A TIMETABLING PROBLEM: CONSTRAINT AND MATHEMATICAL PROGRAMMING APPROACHES

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Ahmet Reha Botsalı

June 2000

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Ömer S. Benli (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Osman Oğuz

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assist. Prof. Dr. Cemal Akyel

Approved for the Institute of Engineering and Sciences:

[illegible]

Prof. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

A TIMETABLING PROBLEM: CONSTRAINT AND MATHEMATICAL PROGRAMMING APPROACHES

Ahmet Reha Botsalı
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. Dr. Ömer S. Benli
June 2000

Constraint programming is a relatively new approach for solving combinatorial optimization problems. This approach is especially effective for large scale scheduling problems with side conditions. University course scheduling problem is one of the hard problems in combinatorial optimization. Furthermore, the specific requirements of each institution make it very difficult to suggest a generalized model and a solution algorithm for this problem. The purpose of this study is to design a system for scheduling courses at Bilkent University. This system utilizes both constraint programming and mathematical programming techniques. The problem is solved in three stages. The first two stages, in tandem, generate a course schedule using constraint programming techniques, and in the last stage classrooms are assigned to courses by means of a mixed integer programming model. The proposed system is validated by experimental runs using Bilkent University course offerings and classroom data from past semesters.

Keywords: Timetabling, university course scheduling, constraint programming, mathematical programming.

ÖZET

BİR DERS ÇİZELGELEME PROBLEMİ: KISIT VE MATEMATİKSEL PROGRAMLAMA UYGULAMASI

Ahmet Reha Botsalı
Endüstri Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Ömer S. Benli
Haziran 2000

Kısıt programlaması kombinatoryal optimizasyon problemlerinin çözümünde kullanılan oldukça yeni bir yöntemdir. Bu yöntem, özellikle yan kısıtları olan büyük ölçekli çizelgeleme problemlerinde çok etkin olmaktadır. Üniversite ders çizelgelemesi problemi kombinatoryal optimizasyon problemlerinin en zorlarından biridir. Ek olarak, her kurumun özel gereksinimleri, bu problem için genel bir model ve çözüm algoritması önermeyi olanaksız kılmaktadır. Bu çalışmanın amacı, Bilkent Üniversitesi için ders çizelgelemesi oluşturan bir sistem tasarlamaktır. Bu sistem hem kısıt programlaması hem de matematiksel programlama tekniklerinden yararlanmaktadır. Problem, üç aşamada çözülmektedir. İlk iki aşamada kısıt programlaması teknikleri kullanılarak bir ders çizelgesi oluşturulmakta, son aşamada ise sınıflar derslere tam sayı programlaması kullanılarak atanmaktadır. Önerilen sistemin uygulanabilirliği, Bilkent Üniversitesi'nin geçmiş dönemlere ait verileri kullanılarak gösterilmiştir.

Anahtar Kelimeler: Üniversite ders çizelgelemesi, kısıt programlaması, matematiksel programlama

ACKNOWLEDGEMENT

I would like to use this opportunity to express my deep gratitude to my supervisor Assoc. Prof. Dr. Ömer S. Benli for his patience, guidance and invaluable encouragement throughout the development of this thesis.

I would like to thank Assoc. Prof. Dr. Osman Oğuz and Assist. Prof. Dr. Cemal Akyel for reading and commenting on the thesis.

I express my special thanks to my family for their constant support, patience and sincere love.

Finally, many thanks to all of my close friends.

Contents

List of Figures

List of Tables

To my family ...

Chapter 1

Introduction

The purpose of this study is to design a system for scheduling courses at Bilkent University. The system utilizes both constraint programming and mathematical programming techniques. In the next section, timetabling problem is discussed in general and, then in the following sections, the course scheduling problem at Bilkent University is described and an overview of the approaches to timetabling problem is presented.

1.1 Timetabling Problems

Timetabling is one of the computationally difficult problems in scheduling. In timetabling, the aim is to find suitable time slots for a number of tasks that require limited resources. Depending on the nature of the problem, the constraints in the problem can vary and there may be many different objectives. For example, in some cases the objective may be to minimize the length of the total time period over which the tasks are to be scheduled, in other cases the objective may be to find a feasible solution subject to a fixed total time period and several other constraints. Yet in others the objective can be to find a solution in which least number of constraints are violated.

Timetabling problems can arise in many different settings, but generally it refers to the timetabling at educational institutions. The significance of this problem is mainly due to the difficulty of constructing a feasible timetable that satisfies the preferences of the administration, the instructors, and the students. In certain cases, it may be extremely difficult even to find a single feasible solution.

It is not possible to formulate a general model that is applicable for all cases, since every educational institution has its own special constraints and objectives. For example, for a secondary school, there should not be any gap between the class meetings, on the other hand this is allowed, and in some cases encouraged, in a university.

Timetabling problems can be grouped into two types: Examination and course scheduling problems. Examination scheduling problems deal with assigning the examinations over an examination period subject to several constraints. The objective can be assignment of the examinations to a minimum number of periods without any conflict. In other problems, the number of periods are fixed and the objective is to optimize a preference function that is based on the preferences of the administration, the instructors, and the students. Some of these preferences may be:

- Maximizing the time interval between the two consecutive examinations of a student.
- Scheduling the examinations with large number of students in earlier time slots, to allow more time for grading.

In course scheduling, the time period is fixed and, in general, it is one week. The objective is to find a course schedule that is feasible with respect to a number of constraints. Course scheduling and examination scheduling problems have both similar and differing characteristics. For example, in both problems a student cannot take more than one examination or attend more than one class meeting at a time. On the other hand, in examination scheduling problems there may not have a fixed time period, however all course scheduling problems are for fixed time periods.

Regardless of the differences among problem types, similar solution approaches are used in all. In this thesis, the focus is on the course scheduling problems. In the next section the characteristics of the problem are discussed.

1.2 Characteristics of Course Scheduling Problems

In course scheduling problems, some constraints are case specific, but there are some constraints which should be satisfied in every course scheduling problem. These are called *hard constraints.* For example,

- The class meetings of two courses with the same student enrollment cannot be assigned to the same time slot.
- An instructor cannot teach more than one class meeting at the same time slot.
- The assigned number of class meetings that are scheduled for the same time slot cannot exceed the number of available classrooms.
- All class meetings should be assigned to a time slot.

Hard constraints cannot be violated, since violation of any one of them results in an infeasible timetable. On the other hand, there are *soft constraints.* It is not desirable to violate a soft constraint, but if it is violated, the timetable is still feasible, but not as good as a timetable in which that constraint is not violated. Among such soft constraints are:

- An instructor may not want to teach more than, say, four class meetings in a day.
- It is undesirable to have a class meeting during the lunch hour.
- Students may not want to have class meetings in early and late time slots.
- There should be at least one day between the two class meetings of a course.

As mentioned earlier, timetabling problem is computationally very difficult. The basic timetabling problem is reducible to *graph coloring problem.* Since in 1972, Karp [22] showed that the graph coloring problem is NP-Complete, it consequently follows that timetabling problem is NP-Complete. In addition to this, the varied nature of constraints for each institution makes timetabling problem even more complicated. The next section discusses these specific constraints in the case of Bilkent University.

1.3 The Course Scheduling Problem at Bilkent University

Bilkent is a large university having two campuses, several schools and around ten thousand students. The course scheduling problem is becoming considerably more demanding each year and thus, there is a need to use advanced techniques for course timetabling.

At Bilkent, the classes can meet during the weekdays from 08:40 to 17:30. Each day there are nine time slots in which a class meeting can be scheduled. Each meeting lasts 50 minutes with a ten minute break in between classes. So the class meetings start at {8:40, 9:40,...,16:40} and end at {9:30, 10:30,...,17:30}. A course that requires two time slots is called a 2-hour course. In the similar manner, there are 3-, 4-, and 5-hour courses which requires three, four and five time slots, respectively.

Other than the 2-hour courses, all other courses are to be scheduled on two separate days, that is:

3-hour courses One class meeting requires one time slot and the other one requires two consecutive time slots.

4-hour courses Both class meetings require two consecutive time slots.

5-hour courses One class meeting requires two consecutive time slots and the other one requires three consecutive time slots.

A course may have several sections depending upon the student enrollment in the course. If a course is to meet on two separate days, then the class meetings cannot be assigned to two consecutive days. For example, if the first class meeting is scheduled on Monday, the second class meeting can be scheduled at the earliest on Wednesday.

Each section of a course may have different class sizes ranging from 10 to 65 students. The variety in section sizes requires scheduling of class meetings in an appropriate classroom which has sufficient capacity to accommodate all students. Currently, the classrooms can be grouped into four types based on their capacity:

Type-1 classrooms Maximum capacity of 25 students.

Type-2 classrooms Maximum capacity of 30 students.

Type-3 classrooms Maximum capacity of 40 students.

Type-4 classrooms Maximum capacity of 65 students.

Thus, as an example, if a section has 35 students, then its class meetings can take place in a *type-3* or a *type-4* classroom.

Although in some other institutions instructor and course section assignments may be done by timetabling programs [4], the number of sections for each course and the instructor of each course section are known in advance at Bilkent. Further more, each course section is reserved for a specific student group. That is, for each course section, the students who can enroll are known in advance. This makes it possible to compute the number of required sections for each course prior to the course scheduling process at each semester.

Summing-up, following eight constraints are needed to comply with the conditions and requirements of Bilkent University. All these constraints are treated as if they are hard constraints at Bilkent.

1. All class meetings will be assigned to a feasible time slot.
2. Two class meetings belonging to the same course section cannot be scheduled on two consecutive days .
3. All class meetings should be assigned to an appropriate size classroom.
4. There are limited number of classrooms in each classroom type.
5. The course sections assigned for specific student groups cannot overlap.
6. Sections of the courses that are taught by the same instructor cannot overlap.
7. An instructor cannot teach more than a fixed number of class meeting hours per day.
8. A student can attend at most eight hours of class meetings per day.

1.4 Review of Approaches to Timetabling Problem

In this thesis, *constraint programming* (CP) and *mathematical programming* (MP) techniques are used for solving the course scheduling problem. There are other solution approaches besides these for course scheduling problem. Usually, it is not possible to decide beforehand which solution approach is the best. Since the constraints vary from one institution to another, a solution procedure that performs well in one case may perform very poorly in another. Sometimes combination of several approaches can give better results. The following is a brief review of approaches to modeling and solution of timetabling problems.

Graph coloring heuristics are one of the earliest approaches used to solve timetabling problems. The graph coloring problem can be described as finding a coloring for the vertices of a graph in such a way that no two vertices have the same color if they have an edge connecting them. In a graph coloring model of a timetabling problem, courses are represented by vertices. For any course pair, if there exists a constraint which states that they cannot be scheduled simultaneously, then there exists an edge in between the two vertices representing those courses. Such constraints may exist if the same instructor is teaching both courses ("instructor clash") or same set of students may have to enroll in both courses ("curriculum clash").

As a simple example, suppose there are five courses A, B, C, D, E. Courses A and B are taught by the same instructor, and D and E are taught by another. Suppose further that same students enrolled in courses B, C, D. The corresponding graph is shown in Figure 1.1.

Since the graph coloring problem is NP-Complete, heuristic procedures are needed for its effective solution. Since early 1960s, different graph coloring heuristics are suggested for solving timetabling problem. As reported in Weare

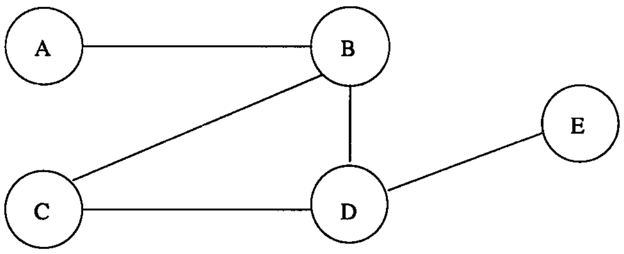


Figure 1.1: A Simple Graph Representation for Course Scheduling

[27], Broder [7] and Cole [12] presented different graph coloring heuristics to solve the same timetabling problem. Broder used a sequential algorithm that schedules exams on the first available time period and this idea was adapted to course scheduling case by scheduling the courses at the period with least number of student conflicts, where the number of periods is restricted. On the other hand, Cole's algorithm was selecting the exams that did not have conflict on the current time period instead of selecting the exams randomly. These are among the first studies for solving the timetabling problem by graph coloring heuristics. Later, there have been other studies that related graph coloring techniques to timetabling problem. A survey of the graph coloring techniques used in timetabling can be found in [10].

Today, graph coloring still attracts interest. For example, Burke et al. [9] discuss the role of graph coloring techniques in automatic timetable generation. Dowsland [16] represents timetabling problem as a graph coloring problem and assigns weights to the edges according to the importance of the conflict represented by that edge. The problem, then reduces to finding a coloring in which the total weight of edges that connects the vertices of the same color is minimized.

Mathematical programming is another approach that is used for solving timetabling problems. Let the index sets be,

$j = 1, 2, \ldots, M$ for the courses, and

$t = 1, 2, \dots, T$ for the time slots.

The decision variables are

$$x_{ij} = \begin{cases} 1, & \text{if course } j = 1, 2, \dots, M \text{ is assigned to time slot } i = 1, 2, \dots, T; \\ 0, & \text{otherwise.} \end{cases}$$

Suppose there are n classrooms and let P denote the set of course pairs that cannot be scheduled at the same time slot, then a simple timetabling problem can be formulated as minimizing or maximizing an objective function subject to the following constraints:

$$\begin{array}{lll} (1) & \sum_{i=1}^{T} x_{ij} = 1, & \forall\ j \in \{1, 2, \dots, M\} \\ (2) & \sum_{j=1}^{M} x_{ij} \leq n, & \forall i \in \{1, 2, \dots, T\} \\ (3) & \sum_{i=1}^{T} (x_{ij} + x_{ik}) \leq 1, & \forall\, (j, k) \in P. \end{array}$$

In this model, the constraint set (1) ensures that all courses assigned to a time slot. The constraint set (2) ensures that for all time slots, there cannot be more courses than the available number of classrooms and the constraint set (3) ensures that if two courses j and k cannot be scheduled at the same time because of a clash constraint, then at a time slot, at most one of these courses can be scheduled. This clash constraints may be due to a number of reasons, such as:

- Same students enroll in courses j and k.
- Courses j and k are taught by the same instructor.
- Courses j and k share a single resource, such as a special equipment or laboratory.

The objective function is not needed when only a feasible solution is needed. Badri et al. [4] discuss the objective functions that combine several preference functions. Generally, these preferences have different weights according to their importance and the objective is maximizing (or minimizing) the total weight.

The above model is the simplest representation of a timetabling problem, naturally the models representing real applications are more complicated. Depending on the institution, the requirements can change, consequently it is necessary to append other constraints. In a timetabling problem, if there are T periods and M courses, then the number of binary variables, x_{ij}'s, will be equal to $M \times T$. Mathematical programming approach can solve fairly small size problems, but when the number of courses increases, the resulting mathematical programming model becomes computationally intractable. It may be necessary to intervene the solution process using relaxation as Tripathy [24] did in his study. Another approach may be to find ways to decrease the number of variables. For example, if two courses should be scheduled on the same time slot, then they can be considered as a single variable as in [25].

Simulated Annealing (SA) and **Tabu Search** (TS) are relatively newer approaches for solving timetabling problem compared to graph coloring and MP techniques. They are iterative improvement algorithms that are designed to search for the optimal solution without being trapped at a local optimum. In these algorithms, an initial solution is iteratively modified. These iterative modifications on the solutions generally cause improvement on the objective function value, but in order not to get stuck at a local optimum point, sometimes the modifications can deteriorate the objective function value. For an introductory discussion of these approaches, see Appendix A.

Hertz [19] proposed a model that uses TS to find a course schedule where the length of class meetings is not known in advance. Different than academic timetabling problems, Wright [28] investigates the timetabling problem for sports events. He tests three techniques experimentally and develops a computerized system that uses a form of TS for timetabling. He also claims that this approach may have applicability in large, complex, multi-objective combinatorial problems.

Johnson [21] gives an MP model for examination timetabling and he suggests an approach that uses simulated annealing algorithm instead of solving the MP model with a large number of binary variables.

Use of **Genetic Algorithms** (GA) is yet another heuristic approach for producing acceptable timetables. In order to apply GA, initially there should be a set of solutions and a fitness (or evaluation) function that is used to measure the quality of the solutions. GA is an iterative algorithm and the aim is to obtain good solutions with respect to the preference function by modifying or combining the solutions in the solution set of the previous iteration. Since these algorithms mimick the natural selection process, they are called genetic algorithms. An introductory discussion of this approach is given in Appendix B. There is a large number of studies that use GA to solve timetabling problems. Burke et al. [8] develop an automated timetabling system based on GA. Abramson and Abela [2] solve a timetabling problem by using GA. However, they include a parallel mating scheme and use shared processors to solve the problem in less time compared to other studies that use GA with sequential breeding. Corne et al. [13] give an overview of GA approaches for solving timetabling problems.

Another solution approach for timetabling problem is considering the problem as a **network flow model**. Chahal and Werra [11] used this idea to construct an interactive system for timetabling. They validate this system for a small size timetable problem of an adult education school in Geneva. Network flow models for timetabling problem and a discussion of other methods related to timetabling problem are discussed in Werra's review paper [15].

Constraint programming (CP) is a relatively new technique that is effective in solving large scale combinatorial problems. In CP approach, the timetabling problem is modeled as *constraint satisfaction problem* (CSP). A CSP consists of three main elements:

1. Variables,

2. Set of values that each variable can take (the "variable domain"),
3. Constraints restricting the values that the variables can take simultaneously.

In timetabling problems solved by CP, generally class meetings are defined as the variables and time slots are the domains of the variables. The typical timetabling constraints form the constraint set. The problem is to find values for all the variables that satisfy the constraints. In its optimization version, the problem is to find the value assignment of variables that minimizes or maximizes an objective function.

There are various reasons for CP to be an attractive candidate approach for solving timetabling problems. For example, in these problems, there is a large number of constraints that restrict the values that two variables can take simultaneously. CP makes use of this fact in *constraint propagation.* Constraint propagation is a mechanism in the solution process that generates new constraints on the values that the variables can take depending on the current set of constraints. *Domain reduction* is another technique that makes CP powerful. When a variable is assigned to a value in its domain, by domain reduction, the value domains of the other variables are reduced.

Such features of CP provide pruning of the search space while assigning values to the variables. This is important for a timetabling problem, because large number of courses and time slots cause a very large search space which results in excessive computation. Besides pruning the search space, the ability to define a *search strategy* in CP makes it more attractive. An introductory discussion of CP can be found in Appendix C.

Frangouli et al. [17] use an instance of the constraint logic programming class of languages, the *Eclipse System* [31], to construct a timetabling system called UTSE at University of Athens. They also provide a user interface that allows user to specify the features of the timetable such as distance of class meetings, room utilization, etc. Henz and Würtz [18] solve the timetabling problem of a German university by using *Oz* which is a concurrent constraint language providing for functional, object oriented and constraint programming [23] . (Recently *Mozart* [32] has replaced Oz). Azevedo and Barahona [3] first give a MP model for the timetabling problem for the Faculty of Science and Technology of the New University of Lisbon and show that the problem has a large number of binary variables to find a solution. Then, instead of MP model, they suggest a CP model that is solved by using system *DOMLOG* which is a constraint logic programming system for finite domain variables. Deris et al. [14] again used a CP system to solve a timetabling problem involving 536 courses, 45 time slots and 21 classrooms. However, as different from other studies, they implement CP using their own C++ code instead of using a general CP software.

1.5 Outline of the Thesis

In the previous sections, after introducing the timetabling problem and discussing the characteristics of course scheduling problem, the course scheduling problem at Bilkent University is presented in detail; followed by a brief overview of the previous approaches to this problem, emphasizing the major contributions.

The rest of the thesis is organized as follows. Chapter 2 develops the main constraint programming and mathematical programming models that are used in the course scheduling system that is presented in Chapter 3. The system is validated using the data of past course offerings at Bilkent University. This and the final conclusions are given in Chapter 4.

Chapter 2

Modeling of the Timetabling Problem

Since course scheduling problem is NP-Complete, the increase in the size of the problem makes it computationally very difficult to find a solution in a reasonable time. To overcome this problem, the constraint set is decomposed into subsets. By this way, the overall course scheduling problem can be represented as a union of three subproblems (see Figure 2.1):

1. Allocation of class meetings to days,
2. Construction of course schedule for each day,
3. Assigning classrooms to class meetings.

Note that, in its current version, there is no objective function to optimize, hence the above sequential approach does not cause any degradation in the quality of the resulting solution. The course schedule is generated in three stages. In the first two stages, the respective problem is modeled as a constraint program. In

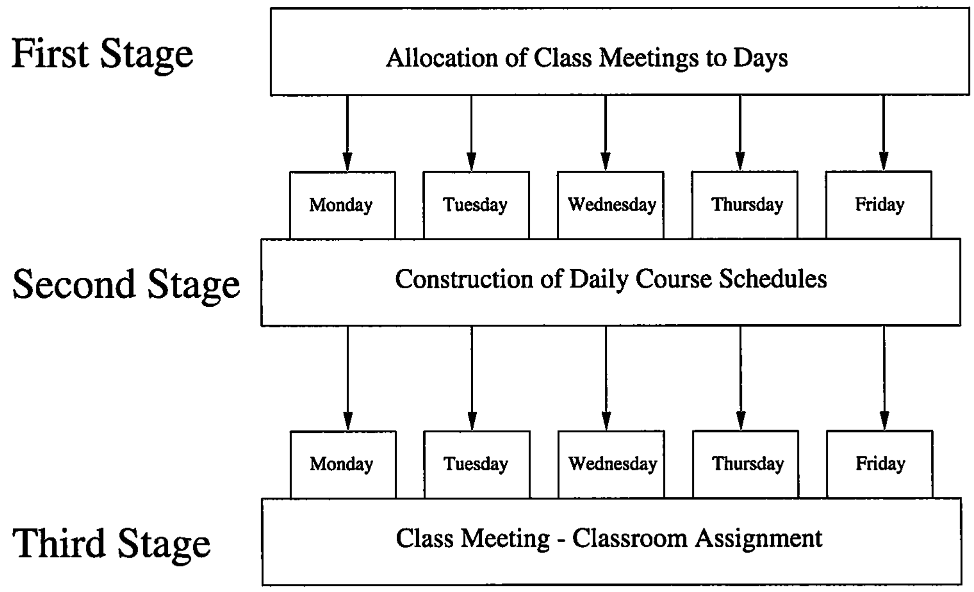

First Stage
Allocation of Class Meetings to Days
Monday
Tuesday
Wednesday
Thursday
Friday
Second Stage
Construction of Daily Course Schedules
Monday
Tuesday
Wednesday
Thursday
Friday
Third Stage
Class Meeting - Classroom Assignment

Figure 2.1: Stages of the Solution Process

the third stage, the problem of class meeting - classroom assignment is formulated as a mathematical program.

These same three stages can all be formulated as mathematical programs, as is done in Appendix E. But CP modeling of the first two stages give considerable flexibility for future inclusion of other constraints with minimal increase in the computational requirements.

2.1 Allocation of Class Meetings to Days

The class meetings are allocated to days considering only the following constraints:

1. All class meetings should be assigned to a day.
2. Daily classroom capacity should not be exceeded.

3. Number of class meeting hours that a student can attend in a day should be taken into account.

4. Number of class meeting hours that an instructor can teach in a day should be taken into account.

5. Two class meetings of the same course section should not be scheduled on consecutive days.

In Chapter 1, it was stated that each class meeting requires a classroom of a specific type. In this problem the classroom types can be considered as resources which are used by class meetings. The daily resource capacities depend on the number of classrooms belonging to the classroom type represented by that resource. For example, having 25 *type-1* classrooms means that the daily class hour capacity of *type-4* resource is $25 \times 9 = 225$ (number of classrooms $\times$ number of time slots in a day). At a day where a class meeting is allocated, this class meeting requires a classroom of its type for x number of hours where x is the length of the class meeting.

During the allocation of class meetings to days, it should be recalled that the total hours of the class meetings that a student attend per day can at most be eight (one out of nine time slots should be left for lunch break). Similar to students, there is a limit for the total hours of the class meetings taught by an instructor. It is possible to change this limit for an instructor upon her request.

The last constraint of this stage is about the time between the two class meetings of a course section. Such class meeting pairs cannot be scheduled on consecutive days.

If all the constraints are considered, it will be seen that this problem is a constraint satisfaction problem (CSP) where class meetings are variables and days are the domains of the variables. This problem is formulated as a CP model. The following definitions and notation are needed to describe the model.

Group is a set of students. For each student group, there is a set of course sections. A student from a student group can attend course sections in that group's set of course sections. For this reason, the class meetings belonging to the course sections from the same set cannot be scheduled at the same time slots and such class meetings' total hours cannot exceed eight in a day, since a student cannot attend more than eight class hours.

Resource The classroom types are the resources. In this problem there are four classroom types, {*type-1, type-2, type-3, type-4*}.

Sets

S Set of class meetings, $\{m_1, \ldots, m_n\}$

I Set of instructors, $\{I_1, \ldots, I_m\}$

S_{I_i} Set of class meetings given by instructor I_i

G Set of student groups, $\{g_1, \ldots, g_k\}$

S_{g_i} Set of class meetings belonging to student group g_i

R Set of resources, $\{r_1, \ldots, r_4\}$

S_{r_k} Set of class meetings that require resource r_i

P Set of class meeting pairs m_j, m_k that belong to the same course section

Parameters

h_{m_j} Hours of class meeting m_j

l_{m_j} Resource requirement of class meeting m_j per day

c_{r_k} Maximum capacity of resource r_k per day

Variables

T_{m_j} Assigned day of class meeting m_j

Constraints

The objective is to find a solution satisfying the following constraints:

(1) $T_{m_j} \in \{0, \ldots, 4\}, \qquad \forall m_j \in S$

(2) $\sum_{m_j \in S_{r_k}} (T_{m_j} = t) \times l_{m_j} \leq c_{r_k}, \qquad \forall r_k \in R, t \in \{0, \ldots, 4\}$

(3) $\sum_{m_j \in S_{g_i}} (T_{m_j} = t) \times h_{m_j} \leq 8, \qquad \forall g_i \in G, t \in \{0, \ldots, 4\}$

(4) $\sum_{m_j \in S_{I_i}} (T_{m_j} = t) \times h_{m_j} \leq 5, \qquad \forall I_i \in I, t \in \{0, \ldots, 4\}$

(5) $T_{m_j} - T_{m_k} \geq 2 \vee T_{m_k} - T_{m_j} \geq 2, \qquad \forall (m_j, m_k) \in P.$

When this problem is solved, the output will indicate which class meeting will take place on which day. The constraints ensure that the daily classroom capacities, the instructors' and the students' daily class hour limits are not exceeded.

2.2 Construction of Daily Course Schedules

In this model, the class meetings that are assigned to a specific day by the previous stage are given a specific time slot of the day, subject to the constraints :

1. Every class meeting on that day should be scheduled to a time slot ensuring that it ends at 5:30 p.m at the latest.
2. At each time slot, the total capacity requirement of the class meetings for a specific classroom type cannot exceed the available capacity of that classroom type.
3. Two class meetings taught by the same instructor cannot be scheduled on overlapping time slots.
4. Two class meetings of a student group cannot be scheduled on overlapping time slots.

The first constraint ensures that if a class meeting is allocated to a day by first stage, then it should be scheduled on a time slot on that day. The classroom types are, again, considered as resources. However, this time the maximum hourly resource capacities are used that is equal to the number of classrooms in that resource type. For any time slot, the class meetings using a specific classroom type cannot exceed the number of classrooms in that classroom type.

Finally, the instructors cannot teach more than one class meeting and a student cannot attend to more than one class meeting at a time slot. In other words, the class meetings taught by the same instructor cannot be scheduled to overlapping time slots, nor the class meetings taken by the same student group.

This problem is a CSP and to find a daily course schedule it is possible to construct a CP model for each day by using the following notation.

Sets

S Set of class meetings, $\{m_1, \ldots, m_n\}$

I Set of instructors, $\{I_1, \ldots, I_m\}$

S_{I_i} Set of class meetings given by instructor I_i

G Set of student groups, $\{g_1, \ldots, g_k\}$

S_{g_i} Set of class meetings belonging to student group g_i

R Set of resources, $\{r_1, \ldots, r_4\}$

S_{r_k} Set of class meetings that require resource r_i

Parameters

h_{m_j} Hours of class meeting m_j

l_{m_j} Resource requirement of class meeting m_j per hour

c_{r_k} Maximum capacity of resource r_k per hour

Variables

T_{m_j} Start time of class meeting m_j

Constraints

The objective is finding a feasible solution subject to the following constraints:

(1) $T_{m_j} \in \{0, \ldots, 9 - h_{m_j}\}, \qquad \forall m_j \in S$

(2) $\sum_{m_j \in S_{r_k}} \sum_{i=0}^{h_{m_j}-1} (T_{m_j} = t - i) \times l_{m_j} \leq c_{r_k}, \qquad \forall r_k \in R,\ t \in \{0, \ldots, 8\}$

(3) $\sum_{m_j \in S_{I_i}} \sum_{i=0}^{h_{m_j}-1} (T_{m_j} = t - i) \leq 1, \qquad \forall I_i \in I,\ t \in \{0, \ldots, 8\}$

(4) $\sum_{m_j \in S_{g_i}} \sum_{i=0}^{h_{m_j}-1} (T_{m_j} = t - i) \leq 1, \qquad \forall g_i \in G,\ t \in \{0,\ldots,8\}.$

This model schedules the class meetings of each day resulting in the weekly course schedule. However, this schedule contains only course and time information, but no classroom assignments are made. Next model assigns classrooms to the scheduled class meetings.

2.3 Assignment of Class Meetings to Classrooms

There are four type of classrooms *type-1*, *type-2*, *type-3* and *type-4* with capacities 25, 30, 40, 65 students, respectively. In previous models, it was assumed that a class meeting requires a classroom of only a specific type, but this assumption is not applied to the meetings requiring *type-1* classrooms. Since the number of classrooms in this resource type cannot meet the demand, while running the previous models, it was allowed that class meetings requiring *type-1* classrooms can take place in every kind of classroom. This is valid, because *type-1* classrooms are for 25 students and if a class meeting has 25 students, then it can be placed in a classroom with 30, 40 or 65 students capacity. In fact every class meeting can be assigned to a classroom having sufficient student capacity. For example a class meeting with 35 students can be assigned to a *type-4* classroom instead of a *type-3* classroom. This is made possible by the current model. The problem is to find appropriate classrooms for class meetings scheduled on that day satisfying the constraints:

1. Every class meeting should be assigned to a classroom.
2. There can be at most one class meeting in a classroom during a time slot.

3. For all class meetings that are two hours long, first and second hours should be assigned to the same classroom.

4. For all class meetings that are three hours long, first, second and third hours should be assigned to the same classroom.

This problem is formulated as an MP model by using the following notation.

Sets

S Set of class meetings, $\{m_1, \ldots, m_n\}$

R Set of classroom types, $\{r_1, r_2, r_3, r_4\}$

S_{r_k} Set of class meetings that need a classroom of type r_k

H_i Set of class meetings that are i hour long, $i \in \{1, 2, 3\}$

C_{r_k} Set of classrooms that are in type r_k

Parameters

$start(m_j)$ Start time of class meeting m_j, $start(m_j) \in \{0, \ldots, 8\}$

Variables

$$X_1(m_j, k, l) = \begin{cases} 1, & \text{if class meeting } m_j \text{ is assigned to} \\ & k^{th} \text{ element of } C_{r_1} \text{ at time slot } l; \\ 0, & \text{otherwise.} \end{cases}$$

$$X_2(m_j, k, l) = \begin{cases} 1, & \text{if class meeting } m_j \text{ is assigned to} \\ & k^{th} \text{ element of } C_{r_2} \text{ at time slot } l; \\ 0, & \text{otherwise.} \end{cases}$$

$$X_3(m_j,k,l) = \begin{cases} 1, & \text{if class meeting } m_j \text{ is assigned to} \\ & k^{th} \text{ element of } C_{r_3} \text{ at time slot } l; \\ 0, & \text{otherwise.} \end{cases}$$

$$X_4(m_j,k,l) = \begin{cases} 1, & \text{if class meeting } m_j \text{ is assigned to} \\ & k^{th} \text{ element of } C_{r_4} \text{ at time slot } l; \\ 0, & \text{otherwise.} \end{cases}$$

Constraints

The objective function is formulated according to the scheduler's preferences, such as maximizing classroom utilization, minimizing the distance traveled by the students, etc. The constraints are:

(1) $\sum_{k\in C_{r_4}} X_4(m_j,k,start(m_j)) = 1, \qquad \forall m_j \in S_{r_4}$

$\sum_{k\in C_{r_4}} X_4(m_j,k,start(m_j))+$
$\sum_{k\in C_{r_3}} X_3(m_j,k,start(m_j)) = 1, \qquad \forall m_j \in S_{r_3}$

$\sum_{k\in C_{r_4}} X_4(m_j,k,start(m_j))+$
$\sum_{k\in C_{r_3}} X_3(m_j,k,start(m_j))+$
$\sum_{k\in C_{r_2}} X_2(m_j,k,start(m_j)) = 1, \qquad \forall m_j \in S_{r_2}$

$\sum_{k\in C_{r_4}} X_4(m_j,k,start(m_j))+$
$\sum_{k\in C_{r_3}} X_3(m_j,k,start(m_j))+$
$\sum_{k\in C_{r_2}} X_2(m_j,k,start(m_j))+$
$\sum_{k\in C_{r_1}} X_1(m_j,k,start(m_j)) = 1, \qquad \forall m_j \in S_{r_1}$

(2) $\sum_{m_j\in(S_{r_1}\cup S_{r_2}\cup S_{r_3}\cup S_{r_4})} X_4(m_j,k,l) \le 1, \qquad \forall k \in C_{r_4},\ l \in \{0,\ldots,8\}$

$\sum_{m_j\in(S_{r_1}\cup S_{r_2}\cup S_{r_3})} X_3(m_j,k,l) \le 1, \qquad \forall k \in C_{r_3},\ l \in \{0,\ldots,8\}$

$\sum_{m_j \in (S_{r_1} \cup S_{r_2})} X_2(m_j, k, l) \leq 1, \qquad \forall\, k \in C_{r_2},\ l \in \{0, \ldots, 8\}$

$\sum_{m_j \in S_{r_1}} X_1(m_j, k, l) \leq 1, \qquad \forall\, k \in C_{r_1},\ l \in \{0, \ldots, 8\}$

(3) $X_4(m_j, k, start(m_j)) = X_4(m_j, k, start(m_j) + 1), \forall\, m_j \in ((S_{r_4} \cup S_{r_3} \cup S_{r_2} \cup S_{r_1}) \cap H_2), \forall\, k \in C_{r_4}$

$X_3(m_j, k, start(m_j)) = X_3(m_j, k, start(m_j) + 1), \forall\, m_j \in ((S_{r_3} \cup S_{r_2} \cup S_{r_1}) \cap H_2), \forall\, k \in C_{r_3}$

$X_2(m_j, k, start(m_j)) = X_2(m_j, k, start(m_j) + 1), \forall\, m_j \in ((S_{r_2} \cup S_{r_1}) \cap H_2), \forall\, k \in C_{r_2}$

$X_1(m_j, k, start(m_j)) = X_1(m_j, k, start(m_j) + 1), \forall\, m_j \in (S_{r_1} \cap H_2), \forall\, k \in C_{r_1}$

(4) $X_4(m_j, k, start(m_j)) = X_4(m_j, k, start(m_j) + 1), \forall\, m_j \in ((S_{r_4} \cup S_{r_3} \cup S_{r_2} \cup S_{r_1}) \cap H_3), \forall\, k \in C_{r_4}$

$X_4(m_j, k, start(m_j)) = X_4(m_j, k, start(m_j) + 2), \forall\, m_j \in ((S_{r_4} \cup S_{r_3} \cup S_{r_2} \cup S_{r_1}) \cap H_3), \forall\, k \in C_{r_4}$

$X_3(m_j, k, start(m_j)) = X_3(m_j, k, start(m_j) + 1), \forall\, m_j \in ((S_{r_3} \cup S_{r_2} \cup S_{r_1}) \cap H_3), \forall\, k \in C_{r_3}$

$X_3(m_j, k, start(m_j)) = X_3(m_j, k, start(m_j) + 2), \forall\, m_j \in ((S_{r_3} \cup S_{r_2} \cup S_{r_1}) \cap H_3), \forall\, k \in C_{r_3}$

$$X_2(m_j, k, start(m_j)) = X_2(m_j, k, start(m_j) + 1), \forall\, m_j \in ((S_{r_2} \cup S_{r_1}) \cap H_3), \\ \forall\, k \in C_{r_2}$$

$$X_2(m_j, k, start(m_j)) = X_2(m_j, k, start(m_j) + 2), \forall\, m_j \in ((S_{r_2} \cup S_{r_1}) \cap H_3), \\ \forall\, k \in C_{r_2}$$

$$X_1(m_j, k, start(m_j)) = X_1(m_j, k, start(m_j) + 1), \forall\, m_j \in (S_{r_1} \cap H_3), \\ \forall\, k \in C_{r_1}$$

$$X_1(m_j, k, start(m_j)) = X_1(m_j, k, start(m_j) + 2), \forall\, m_j \in (S_{r_1} \cap H_3), \\ \forall\, k \in C_{r_1}$$

(5) $X_4,\ X_3,\ X_2,\ X_1 \in \{0, 1\}$.

Chapter 3

A Course Scheduling System

The models formulated in the last chapter are now to be used as parts of the *course scheduling system* (CSS) for Bilkent University. This CSS is designed to help the timetabler for generating feasible course schedules. As seen in Figure 3.1, it uses both data and model interactively and the software used in this system, as explained in §3.2, provides the necessary user interface.

3.1 Data Transfer in CSS

The proposed system is composed of a number of models and data transfer among these models is necessary, since the output of a model is required as the input data to another. In CSS, data manipulations and transfers can be grouped into three classes:

1. Preparation of input data for allocation of class meetings to days,
2. Input data for construction of daily course schedules, and
3. Input data for class meeting - classroom assignment.

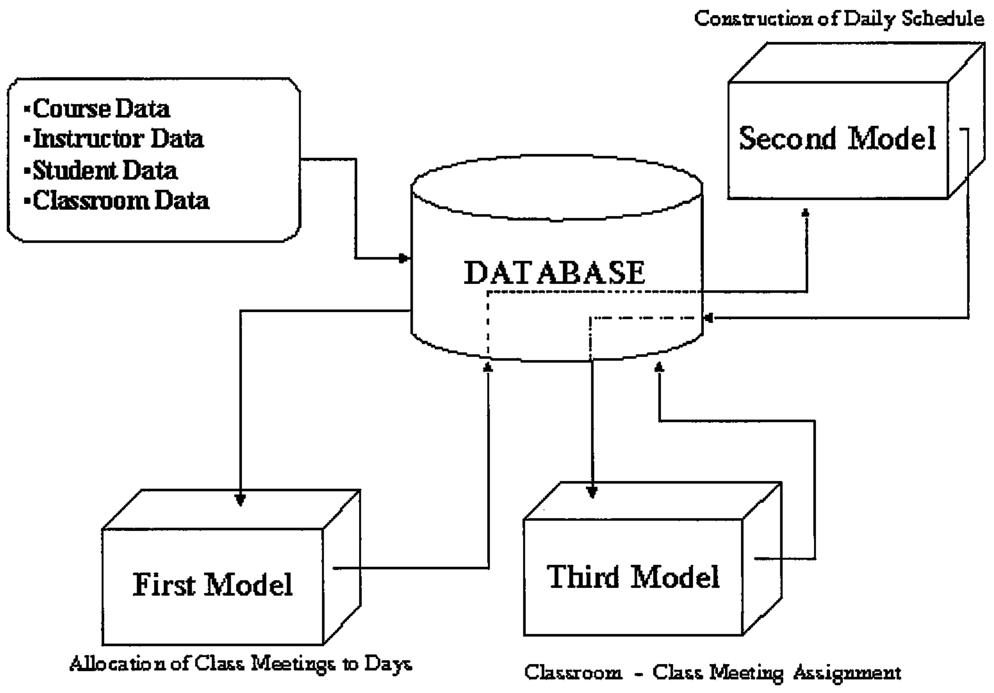

Construction of Daily Schedule
•Course Data
•Instructor Data
•Student Data
•Classroom Data
Second Model
DATABASE
First Model
Third Model
Allocation of Class Meetings to Days
Classroom - Class Meeting Assignment

Figure 3.1: Model and Data Interaction in Proposed CSS

These are explained in detail in the following subsections.

3.1.1 Preparation of Input Data for Allocation of Class Meetings to Days

First, the student groups and the course sections reserved for each student group are determined. To simplify the problem and to prevent infeasibilities, some class meetings belonging to the sections of the same courses are combined and considered as a single class meeting. This is necessary, because the total hours for the class meetings of the course sections reserved for each student group can be at most 40 in a week. However, initially for some student groups the total hours can be more than 40. By considering different class meetings as a single class meeting, the infeasibilities for such student groups are eliminated. During these modifications, the resource and instructor requirements are considered and if a new class meeting is generated, its classroom requirement is set as the total of the classroom requirements of the class meetings that it includes, and it requires all the instructors required by the previous class meetings. This modification results in a course schedule in which class meetings of the combined sections start at the same time.

After generating the course data, a database file with the following column names is constructed:

Group	Meeting	Section	Hour	Resource Req.	Classroom Type	Instructor

A specific row of the above database file gives the following information under each column:

Group Name of the student group attending to class meeting,

Meeting Class meeting for which the information is supplied by that row,

Section Course section of class meeting that it belongs to,

Hour Length of class meeting,

Resource Requirement The total classroom hours required by class meeting in a day,

Classroom Type The type of the classroom that is required by class meeting,

Instructor Identity number of the instructor who teaches the class meeting.

For storing the resource (classroom) data, another database file is constructed with column names:

Resource	Capacity	Daily Capacity

In this database file, the columns represent the following information:

Resource The classroom type (*type-1, type-2, type-3, type-4*),

Capacity Number of classroom hours available in an hour for each classroom type or in other words resource capacity per hour,

Daily Capacity Number of classroom hours available in a day for each resource

Finally, a third database file is required to keep the information about class meeting pairs belonging to the same course sections with column names:

Meeting	Next

The above three database files keep the necessary input data for allocation of class meetings to days. After the run of the first model, the day for each class meeting is determined and the output is again stored in a database file with column names:

Meeting	Day

Where each row shows the class meeting's day in terms of integers $\{0, \ldots, 4\}$ respectively instead of {Monday, ..., Friday}.

3.1.2 Input Data For Construction of Daily Course Schedules

For constructing daily course schedules, the input file is retrieved from a new database file constructed by combining the result of first model and initial course data as seen in Figure 3.2

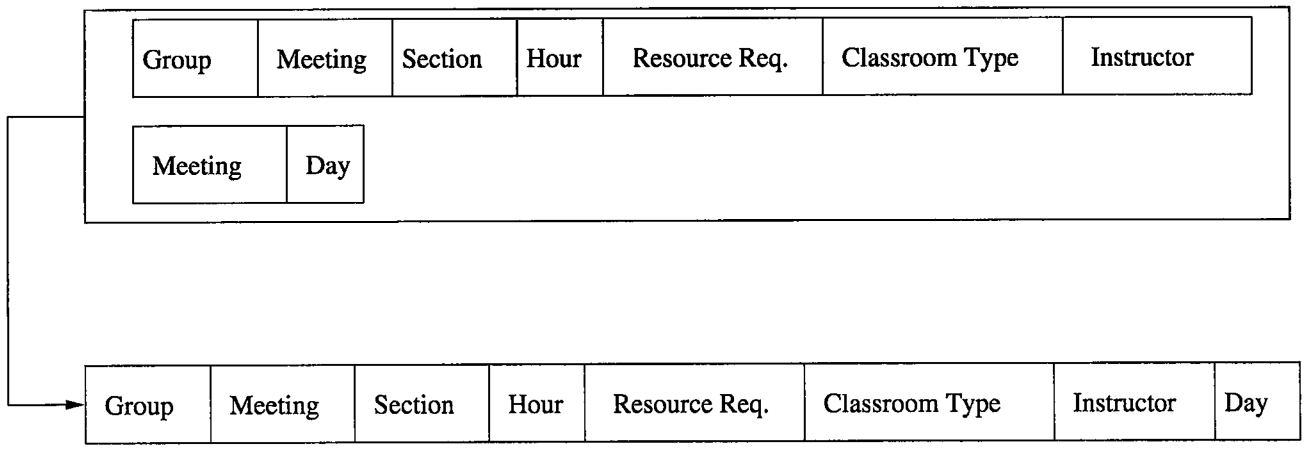


Figure 3.2: Input Data for Constructing Daily Course Schedules

The model for construction of daily course schedules is run for each day. At each run, input data is retrieved from the new database file and the resource file. After each run, the output is stored in a database file with columns:

Meeting	Time Slot

Where time slot column stores the information about the start time of the class meeting in terms of integers $\{0, \dots, 8\}$ respectively instead of {8:40, ..., 16:40}.

3.1.3 Input Data for Class Meeting - Classroom Assignment

In this stage, by using the outputs of previous stages, new database files for each day are constructed with columns:

Section	Hour	Classroom Type	Time Slot

and a new classroom file is constructed with columns:

Classroom Name	Classroom Type

For each day, meeting - classroom assignment is done by retrieving input data from the above files. The outputs are stored in a database file. By using these outputs, a final database file is generated with columns:

Group	Course Section	Hour	Day	Time Slot	Classroom	Instructor

in which all the information about the course schedule is stored. All data storage and retrieval processes are done using *Oracle8 Release 8.0.5* server.

3.2 ILOG OPL Studio

All the models in this CSS are run by using the software ILOG OPL Studio 2.1.3. OPL Studio [29] is an integrated development environment for combinatorial optimization applications. It can be effectively used for constructing and solving linear programming, integer programming, and constraint programming models. The model codes are written in optimization programming language (OPL) developed by Pascal Van Hentenryck [26].

OPL Studio has several tools including CPLEX 6.5.3 (mathematical programming solver), SOLVER 4.4 (solver for constraint programming) and SCHEDULER 4.4 (a tool developed for constraint based scheduling). Once a model is constructed, after compilation, OPL Studio automatically detects the the problem type and determines the most convenient solver to solve it.

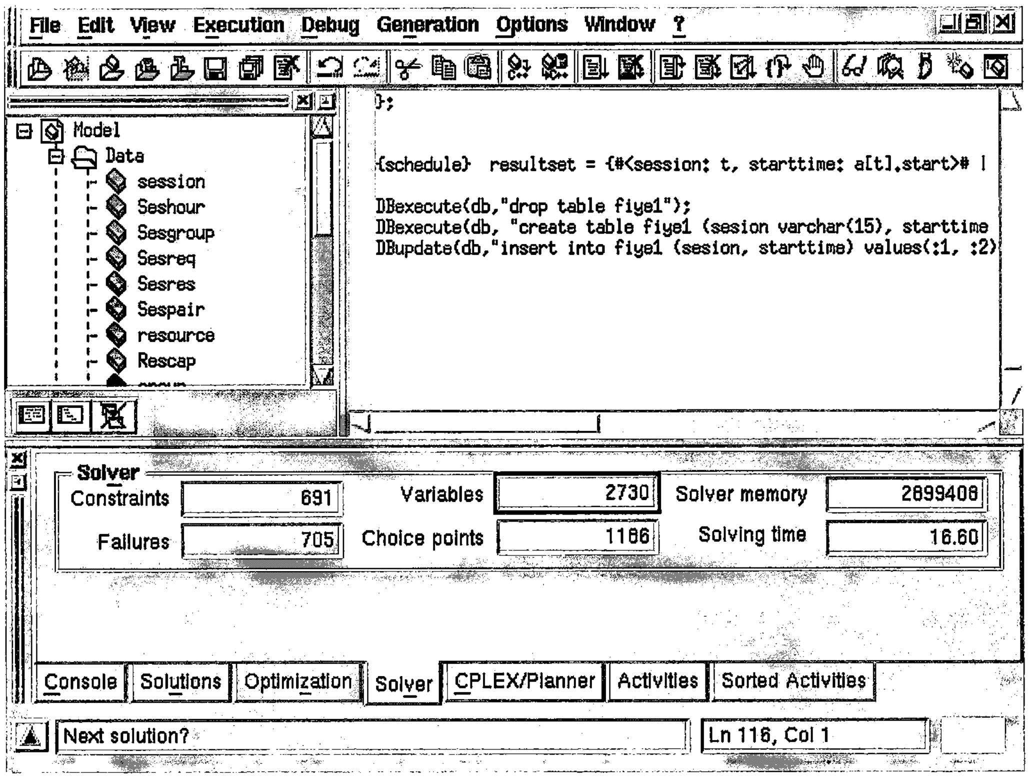
ILOG OPL Studio has a user friendly graphical environment (Figure 3.3). Unlike most other programming environments, the compilation errors are displayed clearly. By using menus, it is possible to change parameter settings (iteration limit, tolerance level, etc), search procedures (dept first search, limited discrepancy search, etc). The display of solution in graphical format is possible such as resource utilizations, and charts for resource constrained scheduling problems.

Another feature of OPL Studio is its ability to decide on the most efficient solver for a model. It is able to solve both mathematical programming and constraint programming models. Since in CSS, the solution is obtained by utilizing both constraint programming and mathematical programming, OPL Studio is very convenient single system.

In this course scheduling problem, the size of the course, student, instructor data is very large and it is not easy to manipulate the data without a database. At the same time, in CSS data is retrieved and stored several times that necessitates a database. Since OPL Studio has database connection capability, the data can easily retrieved and stored by OPL Studio before and after running the models.

OPL Studio has a special tool called *scheduler* that contains efficient algorithms for solving the constraints of resource constrained scheduling problems. This option makes it possible to define and solve the constraints related to the resources and the activities very easily. Without using complicated mathematical expressions, it is possible to define constraints for resources, resource capacities, activities, resource requirements of activities as simple as writing a sentence in an ordinary language. Since the course scheduling problem is a type of resource constrained scheduling problem, the classroom types can be represented as resources with available capacities and the class meetings are the activities to be scheduled whose durations equal to length of class meetings.

All these features make ILOG OPL Studio the ideal tool for CSS of Bilkent University.



```
};


{schedule}  resultset = {#<session: t, starttime: a[t].start># |

DBexecute(db,"drop table fiye1");
DBexecute(db, "create table fiye1 (sesion varchar(15), starttime
DBupdate(db,"insert into fiye1 (sesion, starttime) values(:1, :2)
```

```
Solver
Constraints   691    Variables       2730   Solver memory   2899408
Failures      705    Choice points   1166   Solving time    16.60

Next solution?                          Ln 116, Col 1
```

Figure 3.3: User Interface in ILOG OPL Studio

3.3 OPL Formulations of Constraint Programming Models

CP models that are used in the first and second stages of CSS can easily be represented by using OPL's special constraint definitions. In the following subsections, OPL representation of CP models is described.

3.3.1 OPL Model for Allocation of Class Meetings to Days

Sets and Parameters

S Set of class meetings, $\{m_1, \ldots, m_n\}$

I Set of instructors, $\{I_1, \ldots, I_m\}$

S_{I_i} Set of class meetings taught by instructor I_i

G Set of student groups, $\{g_1, \ldots, g_k\}$

S_{g_i} Set of class meetings belonging to student group g_i

R Set of resources, $\{r_1, r_2, r_3, r_4\}$

S_{r_k} Set of class meetings that require resource r_k

P Set of class meeting pairs m_j, m_k that belong to the same course section

h_{m_j} Hours of class meeting m_j

l_{m_j} Resource requirement of class meeting m_j per day

c_{r_k} Maximum capacity of resource r_k per day

Constraints

(1) Schedule horizon is from 0 to 4,

(2) m_j is an activity with duration 1, $\forall m_j \in S$

(3) r_k is a resource with capacity c_{r_k}, $\forall r_k \in R$

(4) m_j requires l_{m_j} amount of resource r_k, $\forall\, r_k \in \{r_2, r_3, r_4\}, \forall\, m_j \in S_{r_k}$

(5) m_j requires l_{m_j} amount of a possible resource $r_k \in R$, $\forall\, m_j \in S_{r_1}$

(6) $m_j.start \;-\; m_k.start \geq 2 \;\vee\; m_k.start \;-\; m_j.start \geq 2$, $\forall\, (m_j, m_k) \in P$

(7) $\sum_{m_j \in S_{g_i}} (m_j.start = t) \times h_{m_j} \leq 8$, $\forall\, g_i \in G,\ t \in \{0, \ldots, 4\}$

(8) $\sum_{m_j \in S_{I_i}} (m_j.start = t) \times h_{m_j} \leq 5$, $\forall\, I_i \in I,\ t \in \{0, \ldots, 4\}$.

A class meeting requiring *type-1* classroom can be placed in every type of classroom (constraint 4). This is provided in OPL Studio by using the special constraint type *alternative resource*. Alternative resource is defined as a set of resources and an activity which requires alternative resource can be scheduled on any of the possible resources in the set of alternative resource.

Constraint 6 ensures that there should be at least one day between the two class meetings of a course section. Constraints 7 and 8 control the daily course load of students and instructors respectively.

3.3.2 OPL Model for Construction of Daily Course Schedules

Sets and Parameters

S Set of class meetings, $\{m_1, \ldots, m_n\}$

I Set of instructors, $\{I_1, \ldots, I_m\}$

S_{I_i} Set of class meetings taught by instructor I_i

G Set of student groups, $\{g_1, \ldots, g_k\}$

S_{g_i} Set of class meetings belonging to student group g_i

R Set of resources, $\{r_1, r_2, r_3, r_4\}$

S_{r_k} Set of class meetings that require resource r_k

h_{m_j} Hours of class meeting m_j

l_{m_j} Resource requirement of class meeting m_j per hour

c_{r_k} Maximum capacity of resource r_k per hour

Constraints

(1)	Schedule horizon is from 0 to 8,	
(2)	m_j is an activity with duration h_{m_j},	$\forall\, m_j \in S$
(3)	r_k is a resource with capacity c_{r_k},	$\forall\, r_k \in R$
(4)	I_i is a resource with capacity 1,	$\forall\, I_i \in I$
(5)	g_i is a resource with capacity 1,	$\forall\, g_i \in G$
(6)	m_j requires g_i,	$\forall\, g_i \in G, \forall\, m_j \in S_{g_i}$
(7)	m_j requires instructor I_i,	$\forall\, I_i \in I, \forall\, m_j \in S_{I_i}$
(8)	m_j requires l_{m_j} amount of resource r_k,	$\forall\, r_k \in \{r_2, r_3, r_4\}, \forall\, m_j \in S_{r_k}$

(9) m_j requires l_{m_j} amount of a possible resource $r_k \in R$, $\quad \forall\, m_j \in S_{r_1}$.

As seen in the formulation, the instructors and the student groups are thought as resources with capacities one. This is valid, because to have a class meeting both the instructor and the students should be available. Since OPL Studio's resource constraints provide efficiency in solution process, defining the student groups and instructors as resources makes the problem easier to solve.

Again as in the previous model, constraint 9 ensures that the class meetings requiring *type-1* classrooms can be scheduled on any of the classrooms.

3.4 Solving Course Scheduling Problem

One of the main features of decision support systems is "friendly" interaction with the user. This is also true for the CSS proposed in this study. During program runs, the timetabler makes adjustments to have a balanced schedule, by

1. Balancing the distribution of class meetings over days,
2. Generating a feasible daily course schedule,
3. Providing lunch breaks to students.

3.4.1 Balancing the Distribution of Class Meetings over Days

One problem that may require timetabler's adjustment in the first stage may be the non-uniform distribution of class meetings over the days of the week. This has two disadvantages:

1. The course schedule.will not be balanced,
2. There may be no feasible daily course schedule for the days on which too many course meetings are allocated.

An unbalanced course schedule, although it is feasible, generally undesirable, but there are ways for the timetabler to avoid such a situation.

Reduction of Capacity:

One solution to avoid non-uniform course distribution over days is to decrease the daily capacity of resources. A capacity decrease in a resource corresponds to a decrease in the number of classrooms represented by that resource. Consequently, this will bound the maximum number of class meetings that can be scheduled on a day for each resource type and force the program to have a uniform distribution of class meetings over days.

Adding Extra Constraints:

By adding new constraints, it is possible to have a uniform distribution of class meetings. If there is a large number of class meetings scheduled for a student group or a resource on a day, this may cause an infeasible course schedule on that day. An example for this case is given in OPL Studio's resource utilization graph

(Figure 3.4). As it is seen, on Monday and Wednesday, there is an overload of class meetings for *type-3* resource. To avoid such a situation, extra constraints can be added that limit the maximum number of class meetings allocated on Monday and Wednesday for the student groups or *type-3* resource's capacity can be decreased for Monday and Wednesday.

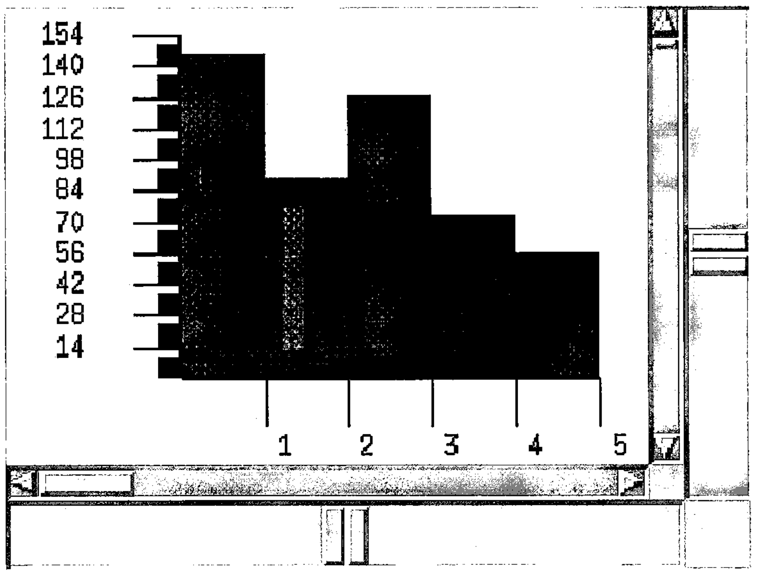


Figure 3.4: Utilization of *Type-3* Resource in First Stage

Another way for having a uniform distribution of courses all over the days may be adding constraints limiting the maximum number of class meeting hours a student group can attend in a day. Ensuring that for the new daily lecture hour limit, all the class meetings of the student group can be scheduled, then instead of eight hours, the maximum number of class meeting hours for a student group can be taken as a lesser number. For example, in this study, the maximum number of class meeting hours for the third and fourth year student groups is taken as six.

3.4.2 Generating a Feasible Daily Course Schedule:

In second stage, during the run of CP model for construction of daily course schedules, although it is highly unlikely, sometimes there may be no feasible course schedule for a day. Such a problem may arise, because some class meetings are reserved for more than one student group and there may be schedule conflicts because of the joint constraints concerning class meetings, instructors, and student groups.

In such a case, the timetabler can find a solution by running the first model again with extra constraints. These extra constraints are useful for removing the student group, class meeting and instructor constraints on the day where schedule conflict occurs. Example of such constraints are:

1. Limiting the number of class meetings on the day where schedule conflict occurs,
2. Limiting total class meeting hours on the day where schedule conflict occurs,
3. Limiting total class meeting hours for the student groups on the day where schedule conflict occurs.

By this way, a new allocation of class meetings to days is obtained and second model is run for each day to generate daily course schedules.

3.4.3 Adjustment for Lunch Breaks

After running the second model for each day in second stage and obtaining daily course schedules, there is a high probability that some student groups have class meetings consequently for seven or eight hours without a lunch break. This is not desirable and to avoid this there are several ways. Timetabler can insert constraints that prevents scheduling class meetings on lunch time for the student groups having more than three class meetings on that day. However, sometimes such a constraint results in no feasible solution. In this case, the constraint can be relaxed. For example, instead of three class meetings, for the student group with more than four class meetings, the lunch break constraint is added.

Another way for providing lunch break can be adding an objective function in the second model that minimizes the number of class meetings scheduled on lunch time. However, since the search space is very large, optimal solution may not be obtained in a reasonable time. Considering this probability, an extra termination criteria can be inserted for the program. This termination criteria may be "elapse of a determined time period after the start of the program" or "having no improvement on the objective function for a determined time period". When the program terminates, the output at termination step is analyzed. After analyzing the output and finding the groups with infeasible schedules, specific constraints are added incrementally for eliminating the infeasibilities and program is run several times till finding a feasible schedule for the lunch break requirements of students.

ILOG OPL Studio's charts, as seen in Figure 3.5, are very convenient for making adjustments. By examining these charts, the student groups having class meeting on lunch time can easily be determined and specific constraints can be added for such student groups.

3.5 Search Strategy

In CSS search strategy is the key point for the success of CP models in first and second stages (see Appendix C for more information about search strategy). In this system, there are two different CP models. First model is run once and the second model is run five times (for each day). This means there should be at least

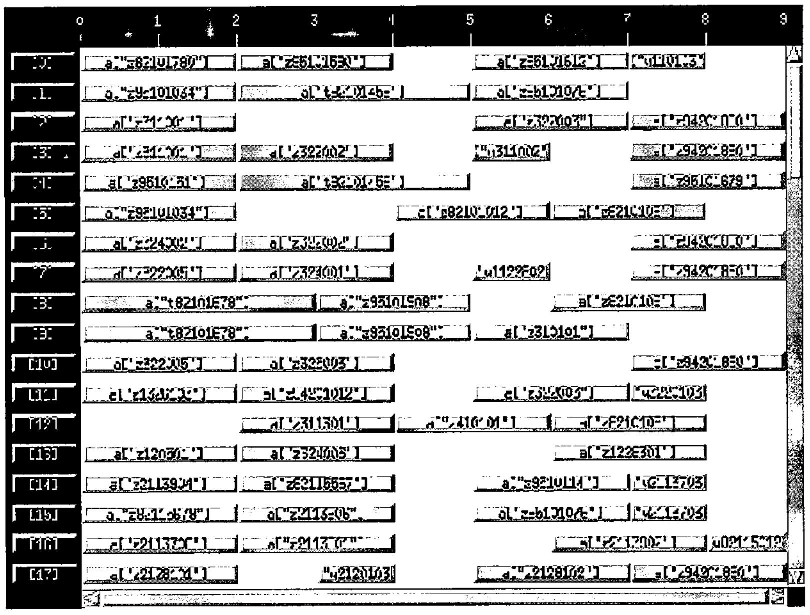


Figure 3.5: Daily Course Schedule of Student Groups

two different search strategies used. One strategy should be for the first model and one or more search strategies should be for the second model. The reason for having more than one search strategy in second model is that in second stage the model is run five times. At each run, the data is different and different data sets may require different search strategies. As an example on the first day there may be a bottleneck for *type-3* classrooms and on the second day the bottleneck can be for *type-4* classrooms. In such a case, if first fail principle is used, then on the first day a search strategy is used to schedule the class meetings requiring *type-3* classrooms first and on the second day, the search strategy should first schedule the class meetings requiring *type-4* classrooms.

In general, first fail principle in variable ordering is a good way to construct search strategy and it is recommended for CSS. To apply first fail principle, the variables with the fewest number of values in their domains are chosen for value assignment. At the same time another criteria for defining variable ordering in search strategy is first assigning values to the variables that are involved in the constraints which are hard to satisfy. This is a kind of first fail principle. If a variable is involved in a constraint that is difficult to satisfy, then there is higher probability that the values assigned to this variable will fail. The timetabler should determine the most effective search strategy for CP models in first and second stage by investigating the course data it deals with.

3.6 Display of Course Schedule

In this CSS, the final course schedule information is stored on a database file named *schedule*. It is possible to obtain course schedule information by using SQL commands such as:

To see all course schedule information:

```
SQL>select * from schedule;

.................
.................

GROUPS   SECTION          HOUR        DAY   TIMESLOT CLASSROO INSTRU
-------- ---------- ---------- ---------- ---------- -------- ------
d31y1a   9810105             2          0          4 SC2      8041
d32y1c   9810106             1          0          2 EB267    3538
d32y1c   9810106             2          3          7 EB168    3538
d32y1c   9810107             1          0          5 MA3      3993
d32y1c   9810107             2          3          2 EB263    3993
d34y1a   9810107             1          0          5 MA3      3993
d34y1a   9810107             2          3          2 EB263    3993
d32y1b   9810108             1          3          0 EB168    5005
d32y1b   9810108             2          1          0 SC1      5005
```

```
1614 rows selected.
```

In the groups column, *d* shows department, *y* shows year, *a*, *b*, *c* shows student group. Then to see the course schedule of the first group of the third year industrial engineering students:

```
SQL> select * from schedule where groups = 'd13y3a';

GROUPS   SECTION          HOUR        DAY   TIMESLOT CLASSROO INSTRU
-------- ---------- ---------- ---------- ---------- -------- ------
d13y3a   1030103             2          3          0 MA3      2206
d13y3a   1331101             1          1          4 EB168    305
d13y3a   1331101             2          3          4 EB162    305
d13y3a   1332101             1          2          3 MA7      4186
d13y3a   1332101             2          4          4 MA14     4186
d13y3a   1334101             1          2          2 EB163    1402
d13y3a   1334101             2          4          2 MA14     1402
d13y3a   1337101             1          2          0 MA4      5248
d13y3a   1337101             2          4          0 MA1      5248

9 rows selected.
```

In the same manner, course schedules for instructors and classrooms are retrieved. For example, To see the course schedule of the instructor with identity no. 50:

```
SQL> select * from schedule where instructor = '50';
```

```
GROUPS   SECTION          HOUR        DAY    TIMESLOT CLASSROO INSTRU
-------- ---------- ---------- ---------- ---------- -------- ------
d13y4a   1347701             1          2          0 SC1      50
d13y4a   1347701             2          4          0 MA10     50
```

and to see the course schedule of the classroom EB167:

```
SQL>select * from schedule where classroom = 'EB167';
...............
...............

GROUPS   SECTION          HOUR        DAY    TIMESLOT CLASSROO INSTRU
-------- ---------- ---------- ---------- ---------- -------- ------
d32y1c   9110109             2          4          6 EB167    906
d33y1b   9110109             2          4          6 EB167    906
d34y1b   9110112             2          0          1 EB167    3925
d34y1c   9110112             2          0          1 EB167    3925
d34y1b   9110113             1          0          0 EB167    906
d31y1c   9810103             1          2          5 EB167    5005
d31y1c   9810103             2          0          4 EB167    5005
d34y1c   9810111             1          0          3 EB167    3538

44 rows selected.
```

Chapter 4

Validation of the System and Conclusions

In order to validate CSS, course schedules for Fall 99 and Spring 00 semesters were generated using data provided by STARS (Student Academic Information Registration System)[30] at Bilkent. The input data is prepared for five faculties involving eighteen departments. There are more than 670 course sections in both semesters. For fall semester 112 student groups and for spring semester 107 student groups are formed. In both semesters, total number of students is more than 4,000. The models are run on a SunOS 5.5 - SPARCserver 1000E computer. The required courses of all the students were successfully scheduled.

4.1 Computational Experience

In CP model of the first stage, the initial search strategy used was designed for assigning class meetings to days considering that the resource constraints were the most difficult constraints to satisfy. The ordering of class meetings for allocating to days was determined by the classroom type they require. However, after hours of run, it was not possible to obtain a solution. Later, it was observed that the most problematic constraint in first model is the constraint requiring that the class meetings of the same course sections cannot be scheduled in consecutive days. In particular, it is difficult to satisfy this constraint for the class meetings of first and second year student groups, because these student groups have a large number of class meetings to schedule. Applying first fail principle, in the new search strategy first the class meetings of the first and second year student groups are allocated to days and class meetings of the third and fourth year student groups are allocated later. This change in the search strategy made the models give output in less than one minute for both semesters as seen in tables 4.1 and 4.2.

Constraints	Variables	Choice Points	Failures	Solution Time
3910	7525	2294	10054	50.12 sec

Table 4.1: Model I Statistics for Fall 99 Data

Constraints	Variables	Choice Points	Failures	Solution Time
3931	7693	1329	76	38.15 sec

Table 4.2: Model I Statistics for Spring 00 Data

In CP model of the second stage, again the search strategy is designed to schedule the class meetings that are difficult to place in a classroom. This is done by scheduling class meetings according to their classroom types. For all days, the same search strategy is used while running the second. The second model runs last in less than minute as seen in tables 4.3 and 4.4

In both search strategies, during the assignment of values to the variables, the variable with the fewest number of values in its domain is chosen for value assignment when two variables are compared. In addition to this, the value assignment process is designed to assign the maximum or minimum available values for the variables in their domains. By this way, the distribution of class meetings in the schedule is controlled and the probability of schedule conflict because of classroom capacity constraint is decreased during program run. This results in a shorter program run. The search strategies used are discussed in Appendix D.

Day	Constraints	Variables	Choice Points	Failures	Solution Time
Monday	666	1246	243	0	3.55 sec
Tuesday	828	1617	289	10	5.26 sec
Wednesday	875	1743	291	1	5.74 sec
Thursday	738	1491	317	2	4.10 sec
Friday	680	1428	244	0	4.41 sec

Table 4.3: Model II Statistics for Fall 99 Data

Day	Constraints	Variables	Choice Points	Failures	Solution Time
Monday	688	1358	309	4	3.90 sec
Tuesday	847	1743	384	1	6.95 sec
Wednesday	850	1743	295	44	6.36 sec
Thursday	745	1533	394	4	5.11 sec
Friday	622	1316	317	0	3.90 sec

Table 4.4: Model II Statistics for Spring 00 Data

In the third stage, for class meeting - classroom assignment, runs of MP models without objective function last in less than five minutes as seen in tables 4.5 and 4.6.

4.2 Mathematical Programming Models

It is possible to allocate class meetings to days and construct daily course schedules by using MP models that are given in Appendix E. These models are

Day	Constraints	Variables	Solution Time
Monday	23728	106542	83.88 sec
Tuesday	19660	108234	99.54 sec
Wednesday	19348	127890	170.42 sec
Thursday	15630	92772	72.30 sec
Friday	16362	84861	59.15 sec

Table 4.5: Model III Statistics for Fall 99 Data

Day	Constraints	Variables	Solution Time
Monday	25680	115128	153.53 sec
Tuesday	25970	123183	265.25 sec
Wednesday	23924	130941	181.27 sec
Thursday	16838	106200	178.25 sec
Friday	17024	82971	54.31 sec

Table 4.6: Model III Statistics for Spring 00 Data

run in OPL Studio using solver CPLEX 6.5.3 and the results are obtained. In the output of the first model, the problem of not having a uniform allocation of class meetings to days occurred, but this can be solved by adding extra constraints that limit the number of class meetings in a day. Another solution may be tightening some constraints such as decreasing the daily classroom capacity for classroom types. After obtaining the output for the first stage, other MP model is run for each day in second stage and daily course schedules are constructed.

Compared to CP models, solution of MP models takes a little longer (app. 10 - 15 minutes) where it is less than one minute for CP models. But if the search strategy used in a CP model is not well chosen, then it may not be possible to get an output even in one hour using CP models.

Another comparison can be on the user friendliness of MP and CP models run by OPL Studio. OPL Studio has very good graphical interpretation of outputs such as charts for class meetings , resource utilization graphics for CP model runs in this study. However, the output of MP models are in terms of $(0,1)$ values that are difficult to analyze.

Considering that the purpose of this study is constructing a system to solve the course scheduling problem and taking into account the fact that for a timetabler faster runs of programs and a good graphical interface are important, using CP models for solving this problem seems to be more appropriate.

4.3 Summary and Conclusions

In this thesis, a university timetabling problem is analyzed and a course scheduling system (CSS) is constructed to guide the timetabler for solving the course scheduling problem of Bilkent University.

The system consists of three stages. In the first stage, the class meetings are allocated to days and in the second stage daily course schedules are constructed. In both of these stages constraint programming models are used. In the final stage, the scheduled class meetings are assigned to the classrooms by using mathematical programming. It is also shown that allocation of class meetings to days and construction of daily course schedules can be done by mathematical programming models.

The system is validated for Bilkent University's course offerings and classroom data from previous semesters. The course schedule generated in this study involves the required courses, but the results can easily be extended to a full course schedule involving the elective courses and laboratory sessions by defining appropriate variables in the models.

Bibliography

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* Wiley, 1997.

[2] D. Abramson and J. Abela. A parallel genetic algorithm for solving the school timetabling problem. In *15 Australian Computer Science Conference*, Feb 1992.

[3] F. Azevedo and P. Barahona. Timetabling in constraint logic programming. In *Proceedings of the World Congress on Expert Systems'94*, 1994.

[4] M. A. Badri, D. L. Davis, D. F. Davis, D.F. Davis, and J. Hollingsworth. A multi-objective course scheduling model: Combining faculty preferences for courses and times. *Computers and Operations Research*, 25:303–316, 1998.

[5] Roman Bartak. Online guide to constraint programming. Available at `http://kti.mff.cuni.cz/~bartak/constraints`, (5 June 2000).

[6] Sally C. Brailsford, Chris N. Potts, and Barbara M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119:557–581, 1999.

[7] S. Broder. Final examination scheduling. *Communications of the ACM*, 7:494–498, 1964. Elsevier Science Publishers.

[8] E. K. Burke, D. G. Elliman, and R. F. Weare. A genetic algorithm based university timetabling system. In *Proceedings of the the 2nd East-West International Conference on Computer Technologies in Education*, 1994. Available at `http://www.asap.cs.nott.ac.uk/ASAP/papers/index.html`, (5 June 2000).

[9] E. K. Burke, D. G. Elliman, and R. F. Weare. The automation of the timetabling process in higher education. *Journal of Educational Technology Systems*, 23(4):257–266, 1995. Available at `http://www.asap.cs.nott.ac.uk/ASAP/papers/index.html`, (5 June 2000).

[10] M. W. Carter. A survey of practical applications of examination timetabling problems. *Operations Research*, 24(2):193–201, 1986.

[11] N. Chahal and D. de Werra. An interactive system for constructing timetables on a pc. *European Journal of Operational Research*, 40:32–37, 1989.

[12] A. J. Cole. The preperation of examination timetables using a small store computer. *The Computer Journal*, 7:117–121, 1964.

[13] Dave Corne, Peter Ross, and Hsiao-Lan Fang. Evolutionary timetabling: Practice, prospects and work in progress. In *UK Planning and Scheduling Workshop*, September 1994.

[14] S. B. Deris, S. Omatu, H. Ohta, and PABD. Samat. University timetabling by constraint–based reasoning: A case study. *Journal of the Operational Research Society*, 48:1178–1190, 1997.

[15] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19:151–162, 1985.

[16] Kathryn A. Dowsland. A timetabling problem in which clashes are inevitable. *Journal of the Operational Research Society*, 41(10):907–918, 1990.

[17] Harikleia Frangouli, Vassilis Harmandas, and Panagiotis Stamatopoulos. Utse: Construction of optimum timetables for university courses - a clp based approach. In *PAP 95*, April 1995.

[18] M. Henz and J. Wurtz. Using Oz for college timetabling. In *Proceedings of the 1995 International Conference on the Practice and Theory of Automated Timetabling*, 20 August - 1 September 1995.

[19] Alain Hertz. Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics*, 35:255–270, 1992.

[20] John Hooker, editor. *Logic Based Methods for Optimization: Combining Optimization and Constraint Satisfaction.* John Wiley & Sons, May 2000.

[21] David Johnson. Timetabling university examinations. *Journal of the Operational Research Society*, 41(1):39–47, 1990.

[22] N. M. Karp, editor. *Reducibility among Combinatorial Problems In Complexity of Computer Computations.* Plenum Press, Newyork, 1972.

[23] Gert Smolka. The Oz Programming Model. *Lecture Notes in Computer Science*, 1000:324–343, 1995.

[24] Arabinda Tripathy. School timetabling – a case in large binary integer linear programming. *Management Science*, 30(12):1473–1489, 1984.

[25] Arabinda Tripathy. Computerised decision aid for timetabling – a case analysis. *Discrete Applied Mathematics*, 35:313–323, 1992.

[26] Pascal van Hentenryck, editor. *The OPL Optimization Programming Language.* MIT Press, London, England, 1999.

[27] R. F. Weare. *Automated Examination Timetabling.* PhD thesis, University of Nottingham, UK, 1995. Available at `http://www.asap.cs.nott.ac.uk/ASAP/papers/index.html`, (5 June 2000).

[28] Mike Wright. Timetabling county cricket fixtures using a form of tabu search. *Journal of the Operational Research Society*, 45(7):758–770, 1994.

[29] ILOG OPL Studio. `http://www.ilog.com/products/oplstudio/`, (5 June 2000).

[30] STARS: Student Academic Information Registration System. `http://bliss.bilkent.edu.tr/stars/`, (5 June 2000).

[31] The Eclipse Constraint Logic Programming System. `http://www.icparc.ic.ac.uk/eclipse/`, (5 June 2000).

[32] The Mozart Programming System. `http://www.mozart-oz.org/`, (5 June 2000).

APPENDIX A

Tabu Search and Simulated Annealing

Simulated Annealing (SA) and Tabu Search (TS) algorithms are iterative improvement algorithms that are designed to search for the optimal solution without being trapped at a local optimum. In these algorithms, an initial solution is iteratively modified. These iterative modifications on the solutions generally cause improvement on the objective function value, but in order not to get stuck at a local optimum point, sometimes the modifications can deteriorate the objective function value. The algorithms terminate when no further improvement is possible or stop after a predetermined number of iterations. Clearly, the solution found is not necessarily the global optimum.

SA algorithm is a local search that is designed to search for the global optimum. This approach attracts considerable interest mainly because it is possible to perform asymptotic analysis of the results.

For applying SA, initially there should be

f Cost (objective) function that measures the quality of the solution

S Set of all solutions,

N A neighborhood function $N(x)$ that generates all the solutions in the neighborhood of x ($x \in S$),

$\boldsymbol{t_k}$ Control parameter called temperature at iteration k,

L Control parameter called epoch length,

Cooling Schedule A procedure that controls the value of t_k at each iteration.

The algorithm starts with an initial solution $x \in S$ and at each iteration with some probability, the algorithm moves to a solution that is generated by the neighborhood function N. If $i, j \in S$ where i is the current solution and $j \in N(i)$ then the probability of moving to j from i is given by:

$$P_{ij} = \begin{cases} 1 & \text{if } f(j) \leq f(i) \\ e^{\frac{f(i)-f(j)}{t_k}} & \text{otherwise} \end{cases}$$

As it can be seen from above, at each iteration k, there is a positive probability of moving to a solution which deteriorates the cost function value. This positive probability depends on two factors:

1. The difference between the costs of the two solutions $(f(i) - f(j))$, and
2. The temperature at each iteration k (t_k).

A decrease in t_k or a large difference between the costs of solutions i and j decreases the probability of accepting a worse solution. At low t_k values, it is easier to reach a final solution. For that reason, while moving from iteration k to $k+1$, a new t_{k+1} value is computed by using the cooling schedule. This cooling schedule can be a function such as $t_{k+1} = ct_k$ where c is a constant in the interval $(0, 1)$. At each iteration k, a number of moves are made by using the current temperature t_k. The number of moves at each iteration depends on the parameter L. When the algorithm reaches the end of epoch length L for the current iteration, next iteration starts. When a terminating condition occurs such as reaching a predetermined number of iterations or having a temperature value that is lower than a predetermined value, the algorithm stops and gives the final solution as the best solution found. It is proved that SA algorithm asymptotically converges to the optimal solution [1].

In TS, there also exists a cost function f, a set of feasible solutions S and a neighborhood function $N(x)$ where $x \in S$. The algorithm starts with an initial solution as in SA. At each iteration, the solutions in the neighborhood of the current solution are generated. The objective of the algorithm is to move a solution that has better cost than the current solution's cost. However, for reaching the global optimum, sometimes moves to the solutions with worse cost should be done. In TS, cycling is prevented by keeping a list of forbidden moves called *tabu list*. If a move is in tabu list, it is rejected. At each iteration, this list is modified considering certain criteria [1]. Since tabu list requires memory, there is a trade off between the size of the tabu list and the memory usage.

In SA, there is a positive probability for making a move to a solution with a higher cost function value. In TS, accepting such a move is made possible by using a modified cost function f' for some of the consecutive iterations. This modification is done by introducing extra cost for the neighborhood solutions diversified from the current solution (diversification cost) and introducing extra cost for the neighborhood solutions that resembles the current solution (intensification cost). In the iterations where f' is used, only one of the cost factors, diversification or intensification cost, is added to the original cost function f for a few number of consecutive iterations. If added term is diversification cost then the algorithm tends to explore the current region and if the added term is intensification cost then the algorithm tends to leave the current region. The diversification and the intensification cost factors are used alternatively when f' is used. The iterations of TS go on until a stopping criteria occurs such as not having any allowed move from the current solution or reaching an initially determined number of iterations. Applying TS with different starting solutions may be useful for being able to search the whole solution space.

APPENDIX B

Genetic Algorithms

Genetic Algorithm (GA) is a heuristic technique for producing acceptable timetables. In order to initiate GA, there should be a set of solutions and a fitness (evaluation) function that is used to measure the quality of the solutions. GA is an iterative algorithm and the idea is obtaining good solutions with respect to the preference function by modifying or combining the solutions in the solution set of the previous iteration. Since these algorithms mimick the natural selection process, they are called genetic algorithms.

In GA, the solutions are represented in chromosome structure. In other words, the solutions are represented by combination of different sub-solutions which looks like a chain. For example, if in a problem it is required to find some values which can be thought as the solutions for n different subproblems $p_1, \ldots, p_n$ then a solution is represented as:

$$\text{(solution values for } p_1), \ldots, \text{(solution values for } p_n)$$

Generally, in timetabling problems the solutions are represented in two different ways:

1. Direct representation

2. Indirect (implicit) representation

As an example for direct representation, if there are four events e_1, e_2, e_3, e_4 where each event should have a solution that defines its starting time and place, then (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) can be a solution showing that event e_1 starts at time x_1 in classroom y_1, event e_2 starts at time x_2 in classroom y_2, etc.

In indirect representation, the values in the chromosomes represent the procedures that should be used to find value for each event. For example in i^{th} position, the value indicates a specific heuristic that should be used to find the value of the i^{th} event.

In GA, there are four main steps that should be taken at each iteration [13]. These are:

1. Evaluation
2. Selection
3. Breeding
4. Population update

Initially, there is a set of solutions. By using fitness function, the quality of these solutions are evaluated. The fitness function is formed by considering criterion that are important for the timetabler. After the evaluation of the solutions by the fitness function, some of the solutions will be more preferable than others. These solutions have higher probability to be selected. There are several ways for selection process such as selecting the solutions that are in the top predetermined percentage level. Then breeding stage starts. By this process, new solutions are generated. There are two main ways for generating new solutions: First, two or more of the selected solutions (parents) are combined by a cross over operation. This crossover operation will produce a new solution having solution values that are the combination of the selected solutions' values. Another way for generating a new solution can be modifying an existing solution. This is called *mutation.*

After breeding stage, the population of solutions is updated in a way that includes the new solutions and some or none of the previous solutions. These processes are repeated until obtaining good solutions with respect to the preference function.

APPENDIX C

Constraint Programming

Constraint Programming (CP) is a programming technique that tries to solve the problems by using partial information obtained from constraints. As it is understood from its name, it consists of computer implementation of the algorithms that are designed to solve specific type of problems. These algorithms can be implemented in a logic programming language such as PROLOG or more efficiently in a CLP language such as CHIP (Constraint Handling in Prolog) that is developed by modifying PROLOG to have extensive capabilities of solving constraints. However, this does not mean that CP cannot be implemented by using general purpose programming languages. As an example, ILOG Solver, that is used in this study, is a CP solver that consists of routines written in C++ [6].

The two major aspects of CP are:

1. Constraint Satisfaction Problems
2. Constraint Solving

In most of the problems that we face with, the solution requires to find out value assignments for the variables in a way that all the constraints are satisfied. Generally these problems can be represented as a Constraint Satisfaction Problem (CSP). A CSP consists of

- A finite set of variables,
- A finite set of values each variable can take, variable's domain,
- A set of constraints restricting the values that the variables can take simultaneously.

Constraint satisfaction algorithms deal with finding values for the variables in CSP such that all the constraints are satisfied. Since in real life most of the problems can be modeled by using finite set of variables with finite domain for each variable, constraint satisfaction can be used in combinatorial problems.

The second aspect of CP is constraint solving. In constraint solving, the variables may have infinite values in their domains and the constraints are more complicated such as there may be nonlinear equalities. Constraint solving algorithms use algebraic and numeric methods instead of combinations and search algorithms that are used in CSP [5].

The following is a brief overview of concepts and techniques used in CP (for a detailed discussion see [20]).

C.1 Consistency

Consistency is a deterministic technique used in solving CSP's. To use this technique, it is necessary to have binary constraints (constraints containing only two variables) or unary constraints (constraints containing a single variable). In this case, the problem can be modeled as a constraint graph in which the variables are represented by the nodes and the edges between the nodes represent the binary constraints.

As an example consider the following model :

$$x_1 > 3$$
$$x_1 + x_2 < 8$$
$$x_1, x_2 \in \{1, \ldots, 9\}$$

Then this model is represented by a constraint graph as seen in Figure B.1.

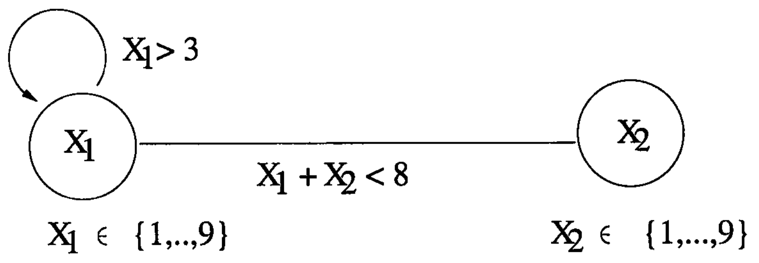


Figure B.1: Constraint Graph of the Model

There is two types of consistency:

1. Node Consistency
2. Arc Consistency

Node consistency deals with the domains of the variables represented by the nodes. A node is consistent, if the domain of the variable represented by that node has no value that does not satisfy the unary constraints containing that variable. The node consistent constraint graph of the above model can be seen in Figure B.2.

In a constraint graph, an arc (x_i, x_j) is consistent if every value v_i that is in the domain of $x_i, (v_i \in D_{x_i})$, then there exists a corresponding v_j value in the domain of variable x_j, $(v_j \in D_{x_j})$ for which the binary constraint represented by the edge (x_i, x_j) is satisfied for the value pair (v_i, v_j).

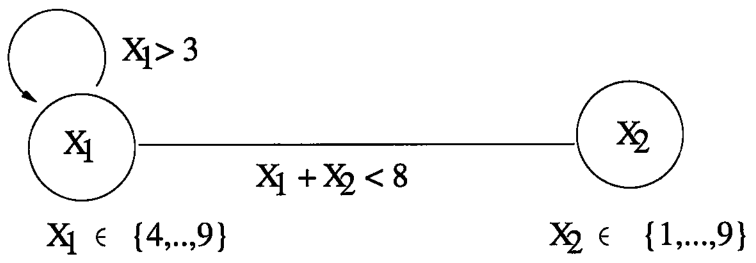


Figure B.2: Constraint Graph is Node Consistent

Then for the previous model given in Figure B.1, if arc (x_1, x_2) is consistent, then the value domains of the variables x_1 and x_2 will be as in Figure B.3.

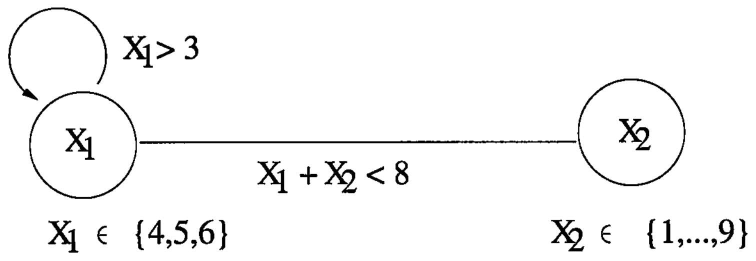


Figure B.3: Constraint Graph is Consistent for Arc (x_1, x_2) and Nodes x_1, x_2

Clearly arc consistency is directional; that is, if arc (x_1, x_2) is consistent then this does not mean that arc (x_2, x_1) is consistent. The arc consistent form of the graph can be seen in Figure B.4.

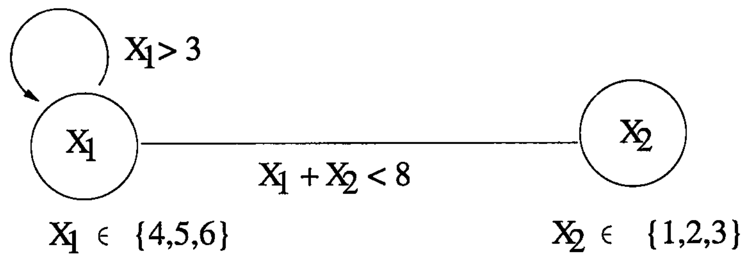


Figure B.4: Constraint Graph is Both Arc and Node Consistent

Related to consistency, another term is *K-consistency*. A constraint graph is *K-consistent* if for any of the $k - 1$ variables that are assigned to values which satisfy the constraints between those $k-1$ variables, then there exists a feasible value for any of the k^{th} variable chosen next for value assignment.

If for a given K, a constraint graph is *J-consistent*, for every J value satisfying the equation $J \leq K$, then this graph is called *strongly K-consistent.* Clearly having a *strongly N-consistent* constraint graph with N nodes means that a solution for the model represented by this constraint graph can be found without a search. However, worst-case behavior of turning a constraint graph with N nodes into *strongly N-consistent* form is exponential.

Although it is not efficient to solve CSP's by using consistency technique alone, it is still useful. Especially, search algorithms, which are explained in the next section, use it to prune the search space. During the search process, the conflicting values for the variables are detected by consistency technique and variables are initiated only to the consistent values. This is like generating new constraints by using the current ones. This called *constraint propagation.*

C.2 Search Algorithms

In order to solve a CSP, it is necessary to find different value assignments of the variables that satisfy the constraints. This value assignment process is specified by the search algorithm.

There are several search algorithms that may be used to solve CSP's. The simplest, but an inefficient one is generating all the value assignments of the variables and testing each value assignment whether it satisfies the constraints or not. This is known as *generate and test algorithm.* There are other, more efficient, algorithms such as *backtracking*, *forward checking*, *look ahead (MAC).*

C.2.1 Backtracking

Backtracking is a widely used search algorithm. The idea is instantiating values to the variables until finding value assignments for all the variables that satisfy all the constraints.

Initially one of the variables is instantiated to a specific value. At each step, if there are $k-1$ variables that are instantiated previously, then the next k^{th} variable is assigned to a value that is consistent with the previously instantiated $k-1$ variables. If it is not possible to find a consistent value for the k^{th} variable then the algorithm backtracks to the closest point where it can generate a new branch.

Backtracking algorithm is more efficient than the simple generate and test algorithm. However, the inconsistencies in value assignments can only be detected when they occur.

C.2.2 Forward Checking

Forward Checking is an improved version of simple backtracking. In forward checking, again initially a variable is instantiated to a value from its domain. Then repeatedly at each step, next variable is instantiated to a value that is consistent with the previous assignments. Different than backtracking, while assigning a value to the current variable, arc consistency between the current variable and the uninstantiated variables are maintained. By this way, current variable cannot take a value that causes an empty domain for one of the uninstantiated variables. If there is not such a value, then the algorithm backtracks to the point where it can start a new branch.

In backtracking, the inconsistencies are detected when they occur, however in forward checking it is possible to detect inconsistencies much earlier. On the other hand, forward checking does more computations compared to backtracking although it has a smaller search tree.

C.2.3 Look Ahead

Look Ahead or MAC (Maintaining Arc Consistency) is more involved than backtracking and forward checking algorithms. In backtracking, arc consistency is maintained only between the variable instantiated at that step and the previously instantiated variables. In forward checking, arc consistency is maintained between the instantiated variables and the uninstantiated variables. However, in look ahead algorithm arc consistency among the uninstantiated variables is maintained as well. This provides look ahead algorithm to detect inconsistencies between the future variables without assigning them any value. Consequently, it has smaller search trees compared to other search algorithms, but as an disadvantage, pruning search space more requires more computations which may cause look ahead algorithm to be inefficient.

C.3 Variable and Value Ordering

It is important to specify the way the search algorithm selects the variables for value assignment. The following are the strategies to achieve this.

C.3.1 Variable Ordering

Variable ordering is the strategy that specifies how the next variable is selected for value assignment at each step of the search algorithm. There is two types of variable ordering:

1. Static Ordering
2. Dynamic Ordering

In static variable ordering, the ordering of the variables for value assignment is known before the search process starts. In dynamic ordering, the candidate variable for value assignment is determined during the search process depending on the partial information at that step. For example, a dynamic variable ordering strategy may be selecting the next uninstantiated variable which has fewer values in its domain at each step. Since dynamic variable ordering requires modifications of variable domains at each step, it can be applied in forward checking or look ahead algorithms.

There are various strategies for variable ordering, but *first-fail* strategy is the most widely used one. This strategy requires to select the next variable that is more likely to fail, in other words the variable with the fewest number of values in its domain is selected. By this way, the branches can terminate at shorter depths and the size of the search tree becomes smaller. Then a feasible value assignment for all the variables can be obtained faster.

Another strategy for variable ordering may be selecting the variable that is involved in the largest number of constraints. In fact this strategy is like first-fail strategy. If a variable is involved in more constraints, then it becomes more difficult to find a value for that variable. By this strategy, again the difficult cases are handled first during search process and it results in the detection of inconsistencies earlier and a smaller sized search tree.

C.3.2 Value Ordering

In search process, after selecting the variable for value assignment, another question is about the value the current variable will take. The answer of this question depends on the value ordering strategy used by the search algorithm.

For example, there may be a strategy that initiates the variable to the value in its domain which is more likely to lead a solution. By this way, the probability of reaching a solution at the end of the current branch increases. However, if all the solutions are required or there are no feasible solution, value ordering strategy has no effect on solution time, since all the value assignments are tried.

APPENDIX D

Search Strategies

Search Strategy Used for Allocating Class Meetings to Days

```
1 For all the meetings that first and second year students attend and that
  are not scheduled yet
    1.1 Select the meeting with minimum domain size and minimum value in its domain
    1.2 While value domain of meeting is not empty do
        1.2.1 Let $m$ be the minimum value in meeting's domain
        1.2.2 Let start time of meeting be $m$
        1.2.3 If fail then delete the minimum value from meeting's domain and go to 1.2
        1.2.4 If there is no violation of constraint then go to 1

2 For all the meetings that third and fourth year students attend and that
  are not scheduled yet
    2.1 Select the meeting with minimum domain size and maximum value in its domain
    2.2 While value domain of meeting is not empty do
        2.2.1 Let $m$ be the maximum value in meeting's domain
        2.2.2 Let start time of meeting be $m$
    2.2.3 If fail then delete the maximum value from meeting's domain and go to 2.2
    2.2.4 If there is no violation of constraint then go to 2
```

Search Strategy Used for Construction of Daily Course Schedules

```
1 For all the meetings requiring *type two*, *type three* or *type four* classroom and
  that are not scheduled yet
  1.1 Select the meeting with minimum domain size and minimum value in its domain
  1.2 While value domain of meeting is not empty do
    1.2.1 Let $m$ be the minimum value in meeting's domain
    1.2.2 Let start time of meeting be $m$
    1.2.3 If fail then delete the minimum value from meeting's domain and go to 1.2
    1.2.4 If there is no violation of constraint then go to 1

2 For all the meetings requiring *type one* classroom and that are not scheduled yet
  2.1 Select the meeting with minimum domain size and maximum value in its domain
  2.2 While value domain of meeting is not empty do
    2.2.1 Let $m$ be the maximum value in meeting's domain
    2.2.2 Let start time of meeting be $m$
    2.2.3 If fail then delete the maximum value from meeting's domain and go to 2.2
    2.2.4 If there is no violation of constraint then go to 2
```

APPENDIX E

Mathematical Programming Models

Although CP models are used for constructing the weekly course schedule in the first two stages, it is possible to do this by mathematical programming. In this approach, the constraints become more complex compared to the declarative nature of the constraints in CP models.

E.1 MP Model for Allocation of Class Meetings to Days

Sets

S Set of class meetings, $\{m_1, \ldots, m_n\}$

G Set of student groups, $\{g_1, \ldots, g_k\}$

S_{g_i} Set of class meetings belonging to student group g_i

I Set of instructors, $\{I_1, \ldots, I_m\}$

S_{I_i} Set of class meetings given by instructor I_i

R Set of resources, $\{r_1, r_2, r_3, r_4\}$

S_{r_k} Set of class meetings that require resource r_k

P Set of class meeting pairs m_j, m_k that belong to the same section

Parameters

h_{m_j} Hours of class meeting m_j

l_{m_j} Resource requirement of class meeting m_j per day

c_{r_k} Maximum capacity of resource r_k per day

Variables

$$x(t, m_j) = \begin{cases} 1, & \text{if class meeting } m_j \text{ is assigned to time } t; \\ 0, & \text{otherwise.} \end{cases}$$

Constraints

1. All class meetings should be assigned to a day.
2. At each day, the total capacity requirement of the class meetings for a specific classroom type cannot exceed the available capacity of that classroom type (for classroom types -2, -3 and -4).
3. Class meetings that require *type-1* classroom can be placed in any type of classroom ensuring that the total daily classroom capacity is not exceeded.
4. Two class meetings belonging to the same course cannot assigned to consecutive days.

5. A student cannot attend class meetings for more than eight hours in a day.
6. An instructor cannot teach for more than five hours in a day.
7. All $x(m_j, t)$'s are binary variables.

These constraints can be written in mathematical form as:

(1) $\sum_{t=0}^{4} x(t, m_j) = 1,$ $\quad \forall m_j \in S$

(2) $\sum_{m_j \in S_{r_k}} x(t, m_j) \times l_{m_j} \leq c_{r_k},$ $\quad \forall r_k \in \{r_2, r_3, r_4\},\ t \in \{0, \ldots, 4\}$

(3) $\sum_{m_j \in S} x(t, m_j) \times l_{m_j} \leq \sum_{r_k \in R} c_{r_k},$ $\quad \forall t \in \{0, \ldots, 4\}$

(4) $\sum_{t=0}^{3} x(t, m_j) + x(t+1, m_j) +$
$x(t, m_k) + x(t+1, m_k) \leq 1,$ $\quad \forall (m_j, m_k) \in P$

(5) $\sum_{m_k \in S_{g_i}} x(t, m_k) \times h_{m_k} \leq 8,$ $\quad \forall g_i \in G,\ t \in \{0, \ldots, 4\}$

(6) $\sum_{m_k \in S_{I_i}} x(t, m_k) \times h_{m_k} \leq 5,$ $\quad \forall I_i \in I,\ t \in \{0, \ldots, 4\}$

(7) $x(t, m_j) \in \{0, 1\}$ $\quad \forall m_j \in S,\ t \in \{0, \ldots, 4\}.$

E.2 MP Model for Construction of Daily Course Schedules

Sets

S Set of class meetings, $\{m_1, \ldots, m_n\}$

I Set of instructors, $\{I_1, \ldots, I_m\}$

S_{I_i} Set of class meetings given by instructor I_i

G Set of student groups, $\{g_1, \ldots, g_k\}$

S_{g_i} Set of class meetings belonging to student group g_i

R Set of resources, $\{r_1, r_2, r_3, r_4\}$

S_{r_k} Set of class meetings that require resource r_k

H_i Set of class meetings that are i hour length ($i \in \{1, 2, 3\}$)

Parameters

h_{m_j} Hours of class meeting m_j

l_{m_j} Resource requirement of class meeting m_j per hour

c_{r_k} Maximum capacity of resource r_k per hour

Variables

$$x(t, m_j) = \begin{cases} 1, & \text{if meeting } m_j \text{ starts at time } t; \\ 0, & \text{otherwise.} \end{cases}$$

Constraints

1. All class meetings should be assigned to a time slot.

2. At each time slot, the total capacity requirement of the class meetings for a specific classroom type cannot exceed the available capacity of that classroom type (For classroom types-2, -3 and -4)

3. Class meetings that require *type-1* classroom can be placed in any type of classroom ensuring that the total classroom capacity at that time slot is not exceeded.
4. An instructor cannot teach more than one class meeting at a time slot
5. A student cannot attend more than one class meeting at a time slot
6. All $x(m_j, t)$'s are binary variables

These constraints can be written in mathematical form as:

(1) $\sum_{t=0}^{8} x(t, m_j) = 1, \quad \forall m_j \in H_1$

$\sum_{t=0}^{7} x(t, m_j) = 1, \quad \forall m_j \in H_2$

$\sum_{t=0}^{6} x(t, m_j) = 1, \quad \forall m_j \in H_3$

(2) $\sum_{m_j \in S_{r_k}} x(0, m_j) \times l_{m_j} \leq c_{r_k}, \quad \forall r_k \in \{r_2, r_3, r_4\}$

$$\begin{aligned}&\textstyle\sum_{m_j \in (S_{r_k} \cap H1)} x(1, m_j) \times l_{m_j} + \\ &\textstyle\sum_{m_j \in (S_{r_k} \cap H2)} (x(1, m_j) + x(0, m_j)) \times l_{m_j} + \\ &\textstyle\sum_{m_j \in (S_{r_k} \cap H3)} (x(1, m_j) + x(0, m_j)) \times l_{m_j} \leq c_{r_k}, \qquad \forall r_k \in \{r_2, r_3, r_4\}\end{aligned}$$

$$\begin{aligned}&\textstyle\sum_{m_j \in (S_{r_k} \cap H1)} x(t, m_j) \times l_{m_j} + \\ &\textstyle\sum_{m_j \in (S_{r_k} \cap H2)} (x(t, m_j) + x(t-1, m_j)) \times l_{m_j} + \\ &\textstyle\sum_{m_j \in (S_{r_k} \cap H3)} (x(t, m_j) + x(t-1, m_j) + \qquad \forall r_k \in \{r_2, r_3, r_4\}, \\ &x(t-2, m_j)) \times l_{m_j} \leq c_{r_k}, \qquad t \in \{2, \ldots, 8\}\end{aligned}$$

(3) $\sum_{m_j \in S} x(0, m_j) \times l_{m_j} \leq \sum_{r_k \in R} c_{r_k},$

$$\begin{aligned}
&\textstyle\sum_{m_j \in H1} x(1, m_j) \times l_{m_j} + \\
&\textstyle\sum_{m_j \in H2} (x(1, m_j) + x(0, m_j)) \times l_{m_j} + \\
&\textstyle\sum_{m_j \in H3} (x(1, m_j) + x(0, m_j)) \times l_{m_j} \leq \sum_{r_k \in R} c_{r_k},
\end{aligned}$$

$$\begin{aligned}
&\textstyle\sum_{m_j \in H1} x(t, m_j) \times l_{m_j} + \\
&\textstyle\sum_{m_j \in H2} (x(t, m_j) + x(t-1, m_j)) \times l_{m_j} + \\
&\textstyle\sum_{m_j \in H3} (x(t, m_j) + x(t-1, m_j) \\
&+ x(t-2, m_j)) \times l_{m_j} \leq \sum_{r_k \in R} c_{r_k}, && \forall t \in \{2, \ldots, 8\}
\end{aligned}$$

$$\text{(4)} \quad \textstyle\sum_{m_j \in S_{I_i}} x(0, m_j) \leq 1, \qquad \forall I_i \in I$$

$$\begin{aligned}
&\textstyle\sum_{m_j \in (S_{I_i} \cap H1)} x(1, m_j) + \\
&\textstyle\sum_{m_j \in (S_{I_i} \cap H2)} (x(1, m_j) + x(0, m_j)) + \\
&\textstyle\sum_{m_j \in (S_{I_i} \cap H3)} (x(1, m_j) + x(0, m_j)) \leq 1, && \forall I_i \in I
\end{aligned}$$

$$\begin{aligned}
&\textstyle\sum_{m_j \in (S_{I_i} \cap H1)} x(t, m_j) + \\
&\textstyle\sum_{m_j \in (S_{I_i} \cap H2)} (x(t, m_j) + x(t-1, m_j)) + \\
&\textstyle\sum_{m_j \in (S_{I_i} \cap H3)} (x(t, m_j) + x(t-1, m_j) + x(t-2, m_j)) \leq 1, && \forall I_i \in I, t \in \{2, \ldots, 8\}
\end{aligned}$$

$$\text{(5)} \quad \textstyle\sum_{m_j \in S_{g_i}} x(0, m_j) \leq 1, \qquad \forall g_i \in G$$

$$\begin{aligned}
&\textstyle\sum_{m_j \in (S_{g_i} \cap H1)} x(1, m_j) + \\
&\textstyle\sum_{m_j \in (S_{g_i} \cap H2)} (x(1, m_j) + x(0, m_j)) + \\
&\textstyle\sum_{m_j \in (S_{g_i} \cap H3)} (x(1, m_j) + x(0, m_j)) \leq 1, && \forall g_i \in G
\end{aligned}$$

$$\begin{aligned}
&\textstyle\sum_{m_j \in (S_{g_i} \cap H1)} x(t, m_j) + \\
&\textstyle\sum_{m_j \in (S_{g_i} \cap H2)} (x(t, m_j) + x(t-1, m_j)) + \\
&\textstyle\sum_{m_j \in (S_{g_i} \cap H3)} (x(t, m_j) + x(t-1, m_j) + x(t-2, m_j)) \leq 1, && \forall g_i \in G,\ t \in \{2, \ldots, 8\}
\end{aligned}$$

$$\text{(6)} \quad x(t, m_j) \in \{0, 1\} \qquad \forall m_j \in S, t \in \{0, \ldots, 8\}.$$