0;j++){
            if (course_list[j].courseID == atoi
                (row[0])){
                cors = j;
                found=1;
            }
        }
        semsubject_list[i].cors_ind = cors;
        semsubject_list[i].crseID = atoi(row[0]
            );
        semsubject_list[i].yearLevel = atoi(
            row[1]);
        semsubject_list[i].subjID = atoi(row[2]
            );
        semsubject_list[i].subjType = atoi(row
            [3]);
        semsubject_list[i].subjlabType = atoi(
            row[4]);

        if (semsubject_list[i].subjType == 1){
            semsubject_list[i].units = atoi(row[
                10]);
        }else{
            semsubject_list[i].units = atoi(row[
                11]);
        }

        found=0;
        for(j=0;j<subject_n && found==0;j++){
            if ((subject_list[j].subjID == atoi
                (row[2])) && (subject_list[j].
                subjType == atoi(row[3]))){
                sub = j;
                found=1;
            }
        }
    }
}

```



```

    }
    semsubject_list[i].sub_ind = sub;
    semsubject_list[i].nSec = atoi(row[5])
    ;

    subject_list[sub].nTSec = subject_list
    [sub].nTSec + semsubject_list[i].
    nSec;
    totalSec += semsubject_list[i].nSec;
    semsubject_list[i].nSlots = atoi(row[6]
    );
    subject_list[sub].demand =
    subject_list[sub].demand + (atoi(
    row[5])* atoi(row[6]));
    semsubject_list[i].nMeet = atoi(row[7]
    );
    semsubject_list[i].hours = atof(row[8]
    );
    semsubject_list[i].dept = atof(row[9])
    ;

    found=0;
    for(j=0;j<department_n && found==0;j+
    +){
        if (department_list[j].deptID ==
        atoi(row[9])){
            dpt_i = j;
            found=1;
        }
    }

    semsubject_list[i].dept_i = dpt_i;

    // counts the number of compatible
    timeslots
    cmtg = semsubject_list[i].nMeet;
    chours = semsubject_list[i].hours;

    semsubject_list[i].slotFit = (int*)
    malloc(slot_n_db*sizeof(int));
    totSlotFit = 0;
    for(j=0; j<slot_n_db; j++){
        smtg = timeslots[j].num meetings;
        shours = timeslots[j].hrs * smtg;
        if((smtg!=cmtg) || (shours!=chours))
            semsubject_list[i].slotFit[j] = 0;
            //subject i does not fit
            timeslot j
        else {
            semsubject_list[i].slotFit[j] = 1;
            //subject i fits timeslot j
            totSlotFit++;
        }
    }
    semsubject_list[i].numFitSlots =
    totSlotFit;

    fflush(stdout);
}

}

free(query);
mysql_free_result(res);
mysql_close(&mysql);

section_n = totalSec;
nvar = section_n*3;

fprintf(subject_file, "\n\n%d\n",
    semsubject_n);
fprintf(subject_file, "| courseID |
    yrLevel | subjectID | type | labtype |
    sec | slots | mtgs | hrs | subjInd |
    corsInd\n");
for(i=0; i<semsubject_n; i++){
    fprintf(subject_file, "%d\qry%d\qry%d\q
    ry%d\qry%d\qry%d\qry%d\qry%f\q
    ry%d\qry%d\n", semsubject_list[i].
    crseID, semsubject_list[i].yearLevel
    , semsubject_list[i].subjID,
    semsubject_list[i].subjType,
    semsubject_list[i].subjlabType,
    semsubject_list[i].nSec,
    semsubject_list[i].nSlots,
    semsubject_list[i].nMeet,
    semsubject_list[i].hours,
    semsubject_list[i].sub_ind,
    semsubject_list[i].cors_ind);
}
fclose(subject_file);
}

void transformSections(char *filename){
    FILE *section_file;
    int i, j, k, sec_c=0, found=0, cors, sub;

    section_list = (section*) malloc(section_n
    *sizeof(section));

    for(i=0; i<semsubject_n; i++){
        for(j=1; j<=semsubject_list[i].nSec; j++
        ){
            section_list[sec_c].section_i = j;
            section_list[sec_c].semsubject_i = i;
            section_list[sec_c].subject_i =
            semsubject_list[i].sub_ind;
            section_list[sec_c].course_i =
            semsubject_list[i].cors_ind;
            section_list[sec_c].yearLevel =
            semsubject_list[i].yearLevel;

            section_list[sec_c].type =
            semsubject_list[i].subjType;
            section_list[sec_c].labtype =
            semsubject_list[i].subjlabType;

            sec_c++;
        }
    }

    section_file = fopen(filename, "w");
    fprintf(section_file, "%d\n", section_n);
    fprintf(section_file, "subject_i/course_i/
    sec_i/type/labtype\n");
    for(i=0; i<section_n; i++){
        fprintf(section_file, "%d %d %d %d %d\n"
        , section_list[i].subject_i,
        section_list[i].course_i,
        section_list[i].section_i,
        section_list[i].type, section_list[i
        ].labtype);
    }
    fclose(section_file);
}

void transformInstructors(char *filename){
    FILE *fac_file;
    int i, j, found=0, dpt_i;
    char *name;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;
    int qry,num;

```

```

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"localhost"
,"root",NULL,"CASScheduling",0,NULL,0)
){
printf("Error connecting to database: %s
\n", mysql_error(&mysql));
} else {
//printf("Connected...\n");
}

query = (char*)malloc(1000*sizeof(char));
strcpy(query,"SELECT a.facultyID,concat(b.
lname,\" \",\" \",b.fname),b.deptID FROM
SEM FACULTY AS a JOIN FACULTY AS b
USING(facultyID) WHERE aysem=");
strcat(query,AY_SEM);
strcat(query," ORDER BY deptID,facultyID\0
");

qry = mysql_real_query(&mysql,query,(
unsigned int) strlen(query));
if (qry) {
printf("Error making query: %s\n",
mysql_error(&mysql));
} else {
//printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);
if (res != NULL) {
num = mysql_num_rows(res);
faculty_n_db = num;
faculty_n = faculty_n_db;

faculty_list = (faculty*) malloc(num*
sizeof(faculty));

for(i=0; i<num; i++) {
row = mysql_fetch_row(res);

faculty_list[i].facID = atoi(row[0]);

name = (char*)malloc(150*sizeof(char)
);
if(row[1] != NULL) {
strcpy(name,row[1]);
strcat(name,"\0");
} else {
strcpy(name,"\0");
}

fflush(stdout);
faculty_list[i].facName = name;
faculty_list[i].dept = atoi(row[2]);

found=0;
for (j=0;j<department_n && found==0;j+
+){
if (department_list[j].deptID ==
atoi(row[2])){
dpt_i = j;
found=1;
}
}

faculty_list[i].dept_i = dpt_i;

faculty_list[i].load = 0;

}
free(name);

}

free(query);

mysql_free_result(res);
mysql_close(&mysql);

fac_file = fopen(filename, "w");
fprintf(fac_file, "%d\n", faculty_n);
fprintf(fac_file, "| facultyID |
facultyName | deptID | load |\n");
for(i=0; i<faculty_n; i++){
fprintf(fac_file, "%d\qry%s\qry%d\qry\n"
, faculty_list[i].facID,
faculty_list[i].facName,
faculty_list[i].dept);
}

fclose(fac_file);
}

void transformRooms(char *filename){
FILE *room_file;
int i,j,found=0,dpt_i;

MYSQL mysql;
MYSQL_RES *res = NULL;
MYSQL_ROW row;
char *query;
int qry,num;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"localhost"
,"root",NULL,"CASScheduling",0,NULL,0)
){
printf("Error connecting to database: %s
\n", mysql_error(&mysql));
} else {
//printf("Connected...\n");
}

query = (char*)malloc(1000*sizeof(char));
strcpy(query,"SELECT a.roomID,b.bldg,b.
roomName,b.type,b.labtype,b.cap,b.
isShared,b.deptID FROM SEM ROOMS AS a
JOIN ROOMS as b USING(roomID) WHERE
status=1 AND aysem = ");
strcat(query,AY_SEM);
strcat(query," ORDER BY bldg,roomID,
roomName,type,cap\0");

qry = mysql_real_query(&mysql,query,(
unsigned int) strlen(query));
if (qry) {
printf("Error making query: %s\n",
mysql_error(&mysql));
} else {
//printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);
if (res != NULL) {
num = mysql_num_rows(res);
room_n_db = num;
room_n = room_n_db;
room_list = (Room*) malloc(num*sizeof(
Room));

for(i=0; i<num; i++){
row = mysql_fetch_row(res);
}
}
}

```

```

room_list[i].roomID = atoi(row[0]);
room_list[i].bldg = (char*) malloc(45*
    sizeof(char));
strcpy(room_list[i].bldg, row[1]);
strcat(room_list[i].bldg, "\0");

room_list[i].roomName = (char*) malloc
    (45*sizeof(char));
strcpy(room_list[i].roomName, row[2]);
strcat(room_list[i].roomName, "\0");

room_list[i].type = atoi(row[3]);
room_list[i].labtype = atoi(row[4]);
room_list[i].capacity = atoi(row[5]);
room_list[i].isShared = atoi(row[6]);
room_list[i].dept = atoi(row[7]);//
    dval;

found=0;
for (j=0;j<department_n && found==0;j+
    +){
    if (department_list[j].deptID ==
        atoi(row[7])){
        dpt_i = j;
        found=1;
    }
}

room_list[i].dept_i = dpt_i;
}
}

free(query);
mysql_free_result(res);
mysql_close(&mysql);

room_file = fopen(filename, "w");
fprintf(room_file, "%d\n", room_n);
fprintf(room_file, "| bldg | roomName |
    type | labtype | cap | isShared |
    deptID | roomID |\n");
for(i=0; i<num; i++) {
    fprintf(room_file, "%s\qry%s\qry%d\qry%d
        \qry%d\qry%d\qry%d\qry%d\n",
        room_list[i].bldg, room_list[i].
        roomName, room_list[i].type,
        room_list[i].labtype, room_list[i].
        capacity, room_list[i].isShared,
        room_list[i].dept, room_list[i].
        roomID);
}

fclose(room_file);
}

void InstructorsExpertise(char* filename){
    int i, j, num, subj_i, fac_i, subj, *curr,
        *FacN, found=0, unit;
    FILE* subjfac_ptr;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;

    mysql_init(&mysql);

    if (!mysql_real_connect(&mysql, "localhost"
        , "root", NULL, "CASScheduling", 0, NULL, 0)
        ) {
        printf("Error connecting to database: %s
            \n", mysql_error(&mysql));
    } else {
        //printf("Connected...\n");
    }

    query = (char*) malloc (1000*sizeof(char))
        ;
    strcpy(query, "SELECT facultyID, subjectID,
        type FROM FACULTY_SUBJECTS WHERE aysem
        ="); strcat(query, AY_SEM);
    strcat(query, " ORDER BY subjectID,
        facultyID\0");

    qry = mysql_real_query(&mysql, query, (
        unsigned int) strlen(query));
    if (qry) {
        printf("Error making query: %s\n",
            mysql_error(&mysql));
    } else {
        //printf("Query %s made...\n", query);
    }

    res = mysql_store_result(&mysql);
    if (res != NULL) {
        num = mysql_num_rows(res);
        subjfac_n = num;
        subjfac_list = (subjectFaculty*) malloc(
            subjfac_n * sizeof(subjectFaculty));
        FacN = (int*) malloc(subject_n*sizeof(
            int));
        curr = (int*) malloc(subject_n*sizeof(
            int));

        for(i=0; i<subjfac_n; i++){
            row = mysql_fetch_row(res);

            for(j=0;j<faculty_n && found==0;j++){
                if (faculty_list[j].facID == atoi(
                    row[0])){
                    fac_i = j;
                    found=1;
                }
            }
            found=0;
            for(j=0;j<subject_n && found==0;j++){
                if ((subject_list[j].subjID == atoi
                    (row[1])) && (subject_list[j].
                    subjType == atoi(row[2]))){
                    subj_i = j;
                }
            }

            subjfac_list[i].subject_i = subj_i;
            subjfac_list[i].faculty_i = fac_i;
            subjfac_list[i].type = atoi(row[2]);
        }

        mysql_free_result(res);
        mysql_close(&mysql);
        free(query);

        for(i=0; i<subject_n; i++){
            FacN[i] = 0;
            curr[i] = 0;
        }
    }
}

```

```

for(j=0; j<subjfac_n; j++){
    subj_i = subjfac_list[j].subject_i;
    FacN[subj_i]++;
}

for(i=0; i<subject_n; i++){
    subject_list[i].numOfFitInsts = FacN[i];
    subject_list[i].fac = (int*) malloc(FacN
    [i]*sizeof(int));
}

for(j=0; j<subjfac_n; j++){
    subj = subjfac_list[j].subject_i;
    fac_i = subjfac_list[j].faculty_i;

    if (subjfac_list[j].type==1) unit =
        subject_list[subj].lecunits;
    else unit=subject_list[subj].labunits;

    subject_list[subj].fac[curr[subj]] =
        fac_i;
    faculty_list[fac_i].load = faculty_list[
    fac_i].load + unit;
    curr[subj]++;
}

subjfac_ptr = fopen(filename, "w");
fprintf(subjfac_ptr, "%d\n", subjfac_n);
fprintf(subjfac_ptr, "| subject_i | fac_i
|\n");
for(i=0; i<subject_n; i++){
    fprintf(subjfac_ptr, "%d\qry%d\qry", i,
        subject_list[i].numOfFitInsts);
    for(j=0; j<subject_list[i].numOfFitInsts
        ; j++)
        fprintf(subjfac_ptr, "%d ",
            subject_list[i].fac[j]);
    fprintf(subjfac_ptr, "\n");
}
fclose(subjfac_ptr);
free(FacN);
free(curr);
}

void transformPreferredSchedules (char *
    filename) {
    int i, j;
    FILE *file;

    MYSQL mysql;
    MYSQL_RES *res = NULL,*res2 = NULL;
    MYSQL_ROW row, row2;
    char *query, *query2, *facID, *dayp, *daym
    ;
    int qry, r, num, k, T_ind;

    facID = (char*)malloc(5*sizeof(char));
    dayp = (char*)malloc(5*sizeof(char));
    daym = (char*)malloc(5*sizeof(char));
    mysql_init (&mysql);
    if (!mysql_real_connect (&mysql, "localhost"
        , "root", NULL, "CASScheduling", 0, NULL, 0)
        ) {
        printf("Error connecting to database: %s
        \n", mysql_error (&mysql));
    } else {
        //printf("Faculty Connected...\n");

```

```

    qry = mysql_real_query(&mysql, query2
        , (unsigned int) strlen(query2));
    if (qry) {
        printf("Error making query: %s\n"
            , mysql_error(&mysql));
    } else {
        //printf("Query %s made...\n",
            query2);
    }
    fflush(stdout);
    res2 = mysql_store_result(&mysql);
    if (res2 != NULL) {
        num = mysql_num_rows(res2);
        for (j=0; j<num; j++) {
            row2 = mysql_fetch_row(res2);
            for (k=0; k<slot_n; k++) {
                if (timeslots[k].id == atoi(
                    row2[0]))
                    T_ind = k;
            }
            faculty_list[i].unavTime[T_ind]
                = 1;
        }
        num = mysql_num_rows(res);
    }
}

free(query);
free(query2);
free(facID);
free(dayp);
free(daym);
mysql_free_result(res2);
mysql_free_result(res);
mysql_close(&mysql);

file = fopen(filename, "w");
for(i=0; i<faculty_n_db; i++){
    fprintf(file, "%d ->", faculty_list[i].
        facID);
    for(j=0; j<slot_n_db; j++) {
        fprintf(file, "%d ", faculty_list[i].
            unavTime[j]);
    }
    fprintf(file, "\n");
}
fclose(file);

void transformDepartmentRooms (char *filename)
{
    int i, j, *lecN, *labN, dept, type, *
        currLec, *currLab;
    FILE* dept_ptr;

    lecN = (int*) malloc(department_n*sizeof(
        int));
    labN = (int*) malloc(department_n*sizeof(
        int));
    currLec = (int*) malloc(department_n*
        sizeof(int));
    currLab = (int*) malloc(department_n*
        sizeof(int));

    for(i=0; i<department_n; i++){
        lecN[i] = 0;
        labN[i] = 0;
        currLec[i] = 0;
        currLab[i] = 0;
    }

    for(j=0; j<room_n_db; j++){
        dept = room_list[j].dept_i;
        type = room_list[j].type;
        if(type == 1){
            lecN[dept]++;
        } else if(type == 2){
            labN[dept]++;
        } else if(type == 3){
            lecN[dept]++;
            labN[dept]++;
        }
        else{}
    }

    for(i=0; i<department_n; i++){
        department_list[i].numLecRooms = lecN[i]
            ;
        department_list[i].lecRooms = (int*)
            malloc(lecN[i]*sizeof(int));
        department_list[i].numLabRooms = labN[i]
            ;
        department_list[i].labRooms = (int*)
            malloc(labN[i]*sizeof(int));
    }

    for(j=0; j<room_n_db; j++){
        dept = room_list[j].dept_i;
        type = room_list[j].type;
        if(type == 1){
            department_list[dept].lecRooms[
                currLec[dept]] = j;
            currLec[dept]++;
        } else if(type == 2){
            department_list[dept].labRooms[
                currLab[dept]] = j;
            currLab[dept]++;
        } else if(type == 3){
            department_list[dept].lecRooms[
                currLec[dept]] = j;
            currLec[dept]++;
            department_list[dept].labRooms[
                currLab[dept]] = j;
            currLab[dept]++;
        }
        else{}
    }

    free(lecN);
    free(labN);
    free(currLec);
    free(currLab);

    dept_ptr = fopen(filename, "w");
    fprintf(dept_ptr, "%d\n", department_n);
    fprintf(dept_ptr, "name/lecN/labN/lecRooms
        /labRooms\n");
    for(i=0; i<department_n; i++){
        fprintf(dept_ptr, "%s\qry%d\qry%d/",
            department_list[i].deptName,
            department_list[i].numLecRooms,
            department_list[i].numLabRooms);
        for(j=0; j<department_list[i].
            numLecRooms; j++){
            fprintf(dept_ptr, "%d ",
                department_list[i].lecRooms[j]);
        }
        fprintf(dept_ptr, "/");
        for(j=0; j<department_list[i].
            numLabRooms; j++){
            fprintf(dept_ptr, "%d ",
                department_list[i].labRooms[j]);
        }
        fprintf(dept_ptr, "\n");
    }
}

```

```

fclose(dept_ptr);
}

void createFittedSlots(char *filename){
    int i, j, a, b, slotA, slotB, fitSlots;
    FILE *slotFitFile;

    for(i=0; i<semsubject_n_db; i++){
        semsubject_list[i].indexFitSlots = (int*)
            malloc(slot_n_db*sizeof(int));

        for(a=0; a<slot_n_db; a++){
            semsubject_list[i].indexFitSlots[a] =
                a;
        }

        for(a=0; a<slot_n_db; a++){
            for(b=0; b<slot_n_db-a-1; b++){
                slotA = semsubject_list[i].
                    indexFitSlots[b];
                slotB = semsubject_list[i].
                    indexFitSlots[b+1];
                if(semsubject_list[i].slotFit[slotA]
                    < semsubject_list[i].slotFit[
                        slotB]){
                    semsubject_list[i].indexFitSlots[b
                        ] = slotB;
                    semsubject_list[i].indexFitSlots[b
                        +1] = slotA;
                }
            }
        }

        slotFitFile = fopen("slotFitFile.in", "w")
            ;
        fprintf(slotFitFile, "subjectID/mtgs/hours
            /numFit/fitSlots\n");
        for(i=0; i<semsubject_n_db; i++){
            fprintf(slotFitFile, "%d\qry%d\qry%f\qry
                %d\qry", i, semsubject_list[i].nMeet
                , semsubject_list[i].hours,
                semsubject_list[i].numFitSlots);
            fitSlots = semsubject_list[i].
                numFitSlots;
            for(j=0; j<fitSlots; j++){
                fprintf(slotFitFile, "%d ",
                    semsubject_list[i].indexFitSlots[j
                    ]);
            }
            fprintf(slotFitFile, "\n");
        }
        fclose(slotFitFile);
    }
}

```

## parameters.h (final exam scheduling)

```

void transformParameters();
void countTimeslotSize();
void transformTimeslots(char *filename);
void fillTimeslotsTable();
void transformDepartments(char *filename);
void transformDepartmentRooms(char *filename
    );
void transformRooms(char *filename);
void transformCourses(char *filename);
void transformSubjects(char *filename);
void createFittedSlots(char *filename);
void transformSections(char *filename);

int numSlotTypes = 0;
int faculty_n_db = 0;
int subject_n_db = 0;
int subject_nn_db = 0;
int semsubject_n_db = 0;
int course_n_db = 0;
int department_n_db = 0;
int slot_n_db = 0;
int room_n_db = 0;

```

```

void transformParameters(){
    transformDepartments("deptFile.in");
    countTimeslotSize();
    if (slot_n==0){
        fillTimeslotsTable();
        countTimeslotSize();
    }
    transformTimeslots("slotFile.in");
    transformCourses("courseFile.in");
    transformSubjects("subjectFile.in");
    transformSections("sectionFile.in");
    transformRooms("roomFile.in");
    transformDepartmentRooms("deptsFile.in");
    createFittedSlots("slotFitFile.in");
}

void fillTimeslotsTable() {
    int i, days;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query, *secCode, *secCode2;
    char str1[10], str2[10], strID[10], strHrs[
        10], strDayID[10];
    int qry, id=1, plus=3;
    int hours;

    mysql_init(&mysql);

    if (!mysql_real_connect(&mysql, "localhost"
        , "root", NULL, "CASScheduling", 0, NULL, 0)
        ) {
        printf("Error connecting to database: %s
            \n", mysql_error(&mysql));
    } else {
        //printf("Connected...\n");
    }

    secCode = (char*)malloc(5000*sizeof(char
        ));
    secCode2 = (char*)malloc(1000*sizeof(
        char));
    query = (char*)malloc(1000*sizeof(char))
        ;

    for (hours=2; hours<=3; hours++){
        for(i=1; i<=(20-plus); i++){
            for (days=1; days<=6; days++){
                strcpy(query, "SELECT dayCode FROM
                    DAYSPOSS WHERE dayID =");
                sprintf(strDayID, "%d", days);
                strcat(query, strDayID);
                strcat(query, "\0");

                qry = mysql_real_query(&mysql, query, (
                    unsigned int) strlen(query));
                if (qry) {
                    printf("Error making query: %s\n",
                        mysql_error(&mysql));
                } else {
                    //printf("Query %s made...\n", query);
                }

                res = mysql_store_result(&mysql);

                if (res != NULL) {
                    row = mysql_fetch_row(res);
                    strcpy(secCode, row[0]);
                    strcat(secCode, "\0");
                }
                //mysql_free_result(res);
            }
        }
    }
}

```

```

strcpy(query, "SELECT startTime FROM
    HALFTIME WHERE timeID =");
sprintf(str1, "%d", (i));
strcat(query, str1);
strcat(query, "\0");

qry = mysql_real_query(&mysql, query, (
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);

if (res != NULL) {
    row = mysql_fetch_row(res);
    strcat(secCode, row[0]);
    strcat(secCode, "-");
}

strcpy(query, "SELECT endTime FROM
    HALFTIME WHERE timeID =");
sprintf(str2, "%d", (i+plus));
strcat(query, str2);
strcat(query, "\0");

qry = mysql_real_query(&mysql, query, (
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);

if (res != NULL) {
    row = mysql_fetch_row(res);
    strcat(secCode, row[0]);
}

sprintf(strID, "%d", (id));
sprintf(strHrs, "%d", (hours));

strcpy(query, "INSERT INTO EXAM_TIME
    VALUES ");
strcat(query, "(");
strcat(query, strID); strcat(query, ", ");
    strcat(query, secCode); strcat(query,
    ", "); strcat(query, strDayID); strcat
    (query, ", "); strcat(query, str1);
    strcat(query, ", "); strcat(query, str2
    ); strcat(query, ", "); strcat(query,
    strHrs);
strcat(query, ")\0");

qry = mysql_real_query(&mysql, query, (
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);
id++;
}
}
plus = plus + 2;
}
free(query);
free(secCode);
free(secCode2);
}

void countTimeslotSize() {
    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;
    int qry;

    mysql_init(&mysql);
    if (!mysql_real_connect(&mysql, "localhost"
        , "root", NULL, "CASScheduling", 0, NULL, 0)
        ) {
        printf("Error connecting to database: %s
            \n", mysql_error(&mysql));
    } else {
        //printf("Connected...\n");
    }

    query = "SELECT COUNT(examTID) FROM
        EXAM_TIME";

    qry = mysql_real_query(&mysql, query, (
        unsigned int) strlen(query));
    if (qry) {
        printf("Error making query: %s\n",
            mysql_error(&mysql));
    } else {
        //printf("Query %s made...\n", query);
    }

    res = mysql_store_result(&mysql);
    row = mysql_fetch_row(res);
    slot_n_db = atoi(row[0]);
    slot_n = slot_n_db;

    mysql_free_result(res);
    mysql_close(&mysql);
}

void transformTimeslots(char *filename){
    FILE *slot_file;
    int i;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;
    int qry, num, index=-1;

    mysql_init(&mysql);
    if (!mysql_real_connect(&mysql, "localhost"
        , "root", NULL, "CASScheduling", 0, NULL, 0)
        ) {
        printf("Error connecting to database: %s
            \n", mysql_error(&mysql));
    } else {
        //printf("Connected...\n");
    }
    query = "SELECT * FROM EXAM_TIME";
    qry = mysql_real_query(&mysql, query, (
        unsigned int) strlen(query));
    if (qry) {
        printf("Error making query: %s\n",
            mysql_error(&mysql));
    } else {
        //printf("Query %s made...\n", query);
    }

    res = mysql_store_result(&mysql);
    if (res != NULL) {
        num = mysql_num_rows(res);
        slot_n = slot_n_db;

        timeslots = (slot*) malloc(slot_n*sizeof
            (slot));

        for (index=0; index<slot_n; index++) {
            row = mysql_fetch_row(res);
            timeslots[index].id = atoi(row[0]);
        }
    }
}

```

```

        timeslots[index].code = (char*) malloc
            (1000*sizeof(char));
        strcpy(timeslots[index].code, row[1]);
        strcat(timeslots[index].code, "\0");
        timeslots[index].day = atoi(row[2]);
        timeslots[index].start_time = atoi(row
            [3]);
        timeslots[index].end_time = atoi(row[4
            ]);
        timeslots[index].hrs = atof(row[5]);
    }
    //mysql_free_result(res);
    mysql_close(&mysql);

    slot_file = fopen(filename, "w");
    fprintf(slot_file, "%d\n", slot_n_db);

    fprintf(slot_file, "| sectionID |
        sectionCode | dayID | starttime |
        endtime | nhrs |\n");
    for(i=0; i<slot_n; i++){
        fprintf(slot_file, "%d\qry%s\qry%d\qry%d
            \qry%d\qry%f\n", timeslots[i].id,
            timeslots[i].code, timeslots[i].day,
            timeslots[i].start_time, timeslots[i
            ].end_time, timeslots[i].hrs);
    }

    fclose(slot_file);
}

void transformDepartments(char *filename){
    int i;
    FILE *department_file;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;

    int qry,num;

    mysql_init(&mysql);
    if (!mysql_real_connect(&mysql,"localhost"
        ,"root",NULL,"CASScheduling",0,NULL,0)
        ) {
        printf("Error connecting to database: %s
            \n", mysql_error(&mysql));
    } else {
        //printf("Connected...\n");
    }

    query = (char*)malloc(1000*sizeof(char));
    strcpy(query,"SELECT deptID, deptName FROM
        DEPTS");

    qry = mysql_real_query(&mysql,query,(
        unsigned int) strlen(query));
    if (qry) {
        printf("Error making query: %s\n",
            mysql_error(&mysql));
    } else {
        //printf("Query %s made...\n", query);
    }
    fflush(stdout);

    res = mysql_store_result(&mysql);
    if (res != NULL){
        num = mysql_num_rows(res);
        department_n = num;

        department_list = (department*) malloc(
            department_n*sizeof(department));

        for(i=0; i<department_n; i++){
            row = mysql_fetch_row(res);

            department_list[i].deptID = atoi(row[0
                ]);
            department_list[i].deptName = (char*)
                malloc(50*sizeof(int));
            strcpy(department_list[i].deptName,row
                [1]);
            strcat(department_list[i].deptName,"\0
                ");
        }

        free(query);
        mysql_free_result(res);
        mysql_close(&mysql);

        department_file = fopen(filename, "w");
        fprintf(department_file, "%d\n",
            department_n);
        fprintf(department_file, "| deptID |
            deptName |\n");
        for(i=0; i<department_n; i++){
            fprintf(department_file, "%d\qry%s\n",
                department_list[i].deptID,
                department_list[i].deptName);
        }
        fclose(department_file);
    }
}

void transformCourses(char *filename){
    int i;
    FILE *course_file;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;

    int qry,num,j,found=0,dpt_i;

    mysql_init(&mysql);
    if (!mysql_real_connect(&mysql,"localhost"
        ,"root",NULL,"CASScheduling",0,NULL,0)
        ) {
        printf("Error connecting to database: %s
            \n", mysql_error(&mysql));
    } else {
        //printf("Connected...\n");
    }

    query = (char*)malloc(1000*sizeof(char));
    strcpy(query,"SELECT courseID, courseCode,
        deptID FROM COURSES WHERE status=1 ");
    strcat(query," ORDER BY deptID, courseCode
        \0");

    qry = mysql_real_query(&mysql,query,(
        unsigned int) strlen(query));
    if (qry) {
        printf("Error making query: %s\n",
            mysql_error(&mysql));
    } else {
        //printf("Query %s made...\n", query);
    }

    res = mysql_store_result(&mysql);
    if (res != NULL){
        num = mysql_num_rows(res);
        course_n_db = num;
        course_n = course_n_db;
    }
}

```



```

course_list = (course*) malloc(num*
    sizeof(course));
for(i=0; i<num; i++){
    row = mysql_fetch_row(res);

    course_list[i].courseID = atoi(row[0])
    ;

    course_list[i].courseCode = (char*)
        malloc(45*sizeof(int));
    strcpy(course_list[i].courseCode,row[1]
    );
    strcat(course_list[i].courseCode,"\0")
    ;

    course_list[i].dept = atoi(row[2]);

    found=0;
    for (j=0;j<department_n && found==0;j+
    ){
        if (department_list[j].deptID ==
            atoi(row[2])){
            dpt_i = j;
            found=1;
        }
    }

    course_list[i].dept_i = dpt_i;

    fflush(stdout);
}
}

free(query);
mysql_free_result(res);
mysql_close(&mysql);

course_file = fopen(filename, "w");
fprintf(course_file, "%d\n", course_n);
fprintf(course_file, "| courseID |
    courseCode | deptID |\n");
for(i=0; i<course_n; i++){
    fprintf(course_file, "%d\qry%s\qry%d\n",
        course_list[i].courseID, course_list
        [i].courseCode, course_list[i].dept
        );
}
fclose(course_file);

void transformSubjects(char *filename){
    int i, j, totalSec = 0;
    int chours, shours;
    FILE *subject_file;
    int totSlotFit = 0;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;
    int qry,num, found, sub, cors, dpt_i;

    mysql_init(&mysql);
    if (!mysql_real_connect(&mysql,"localhost"
        ,"root",NULL,"CASScheduling",0,NULL,0)
        ){
        printf("Error connecting to database: %s
            \n", mysql_error(&mysql));
    } else {
        //printf("Connected...\n");
    }

    query = (char*)malloc(1000*sizeof(char));

    strcpy(query,"SELECT DISTINCT(a.subjectID)
        , b.subjectCode, a.type, a.labtype, b.
        deptID FROM SEM SUBJECTS AS a JOIN
        SUBJECTS AS b USING (subjectID) WHERE
        status=1 AND a.aysem = ");
    strcat(query,AY_SEM);
    strcat(query," ORDER BY subjectID, type,
        labtype, deptID\0");

    qry = mysql_real_query (&mysql,query,(
        unsigned int) strlen(query));
    if (qry) {
        printf("Error making query: %s\n",
            mysql_error (&mysql));
    } else {
        //printf("Query %s made...\n", query);
    }

    res = mysql_store_result (&mysql);
    if (res != NULL){
        num = mysql_num_rows (res);
        subject_n_db = num;
        subject_n = subject_n_db;
        printf("subject n: %d\n", subject_n);

        subject_list = (subject*) malloc(
            subject_n*sizeof(subject));

        for(i=0; i<num; i++){
            row = mysql_fetch_row (res);
            subject_list[i].subjID = atoi(row[0]);
            subject_list[i].subjCode = (char*)
                malloc(45*sizeof(int));
            strcpy(subject_list[i].subjCode,row[1]
            );
            strcat(subject_list[i].subjCode,"\0");
            subject_list[i].subjType = atoi(row[2]
            );
            subject_list[i].subjlabType = atoi(row
                [3]);
            subject_list[i].dept = atoi(row[4]);

            found=0;
            for (j=0;j<department_n && found==0;j+
            ){
                if (department_list[j].deptID ==
                    atoi(row[4])){
                    dpt_i = j;
                    found=1;
                }
            }

            subject_list[i].dept_i = dpt_i;

            subject_list[i].nTSec=0;
            subject_list[i].demand=0;
            fflush(stdout);
        }
    }

    subject_file = fopen(filename, "w");
    fprintf(subject_file, "%d\n", subject_n);
    fprintf(subject_file, "subjectID |
        subjectCode | Type |\n");
    for(i=0; i<num; i++){
        fprintf(subject_file, "%d\qry%s\qry%d\n"
            , subject_list[i].subjID,
            subject_list[i].subjCode,
            subject_list[i].subjType);
    }

    strcpy(query,"SELECT DISTINCT (subjectID)
        FROM SEM_SUBJECTS WHERE aysem = ");

```

```

strcat(query,AY_SEM);
strcat(query,"0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    // printf("Query %s made...\n", query
    );
}

res = mysql_store_result(&mysql);
if (res != NULL){
    num = mysql_num_rows(res);
    subject_nn_db = num;
    subject_nn = subject_nn_db;

    subjects_list = (subject*) malloc(
        subject_nn*sizeof(subjects));

    for(i=0; i<num; i++){
        row = mysql_fetch_row(res);
        subjects_list[i].subjID = atoi(row[0])
        ;
    }
}

strcpy(query,"SELECT a.courseID, a.yrLevel
, a.subjectID, a.type, a.labtype, a.
sec, a.slots, a.xhrs, b.deptID FROM
SEM SUBJECTS AS a JOIN SUBJECTS AS b
USING (subjectID) WHERE status=1 AND
aysem = ");
strcat(query,AY_SEM);
strcat(query," ORDER BY subjectID, type,
courseID, yrLevel\0");

qry = mysql_real_query(&mysql,query,(
    unsigned int) strlen(query));
if (qry) {
    printf("Error making query: %s\n",
        mysql_error(&mysql));
} else {
    //printf("Query %s made...\n", query);
}

res = mysql_store_result(&mysql);
if (res != NULL){
    num = mysql_num_rows(res);
    semsubject_n_db = num;
    semsubject_n = semsubject_n_db;
    printf("semsubject n: %d\n",
        semsubject_n);

    semsubject_list = (sem_subject*) malloc(
        semsubject_n*sizeof(sem_subject));

    for(i=0; i<num; i++){
        row = mysql_fetch_row(res);
        found=0;
        for(j=0;j<course_n && found==0;j++){
            if (course_list[j].courseID == atoi
                (row[0])){
                cors = j;
                found=1;
            }
        }
        semsubject_list[i].cors_ind = cors;
        semsubject_list[i].crseID = atoi(row[0]
            );
        semsubject_list[i].yearLevel = atoi(
            row[1]);
        semsubject_list[i].subjID = atoi(row[2]
            );
        semsubject_list[i].subjType = atoi(row
            [3]);
        semsubject_list[i].subjlabType = atoi(
            row[4]);

        found=0;
        for(j=0;j<subject_n && found==0;j++){
            if ((subject_list[j].subjID == atoi
                (row[2])) && (subject_list[j].
                subjType == atoi(row[3]))){
                sub = j;
                found=1;
            }
        }
        semsubject_list[i].sub_ind = sub;

        for(j=0;j<subject_nn && found==0;j++){
            if (subjects_list[j].subjID == atoi
                (row[2]))
                semsubject_list[i].subject_ii
                    = j;
        }

        semsubject_list[i].nSec = atoi(row[5])
            ;
        subject_list[sub].nTSec = subject_list
            [sub].nTSec + semsubject_list[i].
            nSec;
        totalSec += semsubject_list[i].nSec;
        semsubject_list[i].nSlots = atoi(row[6]
            );
        subject_list[sub].demand =
            subject_list[sub].demand + (atoi(
                row[5]) * atoi(row[6]));

        semsubject_list[i].examinhours = atof(
            row[7]);

        semsubject_list[i].dept = atof(row[8])
            ;

        found=0;
        for (j=0;j<department_n && found==0;j+
            ){
            if (department_list[j].deptID ==
                atoi(row[8])){
                dpt_i = j;
                found=1;
            }
        }
        semsubject_list[i].dept_i = dpt_i;

        // counts the number of compatible
        timeslots
        chours = semsubject_list[i].examinhours;

        semsubject_list[i].slotFit = (int*)
            malloc(slot_n_db*sizeof(int));
        totSlotFit = 0;
        for(j=0; j<slot_n_db; j++){

```

```

    shours = timeslots[j].hrs;
    if(shours!=chours)
        semsubject_list[i].slotFit[j] = 0;
        //subject i does not fit
        timeslot j
    else {
        semsubject_list[i].slotFit[j] = 1;
        //subject i fits timeslot j
        totSlotFit++;
    }
}
semsubject_list[i].numFitSlots =
    totSlotFit;

fflush(stdout);
}
}

free(query);
mysql_free_result(res);
mysql_close(&mysql);

section_n = totalSec;

//subject_file = fopen(filename, "w");
fprintf(subject_file, "\n\n%d\n",
    semsubject_n);
fprintf(subject_file, "| courseID |
yrLevel | subjectID | type | labtype |
sec | slots | hrs | subjInd | corsInd
\n");
for(i=0; i<semsubject_n; i++){
    fprintf(subject_file, "%d\qry%d\qry%d\q
ry%d\qry%d\qry%d\qry%d\qry%f\qry%d\q
ry%d\n", semsubject_list[i].crseID,
semsubject_list[i].yearLevel,
semsubject_list[i].subjID,
semsubject_list[i].subjType,
semsubject_list[i].subjlabType,
semsubject_list[i].nSec,
semsubject_list[i].nSlots,
semsubject_list[i].examhours,
semsubject_list[i].sub_ind,
semsubject_list[i].cors_ind);
}
fclose(subject_file);
}

void transformSections(char *filename){
    FILE *section_file;
    int i, j, sec_c=0;

    section_list = (section*) malloc(section_n
        *sizeof(section));

    for(i=0; i<semsubject_n; i++){
        for(j=1; j<=semsubject_list[i].nSec; j++
        ){
            section_list[sec_c].section_i = j;
            section_list[sec_c].semsubject_i = i;
            section_list[sec_c].subject_i =
                semsubject_list[i].sub_ind;
            section_list[sec_c].subject_ii =
                semsubject_list[i].subject_ii;
            section_list[sec_c].subject_id =
                semsubject_list[i].subjID;
            section_list[sec_c].course_i =
                semsubject_list[i].cors_ind;

            section_list[sec_c].yearLevel =
                semsubject_list[i].yearLevel;

            section_list[sec_c].type =
                semsubject_list[i].subjType;
            section_list[sec_c].labtype =
                semsubject_list[i].subjlabType;

            sec_c++;
        }
    }

    section_file = fopen(filename, "w");
    fprintf(section_file, "%d\n", section_n);
    fprintf(section_file, "subject_i/course_i/
sec_i/type/labtype\n");
    for(i=0; i<section_n; i++){
        fprintf(section_file, "%d %d %d %d %d\n"
            , section_list[i].subject_i,
            section_list[i].course_i,
            section_list[i].section_i,
            section_list[i].type, section_list[i
            ].labtype);
    }
    fclose(section_file);
}

void transformRooms(char *filename){
    FILE *room_file;
    int i, j, found=0, dpt_i;

    MYSQL mysql;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;
    char *query;
    int qry, num;

    mysql_init(&mysql);
    if (!mysql_real_connect (&mysql, "localhost"
        , "root", NULL, "CASScheduling", 0, NULL, 0)
        ) {
        printf("Error connecting to database: %s
\n", mysql_error (&mysql));
    } else {
        //printf("Connected...\n");
    }

    query = (char*)malloc(1000*sizeof(char));
    strcpy(query, "SELECT a.roomID, b.bldg, b.
roomName, b.type, b.labtype, b.cap, b.
isShared, b.deptID FROM SEM_ROOMS AS a
JOIN ROOMS as b USING(roomID) WHERE
status=1 AND aysem = ");
    strcat(query, AY_SEM);
    strcat(query, " ORDER BY bldg, roomID,
roomName, type, cap\0");

    qry = mysql_real_query (&mysql, query, (
    unsigned int) strlen(query));
    if (qry) {
        printf("Error making query: %s\n",
            mysql_error (&mysql));
    } else {
        //printf("Query %s made...\n", query);
    }

    res = mysql_store_result (&mysql);
    if (res != NULL) {
        num = mysql_num_rows (res);
        room_n_db = num;
        room_n = room_n_db;
        room_list = (Room*) malloc(num*sizeof(

```

```

        room));
    for(i=0; i<num; i++){
        row = mysql_fetch_row(res);

        room_list[i].roomID = atoi(row[0]);
        room_list[i].bldg = (char*) malloc(
            45*sizeof(char));
        strcpy(room_list[i].bldg, row[1]);
        strcat(room_list[i].bldg, "\0");

        room_list[i].roomName = (char*) malloc(
            745*sizeof(char));
        strcpy(room_list[i].roomName, row[2]);
        strcat(room_list[i].roomName, "\0");

        room_list[i].type = atoi(row[3]);
        room_list[i].labtype = atoi(row[4]);
        room_list[i].capacity = atoi(row[5]);
        room_list[i].isShared = atoi(row[6]);
        room_list[i].dept = atoi(row[7]);

        found=0;
        for (j=0; j<department_n && found==0; j+
            ){
            if (department_list[j].deptID ==
                atoi(row[7])){
                dpt_i = j;
                found=1;
            }
        }

        room_list[i].dept_i = dpt_i;
    }
}

free(query);
mysql_free_result(res);
mysql_close(&mysql);

room_file = fopen(filename, "w");
fprintf(room_file, "%d\n", room_n);
fprintf(room_file, "| bldg | roomName |
    type | labtype | cap | isShared |
    deptID | roomID |\n");
for(i=0; i<num; i++){
    fprintf(room_file, "%s\qry%s\qry%d\qry%d
        \qry%d\qry%d\qry%d\qry%d\n",
        room_list[i].bldg, room_list[i].
        roomName, room_list[i].type,
        room_list[i].labtype, room_list[i].
        capacity, room_list[i].isShared,
        room_list[i].dept, room_list[i].
        roomID);
}
fclose(room_file);

void transformDepartmentRooms(char *filename)
{
    int i, j, *lecN, *labN, dept, type, *
        currLec, *currLab;
    FILE* dept_ptr;

    lecN = (int*) malloc(department_n*sizeof(
        int));
    labN = (int*) malloc(department_n*sizeof(
        int));

    currLec = (int*) malloc(department_n*
        sizeof(int));
    currLab = (int*) malloc(department_n*
        sizeof(int));

    for(i=0; i<department_n; i++){
        lecN[i] = 0;
        labN[i] = 0;
        currLec[i] = 0;
        currLab[i] = 0;
    }

    for(j=0; j<room_n_db; j++){
        dept = room_list[j].dept_i;
        type = room_list[j].type;
        if(type == 1){
            lecN[dept]++;
        } else if(type == 2){
            labN[dept]++;
        } else if(type == 3){
            lecN[dept]++;
            labN[dept]++;
        }
        else{}
    }

    for(i=0; i<department_n; i++){
        department_list[i].numLecRooms = lecN[i]
            ;
        department_list[i].lecRooms = (int*)
            malloc(lecN[i]*sizeof(int));
        department_list[i].numLabRooms = labN[i]
            ;
        department_list[i].labRooms = (int*)
            malloc(labN[i]*sizeof(int));
    }

    for(j=0; j<room_n_db; j++){
        dept = room_list[j].dept_i;
        type = room_list[j].type;
        if(type == 1){
            department_list[dept].lecRooms[
                currLec[dept]] = j;
            currLec[dept]++;
        } else if(type == 2){
            department_list[dept].labRooms[
                currLab[dept]] = j;
            currLab[dept]++;
        } else if(type == 3){
            department_list[dept].lecRooms[
                currLec[dept]] = j;
            currLec[dept]++;
            department_list[dept].labRooms[
                currLab[dept]] = j;
            currLab[dept]++;
        }
        else{}
    }

    free(lecN);
    free(labN);
    free(currLec);
    free(currLab);

    dept_ptr = fopen(filename, "w");
    fprintf(dept_ptr, "%d\n", department_n);
    fprintf(dept_ptr, "name/lecN/labN/lecRooms
        /labRooms\n");
    for(i=0; i<department_n; i++){
        fprintf(dept_ptr, "%s\qry%d\qry%d/",
            department_list[i].deptName,
            department_list[i].numLecRooms,

```

```

        department_list[i].numLabRooms);
    for(j=0; j<department_list[i].
        numLecRooms; j++)
        fprintf(dept_ptr, "%d ",
            department_list[i].lecRooms[j]);
    fprintf(dept_ptr, "/");
    for(j=0; j<department_list[i].
        numLabRooms; j++)
        fprintf(dept_ptr, "%d ",
            department_list[i].labRooms[j]);
    fprintf(dept_ptr, "\n");
}
fclose(dept_ptr);
}

void createFittedSlots(char *filename){
    int i, j, a, b, slotA, slotB, fitSlots;
    FILE *slotFitFile;

    for(i=0; i<semsubject_n_db; i++){
        semsubject_list[i].indexFitSlots = (int*)
            malloc(slot_n_db*sizeof(int));

        for(a=0; a<slot_n_db; a++){
            semsubject_list[i].indexFitSlots[a] =
                a;
        }

        for(a=0; a<slot_n_db; a++){
            for(b=0; b<slot_n_db-a-1; b++){
                slotA = semsubject_list[i].
                    indexFitSlots[b];
                slotB = semsubject_list[i].
                    indexFitSlots[b+1];
                if(semsubject_list[i].slotFit[slotA]
                    < semsubject_list[i].slotFit[
                        slotB]){
                    semsubject_list[i].indexFitSlots[b
                        ] = slotB;
                    semsubject_list[i].indexFitSlots[b
                        +1] = slotA;
                }
            }
        }

        slotFitFile = fopen("slotFitFile.in", "w")
            ;
        fprintf(slotFitFile, "subjectID/hours/
            numFit/fitSlots\n");
        for(i=0; i<semsubject_n_db; i++){
            fprintf(slotFitFile, "%d\qry%f\qry%d\qry
                ", i, semsubject_list[i].examhours,
                    semsubject_list[i].numFitSlots);
            fitSlots = semsubject_list[i].
                numFitSlots;
            for(j=0; j<fitSlots; j++){
                fprintf(slotFitFile, "%d ",
                    semsubject_list[i].indexFitSlots[j
                ]);
            }
            fprintf(slotFitFile, "\n");
        }
        fclose(slotFitFile);
    }
}

```

## timetableOut.h (class scheduling)

```

/*This is the program used to print out
generation values
*****
******/

int* findError(int *x);
int* IntegerArray(float *x, int xsize);
int* slotSort(int *x, int xsize);
int uniqueness(int *x, int xsize, int solNo)
;

int **priorSolution;

void output_timetables(population *
    old_pop_ptr)
{
    int f, l, secval, sval, cval, rval, ival,
        solNo = 0, *printSlot;
    char secchar,err;
    int *sortedSched;
    int realc, reali, realr, realt;
    int intSecType;
    char* secType[] = { "LEC\0", "LAB\0" };
    int *withErr;
    int resultId;
    FILE *ptr1, *ptr2;

    MYSQL mysql;
    char *query,param1[10],param2[10],param3[
        10],param4[10],param5[10],param6[10],
        param03[10],param7[10];

    FILE* mod_ptr = fopen("timetables.php", "w
        ");
    FILE* sol_ptr = fopen("solOutput.txt", "w"
        );
    fprintf(mod_ptr,"<?php session_start();
        if($_SESSION['valid_user']){
            require 'config.php'; require '
            opendb.php'; $query=mysql_query("\
            SELECT * FROM USER WHERE username='\
            . $_SESSION['valid_user'] .'\
            '\");
            $known=mysql_fetch_array($query,
            MYSQL_ASSOC); if($known['usertype
            ']=='A'){?<html>\n<head>\n<meta http-
            equiv=\\"Content-Language\
            \" content=\
            en-us">\n<link href=\\"CASSstyle.css\
            \" rel=\\"stylesheet\
            \" type=\\"text/css\
            \">\n<title>CASS - College of Arts and
            Sciences Scheduler</title>");
            fprintf(mod_ptr,"<style>\n<!--#nav h1 {\n
            text-transform: uppercase;\nfont-size:
            18pt;\n<!--#nav h2 {\n
            text-align: center;\nborder: 1
            px solid #AAA;\nbackground-color: #DDD
            ;\nmargin-bottom: 0;\n}\n#nav h2 {\n
            font-size: 14pt;\n}\n.ms-list3-main {
            border-left-style: none; border-right-
            style: none; border-top: 1.5pt solid
            black; border-bottom: 1.5pt solid
            black; \nbackground-color: white }\n.
            ms-list3-tl { font-weight: bold; color
            : navy; border-left-style: none;
            border-right-style:\nnone; border-top-
            style: none; border-bottom: 1.5pt
            solid black; \n background-color:
            white }\n.ms-list3-top { font-weight:
            bold; color: navy; border-left-style:
            none; border-right-style: \nnone;
            border-top-style: none; border-bottom:
            1.5pt solid black; \n background-color
            : white }\n.ms-list3-left { font-

```

```

weight: normal; color: black; border-
left-style: none; border-right-style:
none; border-top-style: none; border-
bottom: .75pt solid black; background-
color: white }\n.ms-list3-even { font-
weight: normal; color: black; border-
left-style: none; border-right-style:
none; border-top-style: none; \n
border-bottom: .75pt solid black;
background-color: white }\n.ms-color2-
main { border-left-style: none; border-
-right-style: none; border-top-style:
none; border-bottom: 1.5pt solid black
; background-color: silver }\n.ms-
color2-t1 { font-weight: bold; color:
white; border-left-style: none; border-
-right-style: \n none; border-top-
style: none; border-bottom: 1.5pt
solid black; background-color
: maroon }\n.ms-color2-top { font-
weight: bold; color: white; border-
left-style: none; border-right-style:
none; border-top-style: none; border-
bottom: 1.5pt solid black; background-
color: maroon }\n.ms-color2-left {
font-weight: bold; color: black;
border-style: none; background-color:
#FFFFFF }\n.ms-color2-even { color:
black; border-style: none; background-
color: #FFFFFF }\n-->\n</style>");
fprintf(mod_ptr, "</head>\n<body>\n<div
align=\"center\">\n<table border=\"0\"
cellpadding=\"0\" cellspacing=\"0\"
class=\"tbl1\" width=\"854\" style=\"
border-width: 0; \"> <tr> <td>\n</td
> </tr> <tr valign=\"top\"> <td
width=\"95%\" height=\"136\">\n<td
width=\"4\" height=\"136\">\n</tr> <tr
valign=\"top\" align=\"center\"> <
center> <div align=\"left\">\n<div
align=\"left\">\n<table border=\"0\"
width=\"100%\" height=\"24\" class=
tblhead>\n<tbody> <tr>\n<td align=\"
center\" valign=\"top\" height=\"20\"
>\n</td>\n</tr> </table>\n<div
align=\"left\">\n<table border=\"0\"
width=\"100%\" height=\"479\" class=
tblbody\"> <tr>\n<td align=\"center\"
valign=\"top\" rowspan=\"2\">\n<font
size=\"2\">\n</div>\n</tr>";

/*clock_end = clock();*/
time(&tend);

for(f=0; f<popsiz; f++){
old_pop_ptr->ind_ptr = &(old_pop_ptr->
ind[f]);

//printf("Fitness - ");
for(l = 0;l < nfunc;l++){
fprintf(sol_ptr, "%f\t", old_pop_ptr->
ind_ptr->fitness[l]);
}

//printf("\nConstraints - ");
for(l = 0;l < ncons;l++){
fprintf(sol_ptr, "%f\t", old_pop_ptr->
ind_ptr->constr[l]);
}

for(l=0; l<nvar; l=l+3){
sval = (int) old_pop_ptr->ind_ptr->
xreal[l];
rval = (int) old_pop_ptr->ind_ptr->
xreal[l+1];
ival = (int) old_pop_ptr->ind_ptr->
xreal[l+2];
fprintf(sol_ptr, "%d/%d ", sval,
rval);
}
fprintf(sol_ptr, "\n=====
=====");
}
fclose(sol_ptr);

mysql_init(&mysql);
if (!mysql_real_connect (&mysql, "localhost"
, "root", NULL, "CASScheduling", 0, NULL, 0)
) {
printf("Error connecting to database: %s
\n", mysql_error (&mysql));
} else {
//printf("Connected...\n");
}

query = (char*)malloc(1000*sizeof(char));
strcpy(query, "DELETE FROM SUBJECT_RESULTS;
");

t = mysql_real_query (&mysql, query, (
unsigned int) strlen(query));
if (t) {
printf("Error executing DELETE: %s\n",
mysql_error (&mysql));
} else {
//printf("Insert %s made...\n", query);
}

ptr1 = fopen("intRes.txt", "w");
ptr2 = fopen("floatRes.txt", "w");

for(f=0; f<popsiz && solNo<10; f++){
old_pop_ptr->ind_ptr = &(old_pop_ptr->
ind[f]);

if ((old_pop_ptr->ind_ptr->error <= 0.0)
&& (old_pop_ptr->ind_ptr->rank == 1)
)
// for all feasible solutions and non-
dominated solutions
{
sortedSched = IntegerArray(old_pop_ptr
->ind_ptr->xreal, nvar);

if(uniqeness(sortedSched, nvar, solNo
)){
solNo++;
resultId = solNo;
sortedSched = slotSort(sortedSched,
nvar);

fprintf(mod_ptr, "<br><br><br><table
border=1 width=\"465\" class=\"
ms-list3-main\">\n<tr>\n<td
class=\"ms-list3-t1\"> <p align
=\"center\">SOLUTION %d </td>\n<
/tr>\n</table>\n<table border=1
width=\"301\" class=\"ms-list3-
main\">\n<!-- fpstyle: 26,
011111100 -->\n<tr>\n<td class=
\"ms-list3-t1\">\n<td class=
\"ms-list3-top\">\n<font
size=\"2\"> room</font>\n</td>
<td class=\"ms-list3-top\">\n<
font size=\"2\">timeslot</font>

```

```

        \n</td><td class=\"ms-list3-top
        \>\n<font size=\"2\">instructor
        </font>\n</td>\n</tr>\n<tr>\n<td
        width=\"54\" class=\"ms-list3-
        left\">\n<font size=\"2\">
        Fitness</font>\n</td>>, solNo);
for(l = 0;l < nfunc;l++){
    fprintf(mod_ptr,\<td width=\"69\"
        class=\"ms-list3-even\">%f</td
        >\",old_pop_ptr->ind_ptr->
        fitness[l]);
}

fprintf(mod_ptr,\</tr>\n<tr> <td
class=\"ms-list3-left\">\n<font
size=\"2\">Constraints:</font>\n
</td>>);

for(l = 0;l < ncons;l++){
    fprintf(mod_ptr,\<td class=\"ms-
list3-even\">%f</td>\",
        old_pop_ptr->ind_ptr->constr [l
    ]);
}

fprintf(mod_ptr,\</tr>\n</table>\n<
table border=1 width=\"410\"
class=\"ms-color2-main\" height=
\"100\" style=\"border-bottom-
style: solid\">\n<!-- fpstyle: 9
,011111100 -->\n<tr>>);

printSlot = (int*) malloc(slot_n*
sizeof(int));
for(l=0; l<slot_n; l++)
    printSlot[l] = 0;
for(l=0; l<section_n*4; l=1+4){
    secval = sortedSched [l+1];

    cval = section_list [secval].
        subject_i;
    secchar = ' ';
    sval = sortedSched [l];
    printSlot [sval]++;
    rval = sortedSched [l+2];
    ival = sortedSched [l+3];
    intSecType = section_list [secval].
        type - 1;

    realc = subject_list [cval].subjID;
    reali = faculty_list [ival].facID;
    realr = room_list [rval].roomID;
    reall = timeslots [sval].id;

    strcpy(query,\"INSERT INTO
SUBJECT_RESULTS values (\");
    sprintf(param1,\"%d\",resultId);
    strcat(query,param1);
    strcat(query,\",\");

    sprintf(param2,\"%d\",realc);
    strcat(query,param2);
    strcat(query,\",\");

    sprintf(param03,\"%d\",
        section_list [secval].type);
    strcat(query,param03);
    strcat(query,\",\");

    sprintf(param3,\"%d\", secval);
    strcat(query,param3);
    strcat(query,\",\");

    sprintf(param4,\"%d\", reall);
    strcat(query,param4);
    strcat(query,\",\");

    sprintf(param5,\"%d\", reali);
    strcat(query,param5);
    strcat(query,\",\");

    sprintf(param6,\"%d\", reali);
    strcat(query,param6);
    strcat(query,\",\");

    sprintf(param7,\"%d\", 0);
    strcat(query,param7);
    strcat(query,\",\"0\");

    t = mysql_real_query(&mysql,query,
        (unsigned int) strlen(query));
    if (t) {
        printf(\"Error executing insert:
        %s\n\",mysql_error(&mysql));
    } else {
        //printf(\"Insert %s made...\n\",
        query);
    }

    fprintf(ptr1,\"%d %d %d %d %d\n\",
        secval,cval,sortedSched[l+1],
        sortedSched[l+2],sortedSched[l
        +3]);
    fprintf(ptr2,\"%f %f %f %d %d\n\",
        secval,cval,sortedSched[l+1],
        sortedSched[l+2],sortedSched[l
        +3]);

    if(printSlot[sval] == 1){
        fprintf(mod_ptr,\<tr>\n<td
colspan=4 class=\"ms-color2-
top\">\n<font size=\"2\">%s<
/td>\n</tr>\n\",timeslots[
        sval].code);
    }

    fprintf(mod_ptr,\<tr>\n<td width=
\"150\" class=\"ms-color2-even
\">\n<font size=\"2\">%s %s %c
</td>\n<td width=\"110\" class
=\"ms-color2-even\">\n<font
size=\"2\">%s - %s</td>\n<td
width=\"110\" class=\"ms-
color2-even\">\n<font size=\"2
\">%s</td>\n</tr>\n\",
        subject_list [cval].subjCode,
        secType[intSecType], secchar,
        room_list [rval].bldg,
        room_list [rval].roomName,
        faculty_list [ival].facName);
}
fprintf(mod_ptr,\</table>\n\");
fprintf(mod_ptr,\<br><center><a
href=\"/CASScheduler/Implement.
php?sol=%d&table=1\">Implement<
/a></center><br>\",solNo);
free(printSlot);
}

```

```

    }
}

if(solNo == 0){
    //printf("\n\nNo full solutions were
    found.\n");
    fprintf(mod_ptr, "\n\nNo full solutions
    were found.\n");

    for(f=0; f<popsz && solNo<10; f++){
        old_pop_ptr->ind_ptr = &(old_pop_ptr->
            ind[f]);

        if (old_pop_ptr->ind_ptr->rank == 1)
            // for all feasible solutions and non-
            // dominated solutions
        {
            sortedSched = IntegerArray(
                old_pop_ptr->ind_ptr->xreal,
                nvar);
            if(uniqueness(sortedSched, nvar,
                solNo)){
                solNo++;
                resultId = solNo;
                withErr = findError(sortedSched);
                sortedSched = slotSort(sortedSched
                    , nvar);

                fprintf(mod_ptr, "<br><br><br><table
                border=1 width=\"465\" class=\"
                ms-list3-main\">\n<tr>\n<td
                class=\"ms-list3-tl\"> <p align
                =\"center\">SOLUTION %d </td>\n<
                /tr>\n</table>\n<table border=1
                width=\"301\" class=\"ms-list3-
                main\">\n<!-- fpstyle: 26,
                01111100 -->\n<tr>\n<td class=
                \"ms-list3-tl\">&nbsp;</td>\n<td
                class=\"ms-list3-top\">\n<font
                size=\"2\"> room</font>\n</td>\n
                </td> <td class=\"ms-list3-top\"
                >\n<font size=\"2\">timeslot</
                font>\n</td>\n<td class=\"ms-
                list3-top\">\n<font size=\"2\">
                instructor</font>\n</td></tr>\n<
                tr>\n<td width=\"54\" class=\"ms-
                -list3-left\">\n<font size=\"2\"
                >Fitness</font>\n</td>", solNo);

                for(l = 0; l < nfunc;l++){
                    fprintf(mod_ptr, "<td width=\"69\"
                    class=\"ms-list3-even\">%f</td>
                    >", old_pop_ptr->ind_ptr->
                    fitness[l]);
                }

                fprintf(mod_ptr, "</tr>\n<tr> <td
                class=\"ms-list3-left\">\n<font
                size=\"2\">Constraints:</font>\n
                </td>");

                for(l = 0; l < ncons;l++){
                    fprintf(mod_ptr, "<td class=\"ms-
                    list3-even\">%f</td>",
                        old_pop_ptr->ind_ptr->constr[l
                            ]);
                }

                fprintf(mod_ptr, "</tr>\n</table>\n<
                table border=1 width=\"410\"
                class=\"ms-color2-main\" height=
                \"100\" style=\"border-bottom-
                style: solid\">\n<!-- fpstyle: 9
                ,01111100 -->");

                printSlot = (int*) malloc(slot_n*
                    sizeof(int));
                for(l=0; l<slot_n; l++)
                    printSlot[l] = 0;
                for(l=0; l<section_n*4; l=l+4){
                    secval = sortedSched[l+1];
                    cval = section_list[secval].
                        subject_i;
                    secchar = '!';
                    sval = sortedSched[l];
                    printSlot[sval]++;
                    rval = sortedSched[l+2];
                    ival = sortedSched[l+3];
                    intSecType = section_list[secval]
                        .type - 1;

                    realc = subject_list[cval].
                        subjID;
                    reali = faculty_list[ival].facID;
                    realr = room_list[rval].roomID;
                    realt = timeSlots[sval].id;

                    strcpy(query, "INSERT INTO
                    SUBJECT_RESULTS values(");
                    sprintf(param1, "%d", resultId);
                    strcat(query, param1);

                    strcat(query, ",");
                    sprintf(param2, "%d", realc);

                    strcat(query, param2);
                    strcat(query, ",");

                    sprintf(param03, "%d",
                        section_list[secval].type);
                    strcat(query, param03);
                    strcat(query, ",");

                    sprintf(param3, "%d", secval);
                    strcat(query, param3);
                    strcat(query, ",");

                    sprintf(param4, "%d", realt);
                    strcat(query, param4);
                    strcat(query, ",");

                    sprintf(param5, "%d", realr);
                    strcat(query, param5);
                    strcat(query, ",");

                    sprintf(param6, "%d", reali);
                    strcat(query, param6);
                    strcat(query, ",");

                    sprintf(param7, "%d", withErr[
                        secval]);
                    strcat(query, param7);
                    strcat(query, ";\0");

                    t = mysql_real_query(&mysql, query,
                        (unsigned int) strlen(query));
                    if (t) {
                        printf("Error executing insert:
                        %s\n", mysql_error(&mysql));
                    } else {
                        //printf("Insert %s made...\n",
                        query);
                    }
                }
            }
        }
    }
}

```



```

fprintf(ptr1, "%d %d %d %d %d\n",
        secval, cval, sortedSched[1+1],
        sortedSched[1+2], sortedSched[
1+3]);
fprintf(ptr2, "%f %f %f %d %d\n",
        secval, cval, sortedSched[1+1],
        sortedSched[1+2], sortedSched[1
+3]);

if(printsSlot[sval] == 1){
    fprintf(mod_ptr, "<tr>\n<td
        colspan=4 class=\"ms-
        color2-top\">\n<font size=
        \"2\">%s</td>\n</tr>\n",
        timeslots[sval].code);
}

if(withErr[secval] > 0){
    if (withErr[secval]==2)
        err='s';
    else if (withErr[secval]==3)
        err='i';
    else if (withErr[secval]==1)
        err='r';
    else {}

    fprintf(mod_ptr, "<tr>\n<td
        class=\"ms-color2-even\">
        \n<i>\n<font face=\"Times
        New Roman\" color=\"#
        FF0000\">%c</font>\n</i>\n
        </td>\n<td width=\"150\"
        class=\"ms-color2-even\">
        \n<font size=\"2\">%s %s %
        c</td>\n<td width=\"110\"
        class=\"ms-color2-even\">
        \n<font size=\"2\">%s - %s
        </td>\n<td width=\"110\"
        class=\"ms-color2-even\">
        \n<font size=\"2\">%s</td>
        \n</tr>\n", err,
        subject_list[cval].
        subjCode, secType[
        intSecType], secchar,
        room_list[rval].bldg,
        room_list[rval].roomName,
        faculty_list[ival].facName
        );
}
else
    fprintf(mod_ptr, "<tr>\n<td
        class=\"ms-color2-even\">%
        nbsp;</td>\n<td width=\"
        150\" class=\"ms-color2-
        even\">\n<font size=\"2\">
        %s %s %c</td>\n<td width=
        \"110\" class=\"ms-color2-
        even\">\n<font size=\"2\">
        %s - %s</td>\n<td width=\"
        110\" class=\"ms-color2-
        even\">\n<font size=\"2\">
        %s</td>\n</tr>\n",
        subject_list[cval].
        subjCode, secType[
        intSecType], secchar,
        room_list[rval].bldg,
        room_list[rval].roomName,
        faculty_list[ival].facName
        );
}

fprintf(mod_ptr, "</table>\n");
fprintf(mod_ptr, "<br><center><a
        href=\"/CASScheduler/Implement.
        php?sol=%d&table=1\">Implement<
        /a></center><br>", solNo);

        free(printsSlot);
    }
}

fprintf(mod_ptr, "\n\nTIME ELAPSED: %d
        secs<br>\n<br>\n", tend - tstart);
fprintf(mod_ptr, "</div>\n<p>&nbsp;</td>\n
        </tr>\n</table>\n</div>\n</div>\n<div
        align=\"left\">\n<table border=\"0\"
        width=\"100%\" id=\"table4\" height=\"
        22\" class=\"tblfoot\">\n<tbody>\n <
        tr>\n <td align=\"center\" valign=\"
        top\">\n &nbsp;</td>\n </tr>\n </
        table>\n</div>\n</div>\n</center>\n</
        td>\n</tr>\n</tbody>\n</table>\n</div>
        \n</body>\n</html><? } else{ echo(
        \"<script LANGUAGE='javascript'>
        location.replace(\"\"error.php\"");</
        script>\""); } } else{ echo(\"<
        script LANGUAGE='javascript'>location.
        replace(\"\"error.php\"");</script>\"
        ); }?>");
fclose(mod_ptr);
fclose(ptr1);
fclose(ptr2);

free(query);
mysql_close(&mysql);
}

int uniqueness(int *x, int xsize, int solNo)
{
    int i=0, j=0, **temp;
    int unique = 0; // 1 if unique, 0
        otherwise

    if(solNo == 0) {
        unique = 1;
    } else {
        for(i=0; (i<solNo)&&(!unique); i++){
            for(j=0; j<xsize; j++){
                if(priorSolution[i][j] != x[j]){
                    unique = 1;
                    break;
                }
            }
        }
    }

    if(unique) {
        temp = (int**)malloc((solNo+1)*sizeof(
            int*));
        for(i=0; i<solNo; i++)
            temp[i] = priorSolution[i];
        temp[i] = x;
        priorSolution = temp;
    }

    return unique;
}

int* IntegerArray(float *x, int xsize){
    int i;
    int* sortedSlots = (int*) malloc(xsize*

```

```

        sizeof(int));
    for(i=0; i<xsize; i++)
        sortedSlots[i] = (int) x[i];
    return sortedSlots;
}

int* slotSort(int *x, int xsize){
    int *sortedSlots, i, j, k, sval, cval,
        rval, ival;
    sortedSlots = (int*) malloc((section_n*4)*
        sizeof(int));
    /* copy schedule into int array */
    for(i=0, j=0; i<xsize; i++, j++){
        sortedSlots[j] = x[i];
        if(i%3==0){
            j++;
            sortedSlots[j] = i/3;
        }
    }
    for(i=0; i<section_n; i++){
        for(k=0; k<section_n-i-1; k++){
            j = k*4;
            if(sortedSlots[j]>sortedSlots[j+4]){
                sval = sortedSlots[j];
                cval = sortedSlots[j+1];
                rval = sortedSlots[j+2];
                ival = sortedSlots[j+3];

                sortedSlots[j] = sortedSlots[j+4];
                sortedSlots[j+1] = sortedSlots[j+5];
                sortedSlots[j+2] = sortedSlots[j+6];
                sortedSlots[j+3] = sortedSlots[j+7];

                sortedSlots[j+4] = sval;
                sortedSlots[j+5] = cval;
                sortedSlots[j+6] = rval;
                sortedSlots[j+7] = ival;
            }
        }
    }

    return sortedSlots;
}

int* findError(int *x){
    int i, j;
    int sval, rval, ival, cval;
    int* withError, **roomSlot, **facSlot, **
        subjectSlot;

    withError = (int*) malloc(section_n*sizeof(
        int));
    for(i=0; i<section_n; i++)
        withError[i] = 0;

    roomSlot = (int**) malloc(slot_n*sizeof(
        int));
    facSlot = (int**) malloc(slot_n*sizeof(int
        *));
    subjectSlot = (int**) malloc(slot_n*sizeof(
        int));
    for(i=0; i<slot_n; i++){
        roomSlot[i] = (int*) malloc(room_n*
            sizeof(int));
        for(j=0; j<room_n; j++)
            roomSlot[i][j] = 0;
        facSlot[i] = (int*) malloc(faculty_n*
            sizeof(int));
        for(j=0; j<faculty_n; j++)
            facSlot[i][j] = 0;
        subjectSlot[i] = (int*) malloc(
            semsubject_n*sizeof(int));
        for(j=0; j<semsubject_n; j++)

```

```

        subjectSlot[i][j] = 0;
    }
}

for(i=0; i<section_n; i++){
    sval = x[i*3];
    rval = x[i*3+1];
    ival = x[i*3+2];
    cval = section_list[i].semsubject_i;

    roomSlot[sval][rval]++;
    facSlot[sval][ival]++;
    subjectSlot[sval][cval]++;
}

for(i=0; i<section_n; i++){
    sval = x[i*3];
    rval = x[i*3+1];
    ival = x[i*3+2];
    cval = section_list[i].semsubject_i;

    if(roomSlot[sval][rval] > 1)
        withError[i] = 1;
    else if(facSlot[sval][ival] > 1)
        withError[i] = 3;
    else if(subjectSlot[sval][cval] > 1)
        withError[i] = 2;
    else if(semsubject_list[cval].slotFit(
        sval) == 0)
        withError[i] = 2;
}

return withError;
}
}

```

## timetableOut.h (final exam scheduling)

```

/*This is the program used to print out
generation values
*****
*****/

int* IntegerArray(float *x, int xsize);
int uniqueness(int *x, int xsize, int solNo)
;
int* slotSort(int *x, int xsize);
int* findError(int *x);

int **priorSolution;

void output_timetable(population *
    old_pop_ptr)
{
    int f, l, secval, sval, cval, rval, ival,
        solNo = 0, *printSlot;
    char secchar, err;
    int *sortedSched;
    int realc, reali, realr, reall;
    int intSecType;
    char* secType[] = { "LEC\0", "LAB\0" };
    int *withErr;
    int resultId;
    FILE *ptr1, *ptr2;

    MYSQL mysql;
    char *query, param1[10], param2[10], param3[
        10], param03[10], param4[10], param5[10],
        param6[10];

    FILE* mod_ptr = fopen("mod_output.php", "w
        ");
    FILE* sol_ptr = fopen("solOutput.txt", "w
        ");
}

```

```

fprintf(mod_ptr, "<?php session_start();
if($_SESSION['valid_user']){
require 'config.php'; require '
opendb.php'; $query=mysql_query(\
SELECT * FROM USER WHERE username='\
. $_SESSION['valid_user'] .'\");
$known=mysql_fetch_array($query,
MYSQL_ASSOC); if($known['usertype
']=='A'){?><html>\n<head>\n<meta http-
equiv=\"Content-Language\" content=\"
en-us\">\n<link href=\"CASSstyle.css\"
rel=\"stylesheet\" type=\"text/css\">
\n<title>CASS - College of Arts and
Sciences Scheduler</title>");
fprintf(mod_ptr, "<style>\n<!--#nav h1 {\n
text-transform: uppercase;\nfont-size:
18pt;\ntext-align: center;\nborder: 1
px solid #AAA;\nbackground-color: #DDD
;\nmargin-bottom: 0;\n}\n#nav h2 {\n
font-size: 14pt;\n}\n.ms-list3-main {
border-left-style: none; border-right-
style: none; \nborder-top: 1.5pt solid
black; border-bottom: 1.5pt solid
black; \nbackground-color: white }\n.
ms-list3-tl { font-weight: bold; color
: navy; border-left-style: none;
border-right-style:\nnone; border-top-
style: none; border-bottom: 1.5pt
solid black; \n background-color:
white }\n.ms-list3-top { font-weight:
bold; color: navy; border-left-style:
none; border-right-style: \nnone;
border-top-style: none; border-bottom:
1.5pt solid black; \n background-color
: white }\n.ms-list3-left { font-
weight: normal; color: black; border-
left-style: none; border-right-style:
none; border-top-style: none; border-
bottom: .75pt solid black; background-
color: white }\n.ms-list3-even { font-
weight: normal; color: black; border-
left-style: none; border-right-style:
none; border-top-style: none; \n
border-bottom: .75pt solid black;
background-color: white }\n.ms-color2-
main { border-left-style: none; border
-right-style: none; border-top-style:
none; border-bottom: 1.5pt solid black
; background-color: silver }\n.ms-
color2-tl { font-weight: bold; color:
white; border-left-style: none; border
-right-style: \n none; border-top-
style: none; border-bottom: 1.5pt
solid black; background-color
: maroon }\n.ms-color2-top { font-
weight: bold; color: white; border-
left-style: none; border-right-style:
none; border-top-style: none; border-
bottom: 1.5pt solid black; background-
color: maroon }\n.ms-color2-left {
font-weight: bold; color: black;
border-style: none; background-color:
#FFFFFF }\n.ms-color2-even { color:
black; border-style: none; background-
color: #FFFFFF }\n-->\n</style>");
fprintf(mod_ptr, "</head>\n<body>\n<div
align=\"center\">\n<table border=\"0\"
cellpadding=\"0\" cellspacing=\"0\"
class=\"tbl1\" width=\"854\" style=\"
border-width: 0; \"> <tr> <td>\n</td
> </tr> <tr valign=\"top\"> <td
width=\"95%\" height=\"136\">&nbsp;&nbsp;&nbsp;</
td> </tr> <tr> <td> <tr
valign=\"top\" align=\"center\"> <
center> <div align=\"left\">\n<div
align=\"left\">\n<table border=\"0\"
width=\"100%\" height=\"24\" class=
tblhead>\n<TBODY> <tr>\n<td align=\"
center\" valign=\"top\" height=\"20\">
&nbsp;&nbsp;&nbsp;</td>\n</tr> </table>\n<div

```

```

if(uniqness(sortedSched, nvar, solNo
){
solNo++;
resultId = solNo;
sortedSched = slotSort(sortedSched,
nvar);

fprintf(mod_ptr, "<br><br><br><table
border=1 width=\"465\" class=\"
ms-list3-main\"><n<tr><n<td
class=\"ms-list3-tl\"> <p align
=\"center\">SOLUTION %d </td><n<
/tr><n</table><n<table border=1
width=\"301\" class=\"ms-list3-
main\"><n<!-- fpstyle: 26,
01111100 --><n<tr><n<td class=
\"ms-list3-tl\">&nbsp;</td><n<td
class=\"ms-list3-top\"><n<font
size=\"2\"> room</font><n</td>
<td class=\"ms-list3-top\"><n<
font size=\"2\">timeslot</font>
<n</td><n</tr><n<tr><n<td width=
\"54\" class=\"ms-list3-left\">
<n<font size=\"2\">Fitness</font

><n</td>", solNo);

for(l = 0; l < nfunc; l++){
fprintf(mod_ptr, "<td width=\"69\"
class=\"ms-list3-even\">%f</td>
", old_pop_ptr->ind_ptr->
fitness[l]);
}

fprintf(mod_ptr, "</tr><n<tr> <td
class=\"ms-list3-left\"><n<font
size=\"2\">Constraints:</font><n
</td>");

for(l = 0; l < ncons; l++){
fprintf(mod_ptr, "<td class=\"ms-
list3-even\">%f</td>",
old_pop_ptr->ind_ptr->constr[l
]);
}

fprintf(mod_ptr, "</tr><n</table><n<
table border=1 width=\"410\"
class=\"ms-color2-main\" height=
\"100\" style=\"border-bottom-
style: solid\"><n<!-- fpstyle: 9
,01111100 --><n<tr>");

printSlot = (int*) malloc(slot_n*
sizeof(int));
for(l=0; l<slot_n; l++)
printSlot[l] = 0;
for(l=0; l<section_n*3; l=l+3){
secval = sortedSched[l+1];

cval = section_list[secval].
subject i;
secchar = '-';
sval = sortedSched[l];
printSlot[sval]++;
rval = sortedSched[l+2];
intSecType = section_list[secval].
type - 1;

realc = subject_list[cval].subjID;
realr = room_list[rval].roomID;
realt = timeslots[sval].id;

strcpy(query, "INSERT INTO
EXAM_RESULTS values(");
sprintf(param1, "%d", resultId);
strcat(query, param1);

strcat(query, ",");
sprintf(param2, "%d", realc);

strcat(query, param2);
strcat(query, ",");

sprintf(param03, "%d",
section_list[secval].type);
strcat(query, param03);
strcat(query, ",");

sprintf(param3, "%d", secval);
strcat(query, param3);
strcat(query, ",");

sprintf(param4, "%d", realt);
strcat(query, param4);
strcat(query, ",");

sprintf(param5, "%d", realr);
strcat(query, param5);
strcat(query, ",");

sprintf(param6, "%d", 0);
strcat(query, param6);
strcat(query, ");\0");

t = mysql_real_query(&mysql, query,
(unsigned int) strlen(query));
if (t) {
printf("Error executing insert:
%s\n", mysql_error(&mysql));
} else {
//printf("Insert %s made...\n",
query);
}

fprintf(ptr1, "%d %d %d %d\n",
secval, cval, sortedSched[l+1],
sortedSched[l+2]);
fprintf(ptr2, "%f %f %f %d\n",
secval, cval, sortedSched[l+1],
sortedSched[l+2]);

if(printSlot[sval] == 1){
fprintf(mod_ptr, "<tr><n<td
colspan=4 class=\"ms-color2-
top\"><n<font size=\"2\">%s<
/td><n</tr><n", timeslots[
sval].code);
}

fprintf(mod_ptr, "<tr><n<td width=
\"150\" class=\"ms-color2-even
\"><n<font size=\"2\">%s %s %c
</td><n<td width=\"110\" class
=\"ms-color2-even\"><n<font
size=\"2\">%s - %s</td><n</tr>
\n", subject_list[cval].
subjCode, secType[intSecType],
secchar, room_list[rval].bldg,
room_list[rval].roomName);

fprintf(mod_ptr, "</table><n");
fprintf(mod_ptr, "<br><center><a
href=\"/CASScheduler/implement.
php?sol=%d&ttable=2\">Implement<
/a></center><br>", solNo);
free(printSlot);
}
}
}

```

```

if(solNo == 0){
    //printf("\n\nNo full solutions were
    found.\n");
    fprintf(mod_ptr, "\n\nNo full solutions
    were found.\n");
    for(f=0; f<popsize && solNo<10; f++){
        old_pop_ptr->ind_ptr = &(old_pop_ptr->
            ind[f]);

        if (old_pop_ptr->ind_ptr->rank == 1)
            // for all feasible solutions and non-
            // dominated solutions
            {
                sortedSched = IntegerArray(
                    old_pop_ptr->ind_ptr->xreal,
                    nvar);

                if(uniqeness(sortedSched, nvar,
                    solNo)){
                    solNo++;
                    resultId = solNo;
                    withErr = findError(sortedSched);
                    sortedSched = slotSort(sortedSched
                        , nvar);

                    fprintf(mod_ptr, "<br><br><br><table
                    border=1 width=\\"465\\" class=\
                    ms-list3-main\\>\n<tr>\n<td
                    class=\\"ms-list3-t1\\> <p align
                    =\\"center\\">SOLUTION &d </td>\n<
                    /tr>\n</table>\n<table border=1
                    width=\\"301\\" class=\\"ms-list3-
                    main\\>\n<!-- fpstyle: 26,
                    011111100 -->\n<tr>\n<td class=
                    \\"ms-list3-t1\\>&nbsp;</td>\n<td
                    class=\\"ms-list3-top\\>\n<font
                    size=\\"2\\"> room</font>\n</td>\n
                    </td> <td class=\\"ms-list3-top\\"
                    >\n<font size=\\"2\\">timeslot</
                    font>\n</td>\n</tr>\n<tr>\n<td
                    width=\\"54\\" class=\\"ms-list3-
                    left\\>\n<font size=\\"2\\">
                    Fitness</font>\n</td>", solNo);

                    for(l = 0; l < nfunc; l++){
                        fprintf(mod_ptr, "<td width=\\"69\\"
                            class=\\"ms-list3-even\\">%f</td>
                            >", old_pop_ptr->ind_ptr->
                            fitness[l]);
                    }

                    fprintf(mod_ptr, "</tr>\n<tr> <td
                    class=\\"ms-list3-left\\">\n<font
                    size=\\"2\\">Constraints:</font>\n
                    </td>");

                    for(l = 0; l < ncons; l++){
                        fprintf(mod_ptr, "<td class=\\"ms-
                            list3-even\\">%f</td>",
                            old_pop_ptr->ind_ptr->constr[l
                                ]);
                    }

                    fprintf(mod_ptr, "</tr>\n</table>\n<
                    table border=1 width=\\"410\\"
                    class=\\"ms-color2-main\\" height=
                    \\"100\\" style=\\"border-bottom-
                    style: solid\\>\n<!-- fpstyle: 9
                    ,011111100 -->");

                    printSlot = (int*) malloc(slot_n*
                        sizeof(int));
                    for(l=0; l<slot_n; l++)
                        printSlot[l] = 0;
                    for(l=0; l<section_n*3; l=l+3){
                        secval = sortedSched[l+1];
                        cval = section_list[secval].

                subject_i;
                secchar = ' ';
                sval = sortedSched[l];
                printSlot[sval]++;
                rval = sortedSched[l+2];
                intSecType = section_list[secval]
                    .type - 1;

                realc = subject_list[cval].
                    subjID;
                realr = room_list[rval].roomID;

                realt = timeslots[sval].id;

                strcpy(query, "INSERT INTO
                EXAM_RESULTS values(");
                sprintf(param1, "%d", resultId);
                strcat(query, param1);

                strcat(query, ",");
                sprintf(param2, "%d", realc);

                strcat(query, param2);
                strcat(query, ",");

                sprintf(param03, "%d",
                    section_list[secval].type);
                strcat(query, param03);
                strcat(query, ",");

                sprintf(param3, "%d", secval);
                strcat(query, param3);
                strcat(query, ",");

                sprintf(param4, "%d", realt);
                strcat(query, param4);
                strcat(query, ",");

                sprintf(param5, "%d", realr);
                strcat(query, param5);
                strcat(query, ",");

                sprintf(param6, "%d", withErr[
                    secval]);
                strcat(query, param6);
                strcat(query, ",");

                t = mysql_real_query(&mysql, query,
                    (unsigned int) strlen(query));
                if (t) {
                    printf("Error executing insert:
                    %s\n", mysql_error(&mysql));
                } else {
                    //printf("Insert %s made...\n",
                    query);
                }

                fprintf(ptr1, "%d %d %d %d\n",
                    secval, cval, sortedSched[l+1],
                    sortedSched[l+2]);
                fprintf(ptr2, "%f %f %f %d\n",
                    secval, cval, sortedSched[l+1],
                    sortedSched[l+2]);

                if(printSlot[sval] == 1){
                    fprintf(mod_ptr, "<tr>\n<td
                    colspan=4 class=\\"ms-
                    color2-top\\">\n<font size=
                    \\"2\\">%s</td>\n</tr>\n",
                    timeslots[sval].code);
                }

                if(withErr[secval] > 0){
                    if (withErr[secval]==2)
                        err='s';
                    else if (withErr[secval]==1)
                        err='r';
                    else {}
                }
            }
}

```

```

fprintf(mod_ptr, "<tr>\n<td
width=\"10\" class=\"ms-
color2-even\">\n<i>\n<font
face=\"Times New Roman\"
color=\"#FF0000\">%c</font

>\n</i>\n</td>\n<td width=
\"150\" class=\"ms-color2-
even\">\n<font size=\"2\">
%s %s %c</td>\n<td width=
\"110\" class=\"ms-color2-
even\">\n<font size=\"2\">
%s - %s</td>\n</tr>\n",err
, subject_list[cval].
subjCode, secType[
intSecType], secchar,
room_list[rval].bldg,
room_list[rval].roomName);
}
else
fprintf(mod_ptr, "<tr>\n<td
class=\"ms-color2-even\">&
nbsp;</td>\n<td width=\"
150\" class=\"ms-color2-
even\">\n<font size=\"2\">
%s %s %c</td>\n<td width=
\"110\" class=\"ms-color2-
even\">\n<font size=\"2\">
%s - %s</td>\n</tr>\n",
subject_list[cval].
subjCode, secType[
intSecType], secchar,
room_list[rval].bldg,
room_list[rval].roomName);
}
fprintf(mod_ptr, "</table>\n");
fprintf(mod_ptr, "<br><center><a
href=\"/CASScheduler/implement.
php?sol=%d&ttable=2\">Implement<
/a></center><br>", solNo);

free(printSlot);
}
}
}

fprintf(mod_ptr, "\n\nTIME ELAPSED: %d
secs<br>\n<br>\n", tend - tstart);
fprintf(mod_ptr, "</div>\n<p>&nbsp;</td>\n
</tr>\n </table>\n</div>\n</div>\n<div
align=\"left\">\n<table border=\"0\"
width=\"100%\" id=\"table4\" height=\"
22\" class=\"tblfoot\">\n<TBODY>\n <
tr>\n <td align=\"center\" valign=\"
top\">\n &nbsp;</td>\n </tr>\n </
table>\n</div>\n</div>\n</center>\n</
td>\n</tr>\n</TBODY>\n</table>\n</div>
\n</body>\n</html><? } else{ echo(
\"<script LANGUAGE='javascript'>
location.replace(\"\\\"error.php\\\")</
script>\"); } } else{ echo(\"<
script LANGUAGE='javascript'>location.
replace(\"\\\"error.php\\\")</script>\
\"); }?>");
fclose(mod_ptr);
fclose(ptr1);
fclose(ptr2);

free(query);
mysql_close(&mysql);
}

int uniqueness(int *x, int xsize, int solNo)
{
int i=0, j=0,**temp;
int unique = 0; // 1 if unique, 0
otherwise
if(solNo == 0) {
unique = 1;
} else {
for(i=0; (i<solNo)&&!unique); i++){
for(j=0; j<xsize; j++){
if(priorSolution[i][j] != x[j]){
unique = 1;
break;
}
}
}
}
if(unique) {
temp = (int**)malloc((solNo+1)*sizeof(
int*));
for(i=0; i<solNo; i++)
temp[i] = priorSolution[i];
temp[i] = x;
priorSolution = temp;
}

return unique;
}

int* IntegerArray(float *x, int xsize){
int i;
int* arrayInt = (int*) malloc(xsize*sizeof
(int));
for(i=0; i<xsize; i++)
arrayInt[i] = (int) x[i];
return arrayInt;
}

int* slotSort(int *x, int xsize){
int *sortedSlots, i, j, k, sval, cval,
rval, ival;

sortedSlots = (int*) malloc((section_n*3)*
sizeof(int));

/* copy schedule into int array */
for(i=0, j=0; i<xsize; i++, j++){
sortedSlots[j] = x[i];
if(i%2==0){
j++;
sortedSlots[j] = i/2;
}
}

for(i=0; i<section_n; i++){
for(k=0; k<section_n-i-1; k++){
j = k*3;
if(sortedSlots[j]>sortedSlots[j+3]){
sval = sortedSlots[j];
cval = sortedSlots[j+1];
rval = sortedSlots[j+2];

sortedSlots[j] = sortedSlots[j+3];
sortedSlots[j+1] = sortedSlots[j+4];
sortedSlots[j+2] = sortedSlots[j+5];

sortedSlots[j+3] = sval;
sortedSlots[j+4] = cval;
sortedSlots[j+5] = rval;
}
}
}

return sortedSlots;
}

```

```

int* findError(int *x){
    int i, j;
    int sval, rval, ival, cval;
    int* withError, **roomSlot, **subjectSlot;

    withError = (int*) malloc(section_n*sizeof
        (int));
    for(i=0; i<section_n; i++)
        withError[i] = 0;

    roomSlot = (int**) malloc(slot_n*sizeof(
        int*));
    subjectSlot = (int**) malloc(slot_n*sizeof
        (int*));
    for(i=0; i<slot_n; i++){
        roomSlot[i] = (int*) malloc(room_n*
            sizeof(int));
        for(j=0; j<room_n; j++)
            roomSlot[i][j] = 0;

        subjectSlot[i] = (int*) malloc(
            semsubject_n*sizeof(int));
        for(j=0; j<semsubject_n; j++)
            subjectSlot[i][j] = 0;
    }

    for(i=0; i<section_n; i++){
        sval = x[i*2];
        rval = x[i*2+1];
        cval = section_list[i].semsubject_i;

        roomSlot[sval][rval]++;
        subjectSlot[sval][cval]++;
    }

    for(i=0; i<section_n; i++){
        sval = x[i*2];
        rval = x[i*2+1];
        cval = section_list[i].semsubject_i;

        if(roomSlot[sval][rval] > 1)
            withError[i] = 1;
        else if(subjectSlot[sval][cval] > 1)
            withError[i] = 2;
        else if(semsubject_list[cval].slotFit[
            sval] == 0)
            withError[i] = 2;
    }

    return withError;
}

```

## mutation.h (class scheduling)

```

void test_mutate(population *new_pop_ptr);
void mutate_room(population *new_pop_ptr,
    int** roomSlot, float roomProb, int i,
    int j);
void mutate_faculty(population *new_pop_ptr,
    int **facSlot, float instProb, int i,
    int j);
void mutate_slot(population *new_pop_ptr,
    int i, int j, int cval, int inst);
void rand_mutate(population *new_pop_ptr,
    int i, int j);

FILE *mut_test;

int *freeRooms, *freeInsts;
int numFreeRooms, numFreeInst;
FILE *mut_file;

void mutate_intel(population *new_pop_ptr)
{
    int i, j, k, l;
    float rnd;

    int secval, cval, room_var, slot_var,
        inst_var;

```

```

float probMutRoom = 0, probMutInst = 0,
    probMutSlot = 0;
int **roomSlot, **facSlot;

//mut_file = fopen("mut_file.txt", "w");

roomSlot = (int**) malloc(slot_n*sizeof(
    int*));
facSlot = (int**) malloc(slot_n*sizeof(int
    *));
for(i=0; i<slot_n; i++){
    roomSlot[i] = (int*) malloc(room_n*
        sizeof(int));
    facSlot[i] = (int*) malloc(faculty_n*
        sizeof(int));
}

for(j = 0; j < popsize; j++){
    for(i=0; i<slot_n; i++){
        for(k=0; k<room_n; k++)
            roomSlot[i][k] = 0;
        for(k=0; k<faculty_n; k++)
            facSlot[i][k] = 0;
    }

    for(i=0; i<nvar; i=i+3){
        slot_var = (int) new_pop_ptr->ind[j].
            xreal[i];
        room_var = (int) new_pop_ptr->ind[j].
            xreal[i+1];
        inst_var = (int) new_pop_ptr->ind[j].
            xreal[i+2];
        for(k=0; k<slot_n; k++){
            if(overlaps[slot_var][k] > 0){
                roomSlot[k][room_var]++;
            }
            if(overlaps[slot_var][k] > 0){
                facSlot[k][inst_var]++;
            }
        }
    }

    for (i = 0; i < nvar; i = i+3){
        numFreeRooms = 0;
        numFreeInst = 0;
        for(k=0; k<slot_n; k++){
            for(l=0; l<room_n; l++){
                if(roomSlot[k][l]==0)
                    numFreeRooms++;
            }
            for(l=0; l<faculty_n; l++){
                if(facSlot[k][l]==0)
                    numFreeInst++;
            }
        }

        slot_var = (int) new_pop_ptr->ind[j].
            xreal[i];
        room_var = (int) new_pop_ptr->ind[j].
            xreal[i+1];
        inst_var = (int) new_pop_ptr->ind[j].
            xreal[i+2];

        fflush(stdout);
        secval = i/3;
        cval = section_list[secval].
            semsubject_i;

        if(roomSlot[slot_var][room_var] > 1){
            probMutRoom = pmut_r + (1.0/nvar)*
                0.2;
            if(probMutRoom >= (1.0/nvar))
                probMutRoom = (1.0/nvar) * 0.2;
        } else
            probMutRoom = pmut_r;

        if(facSlot[slot_var][inst_var] > 1) {
            probMutInst = pmut_r + (1.0/nvar)*
                0.15;

```

```

        if(probMutInst >= (1.0/nvar))
            probMutInst = (1.0/nvar) * 0.925;
    } else
        probMutInst = pmut_r;

    if(probMutRoom>probMutInst){
        mutate_faculty(new_pop_ptr, facSlot,
            probMutInst, i, j);
        mutate_room(new_pop_ptr, roomSlot,
            probMutRoom, i, j);
    } else if(probMutInst>probMutRoom) {
        mutate_room(new_pop_ptr, roomSlot,
            probMutRoom, i, j);
        mutate_faculty(new_pop_ptr, facSlot,
            probMutInst, i, j);
    } else {
        rnd = randomperc();
        if(rnd<0.5){
            mutate_room(new_pop_ptr, roomSlot,
                probMutRoom, i, j);
            mutate_faculty(new_pop_ptr,
                facSlot, probMutInst, i, j);
        } else {
            mutate_faculty(new_pop_ptr,
                facSlot, probMutInst, i, j);
            mutate_room(new_pop_ptr, roomSlot,
                probMutRoom, i, j);
        }
    }

    fflush(stdout);

    slot_var = new_pop_ptr->ind[j].xreal[i
        ];
    room_var = new_pop_ptr->ind[j].xreal[i
        +1];
    inst_var = new_pop_ptr->ind[j].xreal[i
        +2];

    probMutSlot = pmut_r;
    slot_var = (int) new_pop_ptr->ind[j].
        xreal[i];
    fflush(stdout);
    if(semsubject_list[cval].slotFit[
        slot_var] != 1){
        probMutSlot = pmut_r + (1.0/nvar)*
            0.2;
        if(probMutSlot >= (1.0/nvar))
            probMutSlot = (1.0/nvar) * 0.925;
    }

    rnd = randomperc();
    if(rnd<probMutSlot)
        mutate_slot(new_pop_ptr, i, j, cval,
            inst_var);

    fflush(stdout);

    slot_var = new_pop_ptr->ind[j].xreal[i
        ];
    room_var = new_pop_ptr->ind[j].xreal[i
        +1];
    inst_var = new_pop_ptr->ind[j].xreal[i
        +2];

    for(l=0; l<slot_n; l++){
        if(overlaps[l][slot_var]>1)
            roomSlot[l][room_var]++;
        if(overlaps[l][slot_var]>0)
            facSlot[l][inst_var]++;
    }
}
}
for(i=0; i<slot_n; i++){
    free(roomSlot[i]);
    free(facSlot[i]);
}
}

free(roomSlot);
free(facSlot);
free(freeRooms);
free(freeInsts);
return ;
}

void mutate_room(population *new_pop_ptr,
    int **roomSlot, float roomProb, int i,
    int j){
    float rnd;
    int dval, rtype, rltype, secval, sval,
        cval, scval, rval, rmCount, curr_r,
        curr_s, tries = 0, done = 0, k, l, m,
        conf,r;
    int *rooms;
    int *roomSel;

    secval = (int) i/3;
    cval = section_list[secval].semsubject_i;
    scval = section_list[secval].subject_i;
    dval = subject_list[scval].dept_i;
    rtype = section_list[secval].type;
    rltype = section_list[secval].labtype;
    if(rtype == LECTURE){
        rmCount = department_list[dval].
            numLecRooms;
        rooms = (int*) malloc(rmCount*sizeof(int
            ));
        for(r=0;r<rmCount;r++){
            rooms[r] = department_list[dval].
                lecRooms[r];
        }
    } else {
        rmCount = department_list[dval].
            numLabRooms;
        rooms = (int*) malloc(rmCount*sizeof(int
            ));
        for(r=0;r<rmCount;r++){
            rooms[r] = department_list[dval].
                labRooms[r];
        }
    }
    rnd = randomperc();
    if(rnd<roomProb){
        roomSel = (int*) malloc(rmCount*sizeof(
            int));
        for(k=0; k<rmCount; k++)
            roomSel[k] = 0;

        for (tries=0; tries<rmCount && done==0
            ; tries++){
            for (curr_r=(int) (randomperc() *
                rmCount - 0.01);(curr_r<rmCount
                ) && (roomSel[curr_r]!=0) && (
                room_list[rooms[curr_r]].labtype
                !=rltype); curr_r=(int) (
                randomperc() * rmCount)){
            }
            roomSel[curr_r] = 1;

            if(rtype == LECTURE)
                curr_r = department_list[dval].
                    lecRooms[curr_r];
            else
                curr_r = department_list[dval].
                    labRooms[curr_r];

            for(k=0; (k<semsubject_list[cval].
                numFitSlots)&&(done==0); k++){
                curr_s = semsubject_list[cval].
                    indexFitSlots[k];

```



```

    if(roomSlot[curr_s][curr_r] == 0){
        conf = 0;
        for(l=0; l<slot_n; l++){
            if((overlaps[curr_s][l]>1)&&(
                roomSlot[l][curr_r]>0))
                conf = 1;
            if(conf==0){
                new_pop_ptr->ind[j].xreal[i] =
                    curr_s;
                new_pop_ptr->ind[j].xreal[i+1] =
                    curr_r;
                done = 1;
            }
        }
    }
}
free(roomSel);
free(rooms);

if(done == 0){
    if(rtype == LECTURE){
        rmCount = department_list[0].
            numLecRooms;
        rooms = (int*) malloc(rmCount*sizeof
            (int));
        for(r=0; r<rmCount; r++){
            rooms[r] = department_list[0].
                lecRooms[r];
        }
    }
    else{
        rmCount = department_list[0].
            numLabRooms;
        rooms = (int*) malloc(rmCount*sizeof
            (int));
        for(r=0; r<rmCount; r++){
            rooms[r] = department_list[0].
                labRooms[r];
        }
    }

    roomSel = (int*) malloc(rmCount*sizeof
        (int));
    tries = 0;
    for(k=0; k<rmCount; k++){
        roomSel[k] = 0;
        for (tries=0; tries<rmCount && done==0
            ; tries++){
            for (curr_r=(int) (randomperc() *
                rmCount); ((curr_r<rmCount) && (
                roomSel[curr_r]!=0) && (
                room_list[rooms[curr_r]].labtype
                !=rltype)); curr_r = (int) (
                randomperc() * rmCount)){
            }

            roomSel[curr_r] = 1;

            if(rtype == LECTURE)
                curr_r = department_list[0].
                    lecRooms[curr_r];
            else
                curr_r = department_list[0].
                    labRooms[curr_r];

            for(k=0; (k<semsubject_list[cval].
                numFitSlots)&&(done==0); k++){
                curr_s = semsubject_list[cval].
                    indexFitSlots[k];

                if(roomSlot[curr_s][curr_r] == 0
                ){
                    conf = 0;
                    for(l=0; l<slot_n; l++){
                        if((overlaps[curr_s][l]>1)&&(
                            roomSlot[l][curr_r]>0))
                            conf = 1;
                        if(conf==0){
                            new_pop_ptr->ind[j].xreal[
                                i] = (float) curr_s;
                                new_pop_ptr->ind[j].xreal[
                                    i+1] = (float) curr_r;
                                    done = 1;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
free(roomSel);
free(rooms);

if(done == 0){
    rnd = randomperc();
    if((numFreeRooms>0)&&(rnd<0.85)){
        rnd = randomperc() * numFreeRooms -
            0.01;
        k = (int) rnd + 1;
        for(l=0; l<slot_n && k>0; l++){
            for(m=0; m<room_n && k>0; m++){
                if(roomSlot[l][m] == 0){
                    k--;
                    if(k==0){
                        sval = l;
                        rval = m;
                    }
                }
            }
        }
        rnd = randomperc();
        new_pop_ptr->ind[j].xreal[i] = (
            float) sval;
        new_pop_ptr->ind[j].xreal[i+1] = (
            float) rval;
    }
    else {
        rand_mutate(new_pop_ptr, i, j);
        rand_mutate(new_pop_ptr, i+1, j);
    }
}
nmut++;
}

void mutate_faculty(population *new_pop_ptr,
    int **facSlot, float instProb, int i,
    int j){
    float rnd;
    int a, b, c, k, l, m, ival, secval, sval,
        scval, cval, curr_s, done = 0, conf,
        countInst;

    secval = (int) i/3;
    cval = section_list[secval].semsubject_i;
    scval = section_list[secval].subject_i;
    countInst = subject_list[scval].
        numOfFitInsts;

    rnd = randomperc();
    if(rnd<instProb){
        for(a=0; a<countInst && done==0; a++){
            ival = subject_list[scval].fac[a];

            for(b=0; (b<semsubject_list[cval].
                numFitSlots) && (done==0); b++){
                curr_s = semsubject_list[cval].
                    indexFitSlots[b];
                if(facSlot[curr_s][ival] == 0){
                    conf = 0;
                    for(c=0; c<slot_n; c++){
                        if((overlaps[curr_s][c]>0)&&(
                            facSlot[c][ival]>0))
                            conf = 1;
                        if(conf==0){
                            new_pop_ptr->ind[j].xreal[i] = (
                                float) curr_s + rnd;
                                new_pop_ptr->ind[j].xreal[i+2] =
                                    (float) ival + rnd;
                                    done = 1;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
}

if(done==0){
    rnd = randomperc();
    if((numFreeInst > 0)&&(rnd<0.85)){
        rnd = randomperc() * numFreeInst -
            0.01;
        k = (int) rnd + 1;
        for(l=0; l<slot_n && k>0; l++){
            for(m=0; m<faculty_n && k>0; m++){
                if(facSlot[l][m] == 0){
                    k--;
                    if(k==0){
                        sval = l;
                        ival = m;
                    }
                }
            }
        }
        rnd = randomperc();
        new_pop_ptr->ind[j].xreal[i] = (
            float) sval + rnd;
        new_pop_ptr->ind[j].xreal[i+2] = (
            float) ival + rnd;
    } else {
        fflush(stdout);
        rand_mutate(new_pop_ptr, i, j);
        rand_mutate(new_pop_ptr, i+1, j)
    }
}
nmut++;
}

void mutate_slot(population *new_pop_ptr,
    int i, int j, int cval, int inst){
    float rnd;
    int curr;
    int countFit=0;
    int a, b, done;

    countFit = semsubject_list[cval].
        numFitSlots;

    rnd = randomperc();
    if(rnd>0.8){
        done=0;
        rnd = randomperc();
        for (a=0;a<countFit && done==0;a++){
            // curr = (int) rnd * countFit - 0.01;
            curr = semsubject_list[cval].
                indexFitSlots[a];
            if (faculty_list[inst].unavTime[curr]
                == 0){
                done=1;
                new_pop_ptr->ind[j].xreal[i] = (
                    float) semsubject_list[cval].
                        indexFitSlots[curr];
            }
        }
    } else {
        rand_mutate(new_pop_ptr, i, j);
        nmut++;
    }
}

void rand_mutate(population *new_pop_ptr,
    int i, int j)
{
    float rnd,delta,indi,deltaq;
    float y,y1,yu,val,xy;

    y = new_pop_ptr->ind[j].xreal[i];
    y1 = lim_r[i][0];
    yu = lim_r[i][1];

```

```

    if(y > y1)
    {
        /*Calculate delta*/
        if((y-y1) < (yu-y))
            delta = (y - y1)/(yu - y1);
        else
            delta = (yu - y)/(yu-y1);

        rnd = randomperc();

        indi = 1.0/(dim +1.0);

        if(rnd <= 0.5)
        {
            xy = 1.0-delta;
            val = 2*rnd+(1-2*rnd)*(pow(xy, (dim+1))
                );
            deltaq = pow(val,indi) - 1.0;
        }
        else
        {
            xy = 1.0-delta;
            val = 2.0*(1.0-rnd)+2.0*(rnd-0.5)*(pow
                (xy, (dim+1)));
            deltaq = 1.0 - (pow(val,indi));
        }

        /*Change the value for the parent */
        y = y + deltaq * (yu-y1);
        if (y < y1) y=y1;
        if (y > yu) y=yu;
        new_pop_ptr->ind[j].xreal[i] = y;
    }
    else // y == y1
    {
        xy = randomperc();
        new_pop_ptr->ind[j].xreal[i] = xy*(yu -
            y1) + y1;
    }

    return ;
}

```

## mutation.h (final exam scheduling)

```

void test_mutate(population *new_pop_ptr);
void mutate_room(population *new_pop_ptr,
    int** roomSlot, float roomProb, int i,
    int j);
void mutate_slot(population *new_pop_ptr,
    int i, int j, int cval);
void rand_mutate(population *new_pop_ptr,
    int i, int j);

FILE *mut_test;

int *freeRooms;
int numFreeRooms;
FILE *mut_file;

void mutate_intel(population *new_pop_ptr)
{
    int i,j,k,l;
    float rnd;
    int secval, cval, subjval, prevsubjval,
        ctype, prevtype, crse, prevcrse, yr1vl
        , prevyr1vl, room_var, slot_var;
    float probMutRoom = 0, probMutSlot = 0;
    int **roomSlot;

    roomSlot = (int**) malloc(slot_n*sizeof(
        int*));
    for(i=0; i<slot_n; i++){
        roomSlot[i] = (int*) malloc(room_n*
            sizeof(int));
    }
}

```

```

for(j = 0; j < popsize; j++){
    for(i=0; i<slot_n; i++){
        for(k=0; k<room_n; k++){
            roomSlot[i][k] = 0;
        }
    }
    for(i=0; i<nvar; i=i+2){
        slot_var = (int) new_pop_ptr->ind[j].
            xreal[i];
        room_var = (int) new_pop_ptr->ind[j].
            xreal[i+1];
        for(k=0; k<slot_n; k++){
            if(overlaps[slot_var][k] > 0){
                roomSlot[k][room_var]++;
            }
        }
    }
}

for (i = 0; i < nvar; i = i+2){
    numFreeRooms = 0;
    for(k=0; k<slot_n; k++){
        for(l=0; l<room_n; l++){
            if(roomSlot[k][l]==0)
                numFreeRooms++;
        }
    }

    slot_var = (int) new_pop_ptr->ind[j].
        xreal[i];
    room_var = (int) new_pop_ptr->ind[j].
        xreal[i+1];
    fflush(stdout);

    secval = i/2;

    subjval = section_list[secval].
        subject_id;
    ctype = section_list[secval].type;
    crse = section_list[secval].course_i;
    yr1vl = section_list[secval].
        yearLevel;
    prevsubjval = section_list[secval-1].
        subject_id;
    prevtype = section_list[secval-1].type;
    prevcrse = section_list[secval-1].
        course_i;
    prevyr1vl = section_list[secval-1].
        yearLevel;

    cval = section_list[secval].
        semsubject_i;

    if(roomSlot[slot_var][room_var] > 1){
        probMutRoom = pmut_r + (1.0/nvar)*
            0.2;
        if(probMutRoom >= (1.0/nvar))
            probMutRoom = (1.0/nvar) * 0.2;
    } else
        probMutRoom = pmut_r;

    mutate_room(new_pop_ptr, roomSlot,
        probMutRoom, i, j);

    if ((subjval==prevsubjval) && (ctype==
        prevtype))
        new_pop_ptr->ind[j].xreal[i] =
            new_pop_ptr->ind[j].xreal[i-2]
            ;
    else if (((subjval==prevsubjval) && (
        ctype!=prevtype)) || ((crse==
        prevcrse) && (yr1vl==prevyr1vl))){
        if (new_pop_ptr->ind[j].xreal[i-2]<(
            slot_n-1))
            new_pop_ptr->ind[j].xreal[i] =
                new_pop_ptr->ind[j].xreal[i-2]
                + 1.00;
        else
            new_pop_ptr->ind[j].xreal[i] =
                new_pop_ptr->ind[j].xreal[i-2]
                - 1.00;
    }
}

else{
    probMutSlot = pmut_r;
    slot_var = (int) new_pop_ptr->ind[j]
        .xreal[i];
    fflush(stdout);
    if(semsubject_list[cval].slotFit[
        slot_var] != 1){
        probMutSlot = pmut_r + (1.0/nvar)*
            0.2;
        if(probMutSlot >= (1.0/nvar))
            probMutSlot = (1.0/nvar) * 0.925
            ;
    }
    rnd = randomperc();
    if(rnd<probMutSlot)
        mutate_slot(new_pop_ptr, i, j,
            cval);
}

fflush(stdout);

slot_var = new_pop_ptr->ind[j].xreal[i]
    ;
room_var = new_pop_ptr->ind[j].xreal[i]
    +1;

for(l=0; l<slot_n; l++){
    if(overlaps[l][slot_var]>1)

        roomSlot[l][room_var]++;
}
}

for(i=0; i<slot_n; i++){
    free(roomSlot[i]);
}
free(roomSlot);
free(freeRooms);
return ;
}

void mutate_room(population *new_pop_ptr,
    int **roomSlot, float roomProb, int i,
    int j){
    float rnd;
    int dval, rtype, rltype, secval, sval,
        cval, scval, rval, rmCount, curr_r,
        curr_s, tries = 0, done = 0, k, l, m,
        conf_r;
    int *rooms;
    int *roomSel;

    secval = (int) i/2;
    cval = section_list[secval].semsubject_i;
    scval = section_list[secval].subject_i;
    dval = subject_list[scval].dept_i;
    rtype = section_list[secval].type;
    rltype = section_list[secval].labtype;
    if(rtype == LECTURE){
        rmCount = department_list[dval].
            numLecRooms;
        rooms = (int*) malloc(rmCount*sizeof(int)
            );
        for(r=0; r<rmCount; r++){
            rooms[r] = department_list[dval].
                lecRooms[r];
        }
    } else {
        rmCount = department_list[dval].
            numLabRooms;
        rooms = (int*) malloc(rmCount*sizeof(int)
            );
        for(r=0; r<rmCount; r++){
            rooms[r] = department_list[dval].
                labRooms[r];
        }
    }
}

```

```

    }
}
rnd = randomperc();
if(rnd<roomProb){
    roomSel = (int*) malloc(rmCount*sizeof(
    int));
    for(k=0; k<rmCount; k++){
        roomSel[k] = 0;
        for (tries=0; tries<rmCount && done==0
        ; tries++){
            for (curr_r=(int) (randomperc() *
            rmCount - 0.01); ((curr_r<rmCount
            ) && (roomSel[curr_r]!=0) && (
            room_list[rooms[curr_r]].labtype
            !=rltype)); curr_r = (int) (
            randomperc() * rmCount)){
                roomSel[curr_r] = 1;
            }
        }
        if(rtype == LECTURE)
            curr_r = department_list[dval].
            lecRooms[curr_r];
        else
            curr_r = department_list[dval].
            labRooms[curr_r];
        for(k=0; (k<semsubject_list[cval].
        numFitSlots)&&(done==0); k++){
            curr_s = semsubject_list[cval].
            indexFitSlots[k];
            if(roomSlot[curr_s][curr_r] == 0){
                conf = 0;
                for(l=0; l<slot_n; l++){
                    if((overlaps[curr_s][l]>1)&&(
                    roomSlot[l][curr_r]>0))
                        conf = 1;
                }
                if(conf==0){
                    new_pop_ptr->ind[j].xreal[i] =
                    curr_s;
                    new_pop_ptr->ind[j].xreal[i+1] =
                    curr_r;
                    done = 1;
                }
            }
        }
    }
}
free(roomSel);
free(rooms);
if(done == 0){
    if(rtype == LECTURE){
        rmCount = department_list[0].
        numLecRooms;
        rooms = (int*) malloc(rmCount*sizeof
        (int));
        for(r=0; r<rmCount; r++){
            rooms[r] = department_list[0].
            lecRooms[r];
        }
    }
    else{
        rmCount = department_list[0].
        numLabRooms;
        rooms = (int*) malloc(rmCount*sizeof
        (int));
        for(r=0; r<rmCount; r++){
            rooms[r] = department_list[0].
            labRooms[r];
        }
    }
}
roomSel = (int*) malloc(rmCount*sizeof
(int));
tries = 0;
for(k=0; k<rmCount; k++){
    roomSel[k] = 0;
    for (tries=0; tries<rmCount && done==0
    ; tries++){
        for (curr_r=(int) (randomperc() *
        rmCount); ((curr_r<rmCount) && (
        roomSel[curr_r]!=0) && (
        room_list[rooms[curr_r]].labtype
        !=rltype)); curr_r = (int) (
        randomperc() * rmCount)){
            roomSel[curr_r] = 1;
        }
        if(rtype == LECTURE)
            curr_r = department_list[0].
            lecRooms[curr_r];
        else
            curr_r = department_list[0].
            labRooms[curr_r];
        for(k=0; (k<semsubject_list[cval].
        numFitSlots)&&(done==0); k++){
            curr_s = semsubject_list[cval].
            indexFitSlots[k];
            if(roomSlot[curr_s][curr_r] == 0
            ){
                conf = 0;
                for(l=0; l<slot_n; l++){
                    if((overlaps[curr_s][l]>1)&&
                    (roomSlot[l][curr_r]>0))
                        conf = 1;
                }
                if(conf==0){
                    new_pop_ptr->ind[j].xreal[
                    i] = (float) curr_s;
                    new_pop_ptr->ind[j].xreal[
                    i+1] = (float) curr_r;
                    done = 1;
                }
            }
        }
    }
}
free(roomSel);
free(rooms);
if(done == 0){
    rnd = randomperc();
    if((numFreeRooms>0)&&(rnd<0.85)){
        rnd = randomperc() * numFreeRooms -
        0.01;
        k = (int) rnd + 1;
        for(l=0; l<slot_n && k>0; l++){
            for(m=0; m<room_n && k>0; m++){
                if(roomSlot[l][m] == 0){
                    k--;
                    if(k==0){
                        sval = l;
                        rval = m;
                    }
                }
            }
        }
        rnd = randomperc();
        new_pop_ptr->ind[j].xreal[i] = (
        float) sval;
        new_pop_ptr->ind[j].xreal[i+1] = (
        float) rval;
    }
    else {
        rand_mutate(new_pop_ptr, i, j);
        rand_mutate(new_pop_ptr, i+1, j);
    }
}
}

```

```

    nmut++;
}
}
void mutate_slot(population *new_pop_ptr,
    int i, int j, int cval)
{
    float rnd;
    int curr;
    int countFit=0;

    countFit = semsubject_list[cval].
        numFitSlots;
    rnd = randomperc();
    if(rnd>0.8){
        rnd = randomperc();
        curr = (int) rnd * countFit - 0.01;
        new_pop_ptr->ind[j].xreal[i] = (float)
            semsubject_list[cval].indexFitSlot[
                curr] + rnd;
    } else {
        rand_mutate(new_pop_ptr, i, j);
        nmut++;
    }
}

void rand_mutate(population *new_pop_ptr,
    int i, int j)
{
    float rnd,delta,indi,deltaq;
    float y,y1,yu,val,xy;

    y = new_pop_ptr->ind[j].xreal[i];
    y1 = lim_r[i][0];
    yu = lim_r[i][1];

    if(y > y1)
    {
        /*Calculate delta*/
        if((y-y1) < (yu-y))
            delta = (y - y1)/(yu - y1);
        else
            delta = (yu - y)/(yu-y1);

        rnd = randomperc();

        indi = 1.0/(dim +1.0);

        if(rnd <= 0.5)
        {
            xy = 1.0-delta;
            val = 2*rnd+(1-2*rnd)*(pow(xy, (dim+1))
                );
            deltaq = pow(val,indi) - 1.0;
        }
        else
        {
            xy = 1.0-delta;
            val = 2.0*(1.0-rnd)+2.0*(rnd-0.5)*(pow
                (xy, (dim+1)));
            deltaq = 1.0 - (pow(val,indi));
        }

        /*Change the value for the parent */
        y = y + deltaq * (yu-y1);
        if (y < y1) y=y1;
        if (y > yu) y=yu;
        new_pop_ptr->ind[j].xreal[i] = y;
    }
    else // y == y1
    {
        xy = randomperc();
        new_pop_ptr->ind[j].xreal[i] = xy*(yu -
            y1) + y1;
    }

    return ;
}

```

## evaluation.h (class scheduling)

```

float totalRoomConflict(float *x, int xsize,
    int** schedOverlap);
float totalFacultyConflict(float *x, int
    xsize, int** schedOverlap);
float totalUnsatisfiedDemand(float *x, int
    xsize);
float totalFacultySubjectConflict(float *x,
    int xsize);
float totalIncompatibleRoomy(float *x, int
    xsize);
float totalCurriculumConflict(float *x, int
    xsize, int** schedOverlap);
float totalSchedPrefConflict(float *x, int
    xsize);
float unallowedRooms(float *x, int xsize);
float FacultyOverLoad(float *x, int xsize);
float slotTypeConflict(float *x, int xsize);

/** CONSTRAINT FUNCTIONS START **/

float totalFacultyConflict(float *x, int
    xsize, int** schedOverlap){
    int i, j, k, **facAssigned, sval, ival,
        numofsec=0, **conflict;
    float val = 0;

    facAssigned = (int**) malloc(faculty_n*
        sizeof(int*));
    conflict = (int**) malloc(faculty_n*sizeof
        (int*));
    for(i=0; i<faculty_n; i++){
        facAssigned[i] = (int*) malloc(slot_n*
            sizeof(int));
        conflict[i] = (int*) malloc(slot_n*
            sizeof(int));
        for(j=0;j<slot_n;j++){
            facAssigned[i][j] = 0;
            conflict[i][j] = 0;
        }
    }

    for(i=0; i<section_n; i++){
        sval = (int) x[i*3];
        ival = (int) x[(i*3)+2];
        facAssigned[ival][sval]++;
    }

    for(i=0; i<faculty_n; i++){
        for(j=0; j<slot_n; j++){
            if(facAssigned[i][j] > 0){
                for(k=0; k<slot_n; k++){
                    if((schedOverlap[j][k] > 0)&&(
                        facAssigned[i][k]>0))
                        conflict[i][j] += facAssigned[i]
                            [k];
                }
            }
        }
    }

    for(i=0; i<faculty_n; i++){
        for(j=0; j<slot_n; j++){
            if(conflict[i][j]>1){
                numofsec = conflict[i][j];
                val += numofsec * FACULTY_CONFLICT;
                fprintf(eval_ptr, "Faculty conflict
                    -- inst: %d, slot=%d, t: %s\n",
                        i, j, timeslots[j].code );
            }
        }
    }

    for(i=0; i<faculty_n; i++){

```

```

        free(facAssigned[i]);
        free(conflict[i]);
    }
    free(facAssigned);
    free(conflict);

    fprintf(eval_ptr, "\nPENALTY: FACULTY
        CONFLICT --- %f\n", val);
    //printf("\nPENALTY: FACULTY CONFLICT --- %f
        \n", val);
    return val;
}

float totalRoomConflict(float *x, int xsize,
    int** schedOverlap){
    int i, j, k, **roomAssigned, sval, rval,
        numofsec=0, **conflict;
    float val = 0;
    roomAssigned = (int**) malloc(room_n *
        sizeof(int*));
    conflict = (int**) malloc(room_n * sizeof(
        int*));

    for(i=0; i<room_n; i++){
        roomAssigned[i] = (int*) malloc(slot_n *
            sizeof(int));
        conflict[i] = (int*) malloc(slot_n *
            sizeof(int));
        for(j=0; j<slot_n; j++) {
            roomAssigned[i][j] = 0;
            conflict[i][j] = 0;
        }
    }

    for(i=0; i<section_n; i++){
        sval = (int) x[i*3];
        rval = (int) x[(i*3)+1];
        roomAssigned[rval][sval]++;
    }

    for(i=0; i<room_n; i++){
        for(j=0; j<slot_n; j++){
            if(roomAssigned[i][j] > 0){
                for(k=0; k<slot_n; k++){
                    if((schedOverlap[j][k] > 1)&&(
                        roomAssigned[i][k]>0)){
                        conflict[i][j] += roomAssigned[i
                            ][k];
                    }
                }
            }
        }
    }

    for(i=0; i<room_n; i++){
        for(j=0; j<slot_n; j++){
            if(conflict[i][j]>1){
                numofsec = conflict[i][j];
                val += numofsec * ROOM_CONFLICT;
                fprintf(eval_ptr, "Room conflict --
                    num: %d, room: %d, slot=%d, t: %
                    s\n", numofsec, i, j, timeslots
                    [j].code );
            }
        }
    }

    for(i=0; i<room_n; i++){
        free(roomAssigned[i]);
        free(conflict[i]);
    }

    free(roomAssigned);
    free(conflict);

    fprintf(eval_ptr, "\nPENALTY: ROOM
        CONFLICT --- %f\n", val);
    return val;
}

float slotTypeConflict(float *x, int xsize){
    float val = 0;
    int i, cval, sval;

    for(i=0; i<section_n; i++){
        sval = (int) x[i*3];
        cval = section_list[i].semsubject_i;

        if (semsubject_list[cval].slotFit[sval]
            == 0) {
            fprintf(eval_ptr, "Slot misfit --
                subject: %d, slot: %d \n", cval,
                sval);
            val += SLOT_TYPE_CONFLICT;
        }
    }
    fprintf(eval_ptr, "PENALTY: TIMESLOT TYPE
        --- %f\n\n", val);

    return val;
}

/* CONSTRAINT FUNCTIONS END HERE */

/* FITNESS FUNCTION STARTS HERE */
float totalUnsatisfiedDemand(float *x, int
    xsize) {
    int room_index, j, i, capacity, diff,
        subject_index;
    float fitness = 0;

    for(i=0; i<subject_n; i++) {
        capacity = 0;
        for(j=0; j<section_n; j++) {
            subject_index = section_list[j].
                subject_i;
            if (i == subject_index) {
                room_index = (int) x[(j*3)+1];
                capacity += room_list[room_index].
                    capacity;
            }
        }
        if (subject_list[i].demand > capacity) {
            diff = subject_list[i].demand -
                capacity;
            fitness += UNSATISFIED_DEMAND * diff;
            fprintf(eval_ptr, "CAP<DEM -- subject:
                %d, total demand: %d, total
                capacity: %d\n", i, subject_list[i
                ].demand, capacity);
        }
    }

    fprintf(eval_ptr, "PENALTY DEMAND: %f\n",
        fitness);
    return fitness;
}

float totalSchedPrefConflict(float *x, int
    xsize) {
    int ival, sval, i;
    float fitness = 0;

```

```

for(i=0; i<section_n; i++){
    sval = (int) x[i*3];
    ival = (int) x[(i*3)+2];

    if (faculty_list[ival].unavTime[sval] ==
        1) {
        fitness += SCHED_PREF_CONFLICT;
    }
}

fprintf(eval_ptr, "PENALTY SCHEDULE
PREFERENCE: %f\n", fitness);
return fitness;
}

float totalIncompatibleRoomy(float *x, int
xsize){
float fitness = 0;
int i, rval, stype, rtype, sltype, rltype;

for(i=0; i<section_n; i++){
    rval = (int) x[(i*3)+1];
    stype = section_list[i].type;
    rtype = room_list[rval].type;
    sltype = section_list[i].labtype;
    rltype = room_list[rval].labtype;

    if((stype > rtype)|| (sltype != rltype)){
        fitness += INCOMPATIBLE_ROOM;
        fprintf(eval_ptr, "Subject-Room
Incompatible-- stype: %d, sltype:
%d, rtype: %d, rltype:%d\n", stype
, sltype, rtype, rltype);
    }
}
fprintf(eval_ptr, "PENALTY: SUBJECT ROOM
COMPATIBILITY --- %f\n\n", fitness);
return fitness;
}

float unallowedRooms(float *x, int xsize){
float fitness = 0;
int i, rval, cval, cdept, rdept;

for(i=0; i<section_n; i++){
    cval = section_list[i].semsubject_i;
    rval = (int) x[(i*3)+1];
    cdept = semsubject_list[cval].dept_i;
    rdept = room_list[rval].dept_i;

    if((rdept!=CONST_ADMIN_DEPT_ID)&&(cdept!
=rdept)){
        fitness = fitness +
        ROOM_NOT_SHARED_CONFLICT;
        fprintf(eval_ptr, "Unallowed room -- r
: %d, rdept: %d, c: %d, cdept: %d
\n", rval, rdept, cval, cdept);
    }
}
fprintf(eval_ptr, "PENALTY: UNALLOWED ROOM
FOR SUBJECT --- %f\n\n", fitness);
return fitness;
}

float totalCurriculumConflict(float *x, int
xsize, int** schedOverlap){
int i, j, k, l, **sectionAssigned, **
conflict, sval, cval, yval, dval,
numofsec = 0;
float val = 0;

sectionAssigned = (int***)malloc(course_n*
sizeof(int**));
conflict = (int***)malloc(course_n*sizeof(
int**));
for(k=0; k<course_n; k++){
    sectionAssigned[k] = (int**) malloc(
    MAX_YEAR_LEVEL*sizeof(int**));
    conflict[k] = (int**) malloc(
    MAX_YEAR_LEVEL*sizeof(int**));
    for(i=0; i<MAX_YEAR_LEVEL; i++){
        sectionAssigned[k][i] = (int*) malloc(
        slot_n*sizeof(int));
        conflict[k][i] = (int*) malloc(slot_n*
sizeof(int));
        for(j=0; j<slot_n; j++){
            sectionAssigned[k][i][j] = 0;
            conflict[k][i][j] = 0;
        }
    }
}

for(i=0; i<section_n; i++){
    sval = (int) x[i*3];
    cval = section_list[i].subject_i;
    yval = section_list[i].yearLevel - 1;
    dval = section_list[cval].course_i;
    if(yval>=0)
        sectionAssigned[dval][yval][sval]++;
}

for(l=0; l<course_n; l++){
    for(i=0; i<MAX_YEAR_LEVEL; i++){
        for(j=0; j<slot_n; j++){
            if(sectionAssigned[l][i][j] > 0){
                for(k=0; k<slot_n; k++){
                    if(schedOverlap[j][k] > 0)
                        conflict[l][i][k] +=
                        sectionAssigned[l][i][j];
                }
            }
        }
    }
}

for(l=0; l<course_n; l++){
    for(i=0; i<MAX_YEAR_LEVEL; i++){
        for(j=0; j<slot_n; j++){
            if(conflict[l][i][j]>1){
                numofsec = conflict[l][i][j];
                val += (numofsec *
                CURRICULUM_CONFLICT);
                fprintf(eval_ptr, "Year Offered
conflict due to timeslot
overlap -- num: %d, course: %d
, yr: %d, slot: %s\n",
                numofsec, l, i, timeslots[j].
                code);
            }
        }
    }
}

for(l=0; l<course_n; l++){
    for(i=0; i<MAX_YEAR_LEVEL; i++){
        free(sectionAssigned[l][i]);
        free(conflict[l][i]);
    }
    free(sectionAssigned[l]);
    free(conflict[l]);
}
free(sectionAssigned);
free(conflict);
}

```

```

    fprintf(eval_ptr, "\nPENALTY: YEAR OFFERED
    CONFLICT --- %f\n", val);
    return val;
}

float FacultyOverLoad(float *x, int xsize){
    int sec_index, inst_index, ival, cval,
        units, diff, i;
    float fitness = 0;
    int* instLoad;

    instLoad = (int*) malloc(faculty_n*
        sizeof(int));
    for(i=0; i<faculty_n; i++){
        instLoad[i] = 0;
    }

    for(sec_index = 0; sec_index<section_n;
        sec_index++){
        inst_index = (sec_index*3) + 2;
        ival = (int) x[inst_index];
        cval = section_list[sec_index].
            subject_i;
        units = semsubject_list[cval].units;
        instLoad[ival] = instLoad[ival] +
            units;
    }

    for(i=0; i<faculty_n; i++){
        diff = instLoad[i] - faculty_list[i]
            .load;
        if(diff>0){
            fitness += ((float)diff*
                OVERLOAD_INSTRUCTOR);
            fprintf(eval_ptr, "Faculty
                overload-- ival: %d, load: %
                d, currLoad: %d\n", ival,
                faculty_list[i].load,
                instLoad[i]);
        }
    }

    fprintf(eval_ptr, "PENALTY FACULTY LOAD:
        %f\n", fitness);
    return fitness;
}

float totalFacultySubjectConflict(float *x,
    int xsize){
    int k, sec_index, inst_index, ival, cval,
        inst_p;
    float fitness = section_n;

    for(sec_index = 0; sec_index<section_n;
        sec_index++){
        inst_index = (sec_index*3) + 2;
        ival = (int) x[inst_index];
        cval = section_list[sec_index].subject_i
            ;
        inst_p=0;

        for(k = 0; (k<subject_list[cval].
            numofFitInsts && inst_p==0); k++){
            if (subject_list[cval].fac[k]==ival)
                inst_p=1;
        }

        if(inst_p == 0){
            fitness += FACULTY_SUBJECT_CONFLICT;
            fprintf(eval_ptr, "Incompatible
                faculty and subject-- i: %d, c: %d
                , s: %d\n", ival, cval, sec_index)
                ;
        }
    }

    fprintf(eval_ptr, "PENALTY SUBJECT FACULTY
        : %f\n", fitness);
    fflush(stdout);
    return fitness;
}

```

## evaluation.h (final exam scheduling)

```

float totalRoomConflict(float *x, int xsize,
    int** schedOverlap);
float totalUnsatisfiedDemand(float *x, int
    xsize);
float totalIncompatibleRoomy(float *x, int
    xsize);
float totalCurriculumConflict(float *x, int
    xsize, int** schedOverlap);
float unallowedRooms(float *x, int xsize);
float evaluateSlotType(float *x, int xsize
    );
float getSubjectClash (float *x, int xsize,
    int** schedOverlap);

/** CONSTRAINT FUNCTIONS START */
float totalRoomConflict(float *x, int xsize,
    int** schedOverlap){
    int i, j, k, **roomAssigned, sval, rval,
        numofsec=0, **conflict;
    float val = 0;
    roomAssigned = (int**) malloc(room_n *
        sizeof(int*));
    conflict = (int**) malloc(room_n * sizeof(
        int*));

    for(i=0; i<room_n; i++){
        roomAssigned[i] = (int*) malloc(slot_n *
            sizeof(int));
        conflict[i] = (int*) malloc(slot_n *
            sizeof(int));
        for(j=0; j<slot_n; j++){
            roomAssigned[i][j] = 0;
            conflict[i][j] = 0;
        }
    }

    for(i=0; i<section_n; i++){
        sval = (int) x[i*2];
        rval = (int) x[(i*2)+1];
        roomAssigned[rval][sval]++;
    }

    for(i=0; i<room_n; i++){
        for(j=0; j<slot_n; j++){
            if(roomAssigned[i][j] > 0){
                for(k=0; k<slot_n; k++){
                    if((schedOverlap[j][k] > 1)&&(
                        roomAssigned[i][k]>0)){
                        conflict[i][j] += roomAssigned[i
                            ][k];
                    }
                }
            }
        }
    }

    for(i=0; i<room_n; i++){
        for(j=0; j<slot_n; j++){
            if(conflict[i][j]>1){
                numofsec = conflict[i][j];
                val += numofsec * ROOM_CONFLICT;
                fprintf(eval_ptr, "Room conflict --
                    num: %d, room: %d, slot=%d, t: %
                    s\n", numofsec, i, j, timeslots
                    [j].code );
            }
        }
    }

    for(i=0; i<room_n; i++){
        free(roomAssigned[i]);
        free(conflict[i]);
    }
}

```



```

free(roomAssigned);
free(conflict);

fprintf(eval_ptr, "\nPENALTY: ROOM
CONFLICT --- %f\n", val);
return val;
}

float getSubjectClash (float *x, int xsize,
int** schedOverlap){
int i, j, k, **secAssigned, sval, cval,
tval, **conflict, numofsec=0;
float val = 0;

secAssigned = (int**) malloc(subject_n*
sizeof(int));
conflict = (int**) malloc(subject_n*sizeof
(int));
for(i=0; i<subject_nn; i++){
secAssigned[i] = (int*) malloc(slot_n*
sizeof(int));
conflict[i] = (int*) malloc(slot_n*
sizeof(int));
for(j=0; j<slot_n; j++) {
secAssigned[i][j] = 0;
conflict[i][j] = 0;
}
}

for(i=0; i<section_n; i++){
sval = (int) x[i*2];
cval = section_list[i].subject_i;
tval = section_list[i].type;

if ((section_list[i-1].subject_i==cval)
&& (section_list[i-1].type != tval))
secAssigned[cval][sval]++;
}

for(i=0; i<subject_nn; i++){
for(j=0; j<slot_n; j++){
if(secAssigned[i][j] > 0){
for(k=0; k<slot_n; k++){
if((schedOverlap[j][k] > 0)&&
(secAssigned[i][k]>0))
conflict[i][j] = conflict[i][j]
+ secAssigned[i][k];
}
}
}

for(i=0; i<subject_nn; i++){
for(j=0; j<slot_n; j++){
if(conflict[i][j]>1){
numofsec = conflict[i][j];
val += numofsec *
PENALTY_SUBJECT_CONFLICT;
fprintf(eval_ptr, "Course conflict -
- num: %d, c: %d, j=%d, t: %s\n"
, numofsec, i, j, timeslots[j].
code );
}
}
}

for(i=0; i<subject_nn; i++){
free(secAssigned[i]);
free(conflict[i]);
}
free(secAssigned);
free(conflict);

fprintf(eval_ptr, "\nPENALTY: SUBJECT
CONFLICT --- %f\n", val);
return val;
}

float slotTypeConflict (float *x, int xsize){
float val = 0;
int i, a, cval, sval, c_slot_type,
slot_type, sec_type;
int smtg, cmtg;
float slec, slab, clec, clab;

for(i=0; i<section_n; i++){
sval = (int) x[i*2];
cval = section_list[i].semsubject_i;

if (semsubject_list[cval].slotFit[sval]
== 0) {
fprintf(eval_ptr, "Slot misfit --
subject: %d, slot: %d \n", cval,
sval);
val += SLOT_TYPE_CONFLICT;
}
}

fprintf(eval_ptr, "PENALTY: TIMESLOT TYPE
--- %f\n\n", val);
//printf("PENALTY: TIMESLOT TYPE --- %f\n\n"
, val);

return val;
}

/* CONSTRAINT FUNCTIONS END HERE */

/* FITNESS FUNCTION STARTS HERE */

float totalUnsatisfiedDemand (float *x, int
xsize) {
//total demand for course <= total
capacity of rooms allotted
int room_index, j, i, capacity, diff,
subject_index;
float fitness = 0;

for(i=0; i<subject_n; i++) {
capacity = 0;
for(j=0; j<section_n; j++) {
subject_index = section_list[j].
subject_i;
if (i == subject_index) {
room_index = (int) x[(j*2)+1];
capacity += room_list[room_index].
capacity;
}
}
if (subject_list[i].demand > capacity) {
diff = subject_list[i].demand -
capacity;
fitness += UNSATISFIED_DEMAND * diff;
fprintf(eval_ptr, "CAP<DEM -- subject:
%d, total demand: %d, total
capacity: %d\n", i, subject_list[i].
demand, capacity);
}
}

fprintf(eval_ptr, "PENALTY DEMAND: %f\n",
fitness);
}

```

```

    return fitness;
}

float totalIncompatibleRoomy(float *x, int
    xsize){
    float fitness = 0;
    int i, rval, stype, rtype, sltype, rltype;

    for(i=0; i<section_n; i++){
        rval = (int) x[(i*2)+1];
        stype = section_list[i].type;
        rtype = room_list[rval].type;
        sltype = section_list[i].labtype;
        rltype = room_list[rval].labtype;

        // Lecture class type = 1 -- can be in
        // any type of room
        if((stype > rtype)|| (sltype != rltype)){
            fitness += INCOMPATIBLE_ROOM;
            fprintf(eval_ptr, "Subject-Room
                Incompatible-- stype: %d, sltype:
                %d, rtype: %d, rltype:%d\n", stype
                , sltype, rtype, rltype);
        }
    }
    fprintf(eval_ptr, "PENALTY: SUBJECT ROOM
        COMPATIBILITY --- %f\n\n", fitness);
    // //printf("PENALTY: SUBJECT ROOM
        COMPATIBILITY --- %f\n\n", fitness);

    return fitness;
}

float unallowedRooms(float *x, int xsize){
    float fitness = 0;
    int i, rval, cval, cdept, rdept;

    for(i=0; i<section_n; i++){
        cval = section_list[i].semsubject_i;
        rval = (int) x[(i*2)+1];
        cdept = semsubject_list[cval].dept_i;
        rdept = room_list[rval].dept_i;

        if((rdept!=CONST_ADMIN_DEPT_ID)&&(cdept!
            =rdept)){
            fitness = fitness +
                ROOM_NOT_SHARED_CONFLICT;
            fprintf(eval_ptr, "Unallowed room -- r
                : %d, rdept: %d, c: %d, cdept: %d
                \n", rval, rdept, cval, cdept);
        }
    }
    fprintf(eval_ptr, "PENALTY: UNALLOWED ROOM
        FOR SUBJECT --- %f\n\n", fitness);
    return fitness;
}

float totalCurriculumConflict(float *x, int
    xsize, int** schedOverlap){
    int i, j, k, l, **sectionAssigned, **
        conflict, sval, secval, cval, yval,
        dval, numofsec = 0;
    float val = 0;
    FILE *yoFile;

    sectionAssigned = (int***)malloc(course_n*
        sizeof(int**));
    conflict = (int***)malloc(course_n*sizeof(
        int**));
    for(k=0; k<course_n; k++){
        sectionAssigned[k] = (int**) malloc(
            MAX_YEAR_LEVEL*sizeof(int*));
        conflict[k] = (int**) malloc(
            MAX_YEAR_LEVEL*sizeof(int*));
        for(i=0; i<MAX_YEAR_LEVEL; i++){
            sectionAssigned[k][i] = (int*) malloc(
                slot_n*sizeof(int));
            conflict[k][i] = (int*) malloc(slot_n*
                sizeof(int));
            for(j=0; j<slot_n; j++){
                sectionAssigned[k][i][j] = 0;
                conflict[k][i][j] = 0;
            }
        }
    }

    for(i=0; i<section_n; i++){
        sval = (int) x[i*2];
        cval = section_list[i].subject_i;
        yval = section_list[i].yearLevel - 1;
        dval = section_list[cval].course_i;
        if(yval>=0)
            sectionAssigned[dval][yval][sval]++;
    }

    for(l=0; l<course_n; l++){
        for(i=0; i<MAX_YEAR_LEVEL; i++){
            for(j=0; j<slot_n; j++){
                if(sectionAssigned[l][i][j] > 0){
                    for(k=0; k<slot_n; k++){
                        if(schedOverlap[j][k] > 0)
                            conflict[l][i][k] +=
                                sectionAssigned[l][i][j];
                    }
                }
            }
        }
    }

    for(l=0; l<course_n; l++){
        for(i=0; i<MAX_YEAR_LEVEL; i++){
            for(j=0; j<slot_n; j++){
                if(conflict[l][i][j]>1){
                    numofsec = conflict[l][i][j];
                    val += (numofsec *
                        CURRICULUM_CONFLICT);
                    fprintf(eval_ptr, "Year Offered
                        conflict due to timeslot
                        overlap -- num: %d, course: %d
                        , yr: %d, slot: %s\n",
                        numofsec, l, i, timeslots[j].
                        code);
                }
            }
        }
    }

    for(l=0; l<course_n; l++){
        for(i=0; i<MAX_YEAR_LEVEL; i++){
            free(sectionAssigned[l][i]);
            free(conflict[l][i]);
        }
        free(sectionAssigned[l]);
        free(conflict[l]);
    }
    free(sectionAssigned);
    free(conflict);

    fprintf(eval_ptr, "\nPENALTY: YEAR OFFERED
        CONFLICT --- %f\n", val);
    return val;
}

```

## CHAPTER XI

### ACKNOWLEDGEMENT

With love and passion, this paper is dedicated to my family who has always been there supporting me in every endeavors I've been through. The completion of this work would have not been achieved if not for their love, support, guidance and encouragement.

To my classmates and friends who has always been praying for me, who has always been there inspiring me to do better, thank you all!

Many, many thanks to the examiners of this SP, and most specially to my adviser, Ma'am Sheila Magboo. You all made me do better.

Most of all, I thank God, for his never ending support, for the strength He's given me, for the talent He's bestowed upon me.

This SP was accomplished through all the help and support of my family and friends and through my sacrifices. This is a fruit of love, passion, inspiration, support and encouragement.