THE UNIVERSITY OF THE SOUTH PACIFIC
LIBRARY

DIGITAL THESES PROJECT

Author Statement of Accessibility- Part 2- Permission for Internet Access

Name of Candidate	: ATISH CHAND
Degree	: MSc
Department/School	: Computing Information and Mathematical Science Faculty of Science and Technology, USP.
Institution/University	:
Thesis Title	: a constraint directed Reasoning System for University Timetabling.
Date of completion of requirements for award	: 15/11/05

1. I authorise the University to make this thesis available on the Internet for access by USP authorised users. Yes/~~No~~

2. I authorise the University to make this thesis available on the Internet under the International digital theses project Yes/~~No~~

Signed: [illegible]

Date: 2/12/2005

Contact Address	Permanent Address
22B Catalina Dr	same as Contact.
USP	
Laucala	
e-mail : chand_at@usp.ac.fj	e-mail: chand_at@usp.ac.fj

July 2004

EF

Rec 23/8/13
SP

Q 340.C532 2005 80%

A Constraint Directed Reasoning System for University Timetabling

MSc Thesis by Atish Chand,

In partial fulfilment of the requirements for the degree of Master of Science

School of Computing , Information and Mathematical Sciences

Faculty of Science and Technology

The University of the South Pacific

November 2005

Declaration of originality

I, Atish Chand, declare that this thesis is my own work and that, to the best of my knowledge, it contains no material previously published, or substantially overlapping with material for the award of any other degree at any institution, except where due acknowledgement is made in the text.

Signed: .

Date: 23rd November 2005

Acknowledgements

Dr Dharmendra Sharma (Department of Mathematics and Computing Science USP, till 2000) for introducing me to timetabling research and to the International Conference on The Practice and Theory of Automated Timetabling. Dr John Hosack (Department of Mathematics and Computing Science USP, till 2004) for sharing his experienced knowledge in timetabling, Dr Arvind Patel (Department of Accounting and Financial Management USP) for his valuable time, advice and seasoned guidance. Dr Zhenquan Li (Department of Mathematics and Computing Science USP) for assisting with his valuable contributions.

The University of the South Pacific for sponsoring my post-graduate studies.

The Acting Head of School, School of Pure and Applied Science (SPAS) at the University of the South Pacific and the SPAS Research Committee for funds for attending conferences to present papers based on this thesis.

Dr Jito Vanualailai, Dr Robin Havea, Mr Bibhya Sharma (Department of Mathematics and Computing Science USP) and mutual friend Mr Sunil Chand (MBA Programme USP) for their morale support and encouragement.

Dedication

To all those who are immortal in time

List of Publications from this thesis:

Chand, A, A Heuristic Approach To Constraint Optimization In Timetabling, 2002 South Pacific Journal Of Natural Science Vol 20. School of Pure and Applied Science, The University of The South Pacific

Chand A, A Heuristic Approach to Examination Timetabling, New Zealand Mathematics Colloquium, 3 - 6 December 2001, Massey University

Chand A, A Constraint Based Generic Model for Representing Complete Timetabling Data" Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. Aug 2004, Carnegie Mellon University, Pittsburgh, USA

Abstract

Timetabling is a difficult NP-complete problem and belongs to a general class of problems known as scheduling. It has been difficult to develop a generic solution for timetabling due to a large variety of constraints typical in different timetabling environments. Additionally, it is difficult to compare timetabling algorithms due to the variety of constraints and data formats. Determining which timetabling resources and constraints are more constrained and restrictive than others is a universal problem. It is difficult to compare constraints and hence determine which constraints are more restrictive than others since the variety of constraints differ in nature. This thesis proposes a generic computational model for university timetabling for predefined constraints found in the problem, and a generic heuristic algorithm used to develop an acceptable solution. It shows that the reported wide variety of constraints can be modelled as smaller sets of constraints that can be easily compared with each other. A university examination timetabling problem is used to illustrate and test the model.

Table of Contents

Chapter 1 INTRODUCTION

This chapter discusses the nature of the general scheduling problems and general timetabling problem. Chapter 2 reviews previous and current work in timetabling, identifies current problems and discuses techniques of solving the timetabling problem discussed. Work on a standard timetabling language is reviewed in Chapter 3. The nature of various timetabling constraints is discussed in Chapter 4 where it is shown that the differing constraints are the essentially similar and how timetabling resources can be specified as constraints. A generic constraint based entity relationship model for specifying complete university timetabling problems is proposed in Chapter 5. A constraint based heuristic algorithm is discussed in Chapter 6. In Chapter 7, the model is implemented for the examination timetabling problem at the University of the South Pacific (USP). A heuristically guided algorithm combined with a sequential search is applied to the USP instance. The examination timetable for semester 1, 1999 is built by the automated system and compared to the manually prepared timetable and another automated timetabling solution built by a different system. Chapter 8 summarizes the thesis and concludes with a discussion on future research.

1.1 THE GENERAL SCHEDULING PROBLEM

Timetabling belongs to a class of general scheduling problem. Weare (1995) describes the general scheduling problem as one being where a set of processes have to be carried out for each member of a set of objects (resources). The general scheduling problem is shown in *Figure 1-1*.

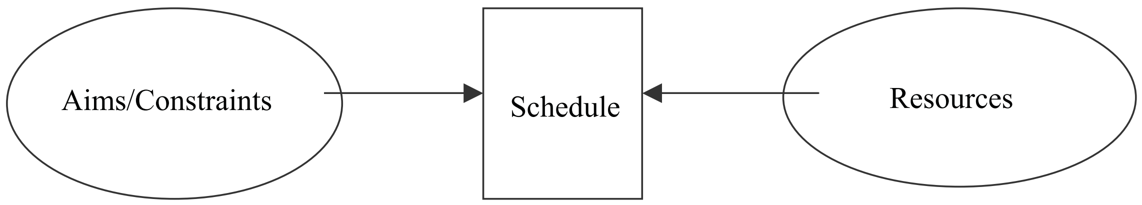


Figure 1-1 The General Scheduling Problem

Resources include any number of objects that are grouped according to type, for example, a type of resource in a school timetabling problem is rooms. Similarly, resources in other scheduling problems could be trains, aeroplanes, machines, teachers, rooms, slide projectors or a host of other objects. Constraints are expressions that indicate the desired relationships between resources, within a grouping of resources, and desired relationships between groupings.

Examples of general scheduling problems are making a duty roster, project schedules, bus scheduling, air flight schedules, etc including the well-known Travelling Salesman Problem (Lawler, E., *et al*., 1985). In this problem, the task is to visit a number of cities in the shortest route possible.

If there are 40 cities in a location, then starting from the first city, there will be a choice of 39 cities to visit next, a choice of 38 cities for the third and so on. Thus, the number of possible routes is 40 factorial (40!) that is approximately equal to 8×10^{47}.

One could do a simple comparison of each route to determine which is the shortest. However, the enormous number of possible routes makes this simple method very time-consuming and impractical. Assuming a computer could examine one billion routes in one second (that is if a 1GHz computer could examine one route per CPU cycle) then it would be possible to examine all routes in 8.15915×10^{47} seconds (9.44346×10^{33} days). Such a simple method is not viable when time is limited. Other strategies are needed to build a timetable in reasonable time.

Similarly, in examination timetabling, there are many choices available for possible exam time and venue allocations. Timetabling has been shown to be an NP-complete problem (Cooper, T. B., Kingston, J. H., 1995). Any problem takes some computational time to solve. In scheduling problems, the computational time can be expressed as a function of the resources and constraints involved. As the size of the problem, that is the number of variables and resources, increases, the computational time may increase polynomially or exponentially. Polynomial problems are relatively easy to solve as the computational time required increases in linear or some higher order polynomial function. However, the time required for exponential problems increases very rapidly as the problem size increases. Special cases of exponential problems are called *NP-complete* or *completely non-deterministic polynomial.* These problems are much harder to solve than regular exponential problems (Lawler *et al* 1985). So-called easy, or tractable problems can be solved by computer algorithms that run in polynomial time. For a problem of size *n*, the time or number of steps needed to find the solution is a polynomial function of *n*. Algorithms for solving hard, or intractable, problems require times that are exponential functions of the problem size *n*. Polynomial-time algorithms are considered efficient, while exponential-time algorithms are considered inefficient, because the execution times of the latter grow much more rapidly as the problem size increases.

A problem is called NP (nondeterministic polynomial) if its solution (if one exists) can be guessed and verified in polynomial time, nondeterministic means that no particular rule is followed to make the guess. If a problem is NP and all other NP problems are polynomial-time reducible to it, the problem is NP-complete. Thus, finding an efficient algorithm for any NP-complete problem implies that an efficient algorithm can be found for all such problems, since any problem belonging to this class can be recast into any other member of the class. It is unknown whether any polynomial-time algorithms will ever be found for NP-complete problems, and determining whether these problems are tractable or intractable remains one of the most important questions in theoretical computer science. When an NP-complete problem must be solved, one approach is to use a polynomial algorithm to approximate the solution; the answer thus obtained will not necessarily be optimal but will be reasonably close.

In timetabling problems, there are many different ways in which each session can be allocated a certain time and room. If *s* is the number of sessions to be timetabled, *t* is the number of timeslices and *r* the number of rooms available for timetabling, then the number timetables that can be built is *rt* factorial *(rt!)* In a small timetabling environment such as that of The University of The South Pacific, with only 20 timeslices for examination timetabling and 10 rooms, the number of different timetables possible is 200 factorial (200! more than 7×10^{306}). As in the case of the travelling salesperson problem, if one billion timetables could be examined in one second to determine each timetable satisfies the requirements, then it would take more then

2.3×10^{288} centuries to examine each possible timetable combination. Therefore, other strategies are needed that do not examine all possible timetables but use some other methodology to build acceptable timetables within reasonable time.

The proposed constraint language model allows the timetabling resources and constraints to be generically specified so that different algorithms for solving the timetable problem can be applied to the model and compared for constraint satisfaction and efficiency. The proposed model is easy to implement in a relational database management system and hence does not require additional expensive software. The amount of time required to train users is decreased, as existing systems will be used for implementation. The proposed model is generic enough to allow users to implement it in different timetabling environments with little or no change thus saving program development time. The next section discusses in detail the university timetabling problem.

1.2 THE UNIVERSITY TIMETABLING PROBLEM

Wren (1995) describes timetabling as “the allocation, subject to constraints, of given resources to objects being placed in space-time in such a way as to satisfy as nearly as possible a set of desirable objectives”. Weare (1995) also defines a timetable or schedule:

"A timetable is a description of the movement and grouping or resources over time, often to achieve a certain aim or aims and/or to satisfy a set of constraints."

According to Weare (1995), "A feasible timetable, or schedule, is one that satisfies its associated set of constraints". Such constraints might include groupings that must occur at some point in time; groupings that may never occur; groupings that must be sequenced as required, groupings that must have particular quantities of particular resources.

The timetable can be represented as a sequence of bins or slots (of predefined fixed duration) arranged against time and each of the bins contains objects and resources. A solution to a timetabling problem is such a sequence of bins that satisfy a given set of hard and soft constraints. Hard constraints are those constraints that have to be satisfied, and soft constraints are those that can be relaxed.

The university timetabling process entails different sub-processes as described below. It will be shown which of these problems is pure timetabling, how the sub-problems are in essence the same and how the proposed constraint model can be applied to each sub problem.

FACULTY SCHEDULING

A set of instructors, a set of lessons, and availability of instructors to teach lessons are defined. The problem is to distribute lessons between the instructors under specified conditions. In university timetabling, this is most often pre-assigned by the faculty management and the faculty allocations do not need to be calculated by the timetabling system.

COURSE SCHEDULING

Each student selects a combination of courses from a set of courses. Lectures must be scheduled in such a way such that certain constraints are satisfied. One of the main constraints affecting students is that no student should be required to attend more than one lecture at the same time. There are two approaches to this problem. The first approach involves the case where the student enrolments for courses are not known beforehand. Students are given the timetable before enrolment and they select courses such that they do not have clashes. This approach considers the course co-requisites and popular combination of courses and ensures that these courses are not scheduled for the same time. The second approach is to consider the course enrolment for the current timetable problem and ensure that courses with overlapping enrolled students are not timetabled at the same time. A hybrid approach is to combine the first approach with course-pre-enrolment and/or previous enrolment trends. The first approach is widely practiced at universities as the second approach invites problems such as students contravening regulations. For example, a student may enrol for two courses, one of which is the pre-requisite of the other, or delay enrolling for a non-compulsory first-year course until the final-year of the programme of study.

EXAMINATION TIMETABLING

For a given set of students and examinations for each student, each examination must be allocated a period such that certain constraints are satisfied. The most important constraints are that *no student should have more than one examination scheduled at the same time; no student should have two or more examinations scheduled consecutively on the same day.* Large halls are usually used to conduct exams. Each hall may accommodate more than one exam. Venues for very large exams may be split into more than one hall. Part of the examination timetabling problem is the invigilator assignment problem. As in the faculty scheduling problem, the invigilator assignment is usually pre-assigned. Course coordinators and lecturers usually supervise their own courses and perhaps help supervise other courses in the same or similar disciplines.

CLASSROOM ASSIGNMENT

When student-lecturer sessions or examinations are scheduled, rooms need to be assigned to each session. The effect of room assignment on timetabling problems depends on the number and size of rooms available. The constraint imposed on room assignment can, technically, be relaxed or removed by the physical construction of more rooms. In contrast, the time constraints cannot be relaxed by the creation of time, as time cannot be created. More time can be obtained by extending the timetabling period, however, there are typically constraints that prevent this from happening, such as *one semester comprises 14 weeks, there being 5 teaching days in a week.*

Other academic problems may be considered together with timetabling. Student course counselling, curriculum development and physical infrastructure and equipment planning feed data into a timetabling system and use information provided. These problems indirectly or directly affect the final timetable. They give rise to a variety of constraints that differ from one timetabling instance to another. This thesis seeks to show that the actual timetabling problem can be isolated from the other problems and solved independently. Problems in timetabling are

- Due to the varieties of timetabling instances, the timetabling data is structured differently from one instance to another, and hence it is difficult to compare timetabling methods. It is important to be able to compare timetabling methods so that the better methods can be utilised and further developed and the weaker methods can be improved.
- The various timetabling environments have a wide range of differing constraints. Current timetabling methods model constraints separately for the timetabling data. Due to the varieties of timetabling constraints, the current models do not completely specify the constraints. As the underlying data structure differs from one timetabling instance to another it is difficult to compare timetabling methods.

1.3 THE UNIVERSITY TIMETABLING RESEARCH QUESTIONS

The research questions specifically answered in this thesis are:

- Is it possible to standardise the university timetabling data format such that it is applicable to all reasonable instances of both classroom and examination timetabling problems?
- Is it possible to standardise university timetabling constraints such that all reasonable constraints in one timetabling instance can be expressed in any other timetabling instance?
- Can a generic model of the university timetabling problem that incorporates both the timetabling data and constraints be developed?
- Can a timetabling algorithm be successfully applied to such a generic model?

In answering these questions, a generic model of timetabling data and constraints will be developed and the model will be translated into an Entity Relationship Model (ERM). The ERM will then be implemented in a relational database and embedded SQL used to allow a timetabling method to build a timetable that will be compared to the timetable that is manually built, and to a timetable built by another automated model.

1.4 SUMMARY

University timetabling is part of a larger group of scheduling problems that involve sequencing groupings of resources and objects such that certain objectives are achieved. In university timetabling, the key resources are time, teaching sessions, rooms, students and staff. The objective is to satisfy constraints expressed over the resources. It is difficult to compare constraints and timetabling solutions due to a large variety of constraints and data formats in different timetabling environments. This thesis proposes a generic constraint based model that simplifies the variety of constraints and data formats, allowing different timetabling constraints, algorithms and solutions to be easily compared. In Chapter 2, recent directions in timetabling algorithms are discussed. Chapter 3 discusses the requirements of a standard timetabling data format and describes some recently proposed timetabling data formats. Chapter 4 shows how the various timetabling constraints can be generalized. Chapter 5 proposes a generic entity relational model for university timetabling. Chapter 6 describes a constraint based heuristic algorithm that is integrated with the generic ERM to form COMBAT, a constrained based reasoning system for university timetabling. Chapter 7 analyses the application of the generic ERM and heuristic algorithm to a university timetabling instance at the University of The South Pacific. It compares the timetable solutions built by the manual system at USP and with an automated system, OPTIME developed at the University of Nottingham. The thesis concludes with a summary in Chapter 8 together with suggestions for further work.

Chapter 2 TIMETABLING ALGORITHMS

There has been recently increasing academic interest in timetabling leading to the formation of the International Conference on the Practice and Theory of Automated Timetabling (ICPATAT) in 1995. A good proportion of the literature that was surveyed is from the proceedings of that ICPATAT and those in subsequent years. Automated timetabling algorithms that are currently used or proposed include: network flow techniques (de Werra 1971), graph colouring (Welsh *et al*, 1996), integer programming (Lawrie 1969), direct heuristics (Burke, *et al* 1995, Barham A, *et al* 1978), simulated annealing (Abramson D, *et al,* 1978), tabu search (Hertz A 1992), neural networks (Kovacic M, 1993), genetic algorithms (Burke *et al* 1994, Corne *et al* 1994), memetic algorithms, logic programming (Kan *et al* 1992, Fahrion *et al* 1992), expert systems (Guides *et* al, 1990) and constraint satisfaction logic programming.(Guret C, *et al*, 1995, Reis L, *et al*, 1999). Surveys of recent developments in university timetabling are available in Burke *et al* (2002), Carter M, *et* al (1995) and Bardadym A, (1995). This chapter briefly examines major and new algorithmic methods of solving the timetabling problem. Details can be obtained in the literature quoted.

2.1 GRAPH COLOURING

In graph colouring (Welsh *et al,* 1996), each session to be scheduled is represented as a vertex. Sessions, which should not be scheduled simultaneously, are connected by edges. The problem now becomes how to colour the graph such that no two adjacent vertices are coloured the same while the number of colours used is minimised.

The difficulty in graph colouring is how to incorporate the various constraints. For example, an edge between two vertices represents two sessions that must not be scheduled simultaneously. Therefore, it becomes very difficult to specify the case where two sessions are combined and taken by the same supervisor because the semantic of any two vertices not connected together means that they may be timetabled at the same time but not that they must be timetabled at the same time.

Moreover, important information regarding constraints can be lost. For example, two adjacent vertices may mean that the sessions are taken by the same lecturer or they have a set of common students. It also becomes difficult to specify that sessions with the same lecturer must absolutely be not scheduled at the same time, while sessions with a small set of common students may be held at the same time but preferably not.

A simple manual derivative of the graph method is used in primary and high schools in Fiji where each teacher is associated with a separate colour. Each teacher is then issued a number of coloured tokens equal to the number of teaching sessions. A blank timetable is represented as a table with subjects as columns and time as rows. The basic problem is to allocate the coloured tokens to the timetable such that no row has two or more tokens of the same colour.

2.2 LOCAL & TABU SEARCH

Local search methods are optimization techniques based on the concept of neighbours. (Hertz A 1992). Given an optimization problem P and an objective function f which is to be minimized, a function N is assigned to each feasible solution s, its neighbourhood solutions N(s). An initial solution S_o is selected from the search space S and its neighbouring solutions are examined. Two techniques of examining the neighbourhood are the steepest descendent method (SDM) and the randomized descendent method (RDM).

Steepest Descendent Method

In SDM, all neighbourhood solutions are examined and the one giving the best improvement is accepted. SDM stops when a local minimum is reached.

The major problems with SDM are the number of neighbouring solutions to be examined and being stuck in a local minimum that is not the global minimum.

Randomized Descendent Method

In RDM, a neighbour is chosen at random and it is accepted if it improves the value of the function *f*. RDM stops after a fixed number of random iterations that do not improve the value of function *f* any further. RDM allows the possibility of being stuck in a local minimum and hence stop at a non-minimum.

Non-ascendant Method

Another technique, the non-ascendant method combined with either SDM or RDM incorporates sideways moves allowing the search to move over a false minimum (a plateau with solutions that give equal values to function *f*) to another true minimum.

Starting from an initial solution S*o*, the tabu search (TS) repeatedly searches the neighbourhood of the current solution s. The solution which gives the minimum value of the function *f* is chosen as the new current solution regardless of whether it is better or worse (depending on the criteria that specify the acceptability of the solution) than the current solution. In order to prevent cycling a list of tabu (forbidden) moves is kept. The list is the last k moves in reverse, stored as a push and pop queue of fixed size. When the previous move is pushed to the bottom of the queue, the oldest move is popped from the top.

2.3 SIMULATED ANNEALING

In simulated annealing, (Abramson D, *et al,* 1978) the randomly generated neighbour is always accepted if it is equal or better than the current solution or if it is an upward move with a better probability (in an effort to move out of local minima). The probability depends on the function *f* and on a parameter called the temperature that decreases with the number of iterations.

2.4 GENETIC ALGORITHM

In the evolution of living organisms, an initial population undergoes reproduction and mutations in a given environment. Mutated populations that are more suitable to the environment thrive better than those with less suitable mutations do. These unsuitable populations dwindle in numbers, and hence after a period or certain generations of mutations, the more suitable populations survive.

Genetic algorithms (Burke *et al* 1994, Corne *et* al 1994) try to simulate the above behaviour. A timetable is represented by a chromosome whose length is same as the number of variables (exams or course) that need to be timetabled. Each piece of the chromosome contains genes that represent the timetable values (venue, time) that must be assigned to the variable.

Mutation is represented by an operator that randomly changes the genes (timetable values) for a chromosome piece. Reproduction is represented by a cross-over operator that switches corresponding pairs of chromosome pieces.

The environment for the population is simulated by an objective function that inspects each mutation and reproduction, and allows those to take place that improve the timetable.

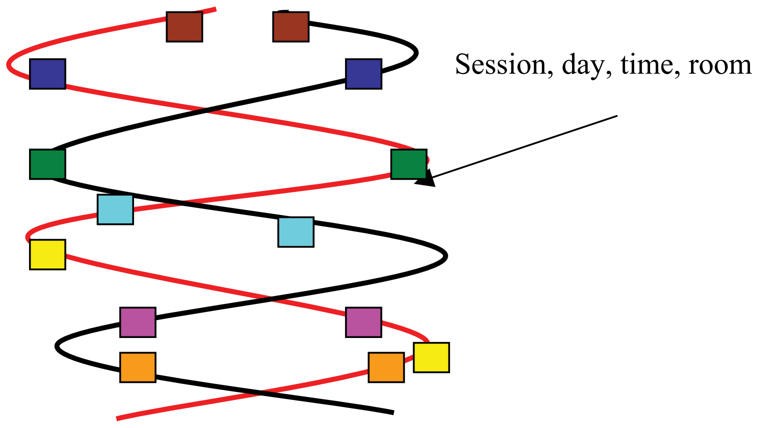


Figure 2-1 Genetic Evolution of a Timetable

In the above *Figure 2-1,* the spiral lines represent two timetables. The shaded boxes represent sessions of courses that have to be timetabled. Boxes of the same colour represent the same sessions. Each box carries data about the session code, and the day, time and room to which it is allocated.

Mutations take place within a box by changes in the data values. Reproduction involves switching of boxes of the same colour. The crossover and mutations are controlled by complex functions.

2.5 CONSTRAINT LOGIC

In logic programming, a variable is unified with a value. In constraint satisfaction, a variable is associated with a domain of values, and constraints are used to remove inconsistent values.

The overall structure of a constraint satisfaction problem is:

i. State domain of variables
ii. State constraints
iii. Generate values

Successful solution of the timetabling problem requires the generation of a timetable that satisfies certain constraints. The disadvantage of constraint logic or constraint satisfaction approaches to timetabling is that, in real world timetabling, there are a large number of constraints and hence such algorithms would run out of time. However, this thesis proposes a constraint model that shows that the variety of constraints are actually due to the human perception of constraints and that in concept, the various constraints are the same. It then combines the constraint satisfaction approach with a generic constraint model of the timetabling problem and applies an enhanced heuristic algorithm to solve the problem.

2.6 SUMMARY

Various algorithms have been used to solve timetabling problems. The algorithms act on computational models that are different for the various algorithms. In the models studied, only the timetabling resources are modelled but not the constraints. In those that do, representing the constraints is difficult. There exists a need for a standard timetabling model that incorporates both the resources as well as constraints, that can be subjected to different algorithms. The standard timetabling data format requirements are discussed in the next chapter.

Chapter 3 TIMETABLING DATA FORMATS

(Note: parts of this chapter are based on a paper by the author A. Chand "A Constraint Based Generic Model for Specifying Complete University Timetabling Data" accepted for publication in "The Proceedings of the 5th International Conference on the Theory and Practice of Automated Timetabling" held at Pittsburgh USA Aug 8-12th 2004).

Currently, no universally accepted model exists for describing timetabling problems. Efforts have been made towards finding a standard information format for timetabling problems. Although some data formats have been developed for representing different timetabling problems, they are usually incomplete in some aspect. This chapter examines the requirements of standard timetabling data format. Various timetabling data formats that have been previously proposed are analysed for their strengths and limitations.

3.1 REQUIREMENTS OF A STANDARD DATA FORMAT

Burke *et al* (1998) have stated three requirements of a standard format – generality, completeness and practicability.

Generality requires that a standard format be able to express any reasonable instance of the timetabling problem. If a timetable instance cannot be stated in the standard format, then it would be unacceptable by users of that timetabling instance. This thesis proposes a data format based on the relational model that is widely prevalent, thus there is little need to learn a new language. The constraints are modelled together with the data instead of being treated separately. This allows the entire timetabling problem to be specified by the same model adding uniformity.

Completeness requires that it must be possible to express an instance completely. This includes straightforward data expressing the resources and sessions, complex logical constraints and proposed solutions including hard and soft constraint weightings. The soft constraint weights are important in order to determine the relative quality of proposed solutions. Hard constraints must be satisfied. Hence, a candidate solution that violates any hard constraint is rejected outright.

This thesis elaborates on numerous examples of common timetabling constraints and explicitly shows how they can be modelled completely. This requirement allows the model to serve as benchmark for comparing different timetabling algorithms.

The third requirement is that the translation of data between the existing format and the standard format should be practicable.

Burke *et al* (1998) have proposed a data format based on set theory and logic, which requires the learning of a new language. Since there are a wide variety of constraints that differ from one timetabling instance to another, constraints are specified in a separate library. Burke *et al* (1998) state “The types of problems tackled by researchers are often so different that interchange of data is inherently impossible. These distinct types of problems will need distinct incompatible library files”.

The model proposed in this thesis does not require a separate library files for different timetabling instances and hence overcomes the difficulty in exchange of data.

There are many other formal specifications for timetabling including concerns for minimising data storage space and facilitating fast data processing. The proposed model tries to compromise between generality and simplicity. It is sufficiently general to allow representation of the most common variations of timetabling problems, including school, university and examination timetabling and almost all other less common constraints. The proposed model exploits the ER Model that is widely popular for its simplicity. After a series of interviews with timetabling experts, Reis and Oliveira (1999) listed the following main requirements associated with the needed model

i. It should be independent of implementation details and timetabling strategies;

ii. It should be easy to extend, including new concepts and constraints;
iii. Existing benchmark problems should be easily translated to the new formalism;
iv. The new formalism should be compatible with most timetabling systems and easily readable by the human user;
v. Information not directly concerned with the scheduling problem should not be included in the formalism. Therefore, common information regarding school administration, like student names and addresses, is omitted;
vi. The formalism should be general enough to enable the representation of school timetabling, university timetabling and exam timetabling problems;
vii. It should be suitable for representing complete problems and simple related sub-problems (like staff or invigilator assignment, section definition and room assignment)
viii. It should include a clear definition of all the constraints (both hard and soft) associated with the problems;
ix. It should be possible to represent both incomplete and complete solutions;
x. A timetabling quality evaluation function should be easy to include, enabling to evaluate directly any proposed solution;
xi. It should be as concise as possible;
xii. It should be robust, enabling simple data validation and override of common errors.

Compared with Burke's (1997) broad classification of the requirements, items *i, ii, vi* belong to the generality requirement, *ii* (overlapping with generality), *vii, viii, ix* and *x* belong to the completeness requirement while *iii, iv, v, xi* and *xii* are of practicability requirements. It should be noted that the requirement *v* is for the *scheduling* problem rather than *timetabling* and conflicts with requirement *vii* which attempts to include sub-problems. It will be shown that requirement *vi* need not be fulfilled as part of the timetabling problem but can be build as an add-on between the data model and the user interface. As mentioned in Chapter one, time is the one commodity that cannot be created as opposed to staff, invigilators, rooms and other resources, and hence this thesis concentrates on the fundamental problem of *time*tabling rather than administrative problems.

3.2 KEYWORDS

Many terms have been used to describe the various elements of the university timetabling problem. Some terms have common meaning while other terms have been given different or slightly different meanings by different researchers. The variety of names for different timetabling objects adds further confusion to the seeming myriads of constraints. Hence, timetabling terminology used in this study is described next. The choice of the names is dictated first by their everyday common meaning, and secondly to avoid ambiguity with other studies where the same word may be used in a different sense.

Reis and Oliveira (1999) have suggested a verbose model that will allow administrators to better specify their needs. In doing so, they have suggested synonyms for key concept words.

Example of a brief list of synonyms for the keywords used in the language.

```
[Default | All | Every | Any], [Year | Problem | File],  Schedule | Timetable],
[Event | Module | Lecture | Lesson | Exam | Examination | Tutorial],
[Teacher | Teachers | Invigilator | Supervisor | Lecturer], [Student | Students],
[Day | Days], [Time, | Times | Hour | Hours | Minute | Minutes], [Slot | Slots | Time
Slot],
[Place | Places | Room | Rooms], [Preference | Weight | Priority | Penalty],
[Consecutive | Continuous], [Simultaneously | Concurrently | Together | At the same
time],
[Teaches | Invigilates | Supervises], [Hold | Holds | Have capacity | Has capacity |
Capacity],
[Specify | Specifies | Require | Requires | Need | Needs], [Last | Lasts], [Duration
| Length],
[Contains | Comprises | Has | Have], [Is | Are],
[At least | No less than | No fewer than], [At most | No more than], [Exactly |
Precisely],
[Group_teachers | Area | Department], [Group_students | Class | Student_type],
[Room_type | Room_Group | Buildings], [Double_bookings | Clashes | Conflicts]
```

The use of these synonyms makes the description of input data files more easily readable by human users. Also, users can build their own list of synonyms in a separate file and, with the help of a simple pre-processor, that file can be directly converted into their proposed specification language. In the model proposed by this thesis, this functionality will be build into an ad-on module between the user interface and the data model. This thesis uses the following keywords to describe the concepts.

Course:	A unit of study offered during a particular semester and year (or period).
Session:	A contiguous period allocated to teaching a course. A course may have more than one session.
CourseCode:	A unique identifier for a course.
SessionCode:	a unique identifier for each session of a course. The SessionCode and CourseCode uniquely identify each Session in the timetable. If course CS111 has four lectures L1, L2, L3 and L4 then CS111L1, CS111L2, CS111L3 and CS111L4 uniquely identify the four sessions.
Day:	A day of the week, uniquely identified by a unique number representing each day of the week
Room:	A physical space in which a session is conducted. This may be a room, a playing field, swimming pool, a laboratory or any other physical space, uniquely identified by RoomCode.
TimeSlice:	A length of time, beginning at a time based on the 24 hour clock
TimeSlot:	a combination of a day and a timeslice
TimetableSlot:	a particular combination of timeslot and room which may be allocated to a session
TimeTableAllocation:	A session associated with a timetable slot
Timetable:	A set of all TimetableAllocations .
Constraint:	An expression that should be satisfied to solve a timetabling problem successfully.
Hard Constraint:	An expression that should (must) be satisfied necessarily to successfully solve a timetabling problem.
Soft Constraint:	An expression that should be preferably but not necessarily satisfied to successfully solve a timetabling problem.

3.3 A DATABASE APPROACH TO TIMETABLING

Johnson (1993) discusses the use of PCs and databases for storing educational timetables and resources. The data is stored as tables in the database. He states that if the system in question is to be used only for the basic 'book-keeping' tasks in timetable information presentation, one approach is to use a database management system. He advocates the most promising way forward is likely to be through the creation of an expert system.

He describes the use of a database system for the 'book keeping' aspects of timetable development. The system described has been implemented at the Loughborough University Business School. It has proved to be very effective means of assisting the production of high quality timetables as well as related teaching lists and forms. Together with Foulds (Foulds, Johnson 1998), he describes the database components of Slot Manager - a decision support system for the University of Waikato Management School timetabling.

Slot manager does not automate the timetabling process but, rather, helps the timetabler by providing powerful and wide ranging tools to designate the required slots, allocated courses to feasible slots and rooms, and create a wide variety of reports on the outcome of the timetabling process.

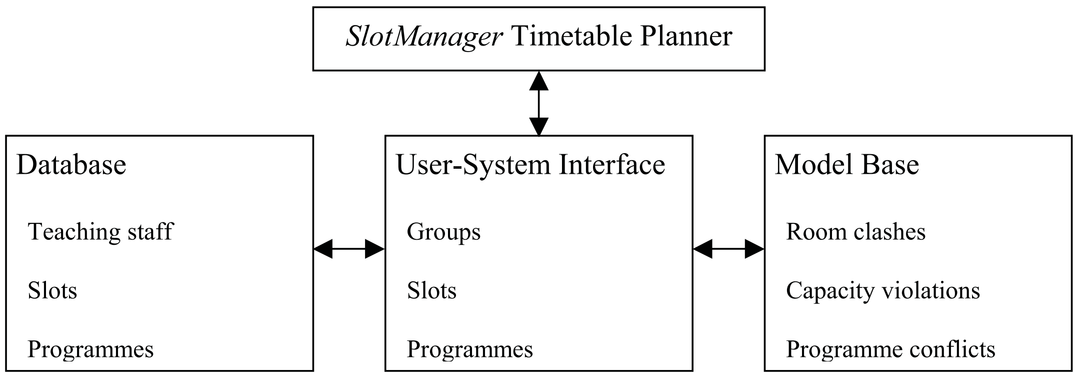


Fig. 3.1. SlotManager Component Structure

Foulds and Johnson (1998) describe an entity relationship database model of the resources. As they have stated, the model is only to 'book-keep' the resource data and therefore does not model the constraints. The ER model does not include the constraints which are modelled separately (*Figure 3.1*) in *SlotManager*, a user friendly, menu driven decision support system.

Foulds and Johnson (1998) state that their approach does not automatically generate timetables that attempt (and most likely fail) to satisfy all the various constraints and complicating factors that will inevitably exist in practical timetabling environment. Rather, the purpose of their DSS is to assist an experienced timetabler to allocate courses to both rooms and slots when producing a timetable from scratch or, more likely, modify an existing timetable. Consequently, their *SlotManager* does not have a mathematically complex model base.

Foulds and Johnson (1998) have thus suggested a decision support system model that provides ancillary aid to the timetabler. This thesis suggests a combination of the ancillary help with constraint based relational model that apart from providing

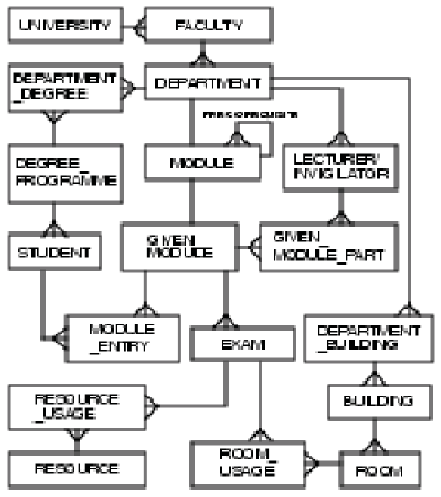


Figure 3.2. An entity relationship diagram of timetabling resources. (Weare 1995)

the ancillary help, also allows automatic generation of a timetable. Weare, R.

(1995) gives an entity relationship diagram (*Figure 3.2*) of the timetabling problem that includes resources but excludes the constraints.

3.4 Cumming and Paechter's Discussion on Standard Data Format

Cumming and Paechter (1995) proposed a standard data format in a discussion paper presented at the First International Conference on the Practice and Theory of Automated Timetabling (Edinburgh, UK) , but not submitted formally to the conference or printed in the proceedings. Their proposal was highly criticised at the conference for lack of generality but generated a big discussion about the subject in which the difficulty of creating such a standard became evident. They proposed principles and requirements to guide the creation of a timetabling data standard.

Their standard claims to represent complete and incomplete timetables and preferences. The components used are time slots (using a day.hh:mm representation format), events, staff and students, and rooms. They make no distinction between staff and students arguing that in some cases students can lecture classes. A list of keywords with different parameters is proposed as the standard. They also propose a cross convention (but not as part of the standard) that may be attached to any keyword and enables a Cartesian product between the arguments of that keyword (lists in this case). They also attempt to represent the soft constraints using cost functions.

Moreover, they conclude that timetable evaluation is likely to be the most difficult part to standardise. Some important omissions of this work are concerned with the availability of resources, split events, groups of resources, weeks and other type of periods, room types, definition of what is the problem to solve and the representation of the solution.

3.5 Cooper and Kingston's Standard Data Format for Timetabling

Cooper and Kingston (1995) presented a specification of the timetable construction problem based on a timetable specification language called TTL (TimeTabling Language). This language was formal yet flexible enough to specify instances encountered in practice. An earlier version of TTL appeared in (Cooper and Kingston 1993).

Cooper and Kingston (1995) modelled a TTL instance as consisting of a time group, some resource groups, and some meetings. They enumerated typical time groups as:

Mon1; Mon2; Mon3; Mon4; Mon5; Mon6; Mon7; Mon8; Tue1; Tue2; Tue3; Tue4; Tue5; Tue6; Tue7; Tue8; Wed1; Wed2; Wed3; Wed4; Wed5; Wed6; Wed7; Wed8; Thu1; Thu2; Thu3; Thu4; Thu5; Thu6; Thu7; Thu8; Fri1; Fri2; Fri3; Fri4; Fri5; Fri6; Fri7; Fri8; "[. . . . : . . : . ,][. . . . : . . : . ,][. . . . : . . : . ,][. . . : . . : . . ,][. . . . : . . : . ,]" end Times.

They listed the names of the times available for meetings, followed by a specification of the way in which the times are distributed over the days of the week: brackets enclose days, colons signify breaks, and dots and commas stand for times, with comma times being considered undesirable. Cooper and Kingston also modelled resources. Here is a typical resource group as they model it:

group Teachers is subgroups English, Science, Computing; Smith in English, Science; Jones in Science, Computing; Robinson in Computing; end Teachers;

This group contains resources (Smith, Jones, and Robinson) which are available to attend meetings, and subgroups that are subsets of the set of resources. These define functions that the resources are qualified to perform: teach English, etc. A resource may be in any number of subgroups. Typical instances have Teachers, Rooms, and Students resource groups.

After the groups come the meetings, which are collections of slots that are to be assigned elements of the various groups, subject to certain restrictions. For example, Cooper and Kingston (1995) typify a meeting expressing five Science classes that meet simultaneously for six times per week:

meeting 10-Science is Year10; 5 Science; 5 ScienceLab; 6 Times: TwoDouble; end 10-Science.

There is one slot that must contain the Year10 resource from the Students group; five slots which must contain resources from the Science subgroup; five resources from the ScienceLab subgroup of the Rooms group, and six times from the Times group, which must satisfy the condition TwoDouble.

This condition was defined to mean that the times must contain two pairs of adjacent times not spanning a break, and that the times are to be spread over as many days as possible with at most one undesirable time in the set. There is a small fixed list of such conditions built into their program, which can easily be extended as required. Formally, the meaning is that the eleven selected resources will all be occupied together for the six times; in fact, it is clear that the Year 10 students will be split into five groups.

The problem is to find an assignment of times and resources to all the slots that satisfy the various conditions and is such that no two meetings with a resource in common share a time. In Cooper and Kinston's (1995) experience, schools insist that all slots be filled with times and appropriate resources, so their program will fail rather than compromise on this. However, they are willing to compromise in two respects: time conditions need not be perfect, and resource slots may be occupied by one appropriate resource at one time, and a different appropriate resource at another (this is called a split assignment). They try to minimise the number of these defects. They have used the TTL language successfully, with some unimportant extensions, to specify high school instances.

Real instances may have two hundred or more meetings altogether, and capturing them in full detail in TTL can take half a day or more of careful work with the school's timetable coordinator.

In Cooper and Kingston's model, constraints are not modelled as part of the timetabling data, or resources. In STTL (Standard Timetabling Language), a further extension to TTL, Kingston (1999) states that one potential problem is the process of acquiring and standardizing problems. It will be necessary to restrict the installation of problem files, to prevent fragmentation of the space of timetabling problems. On the other hand, Kingston is opposed to any policy that excludes any researcher's work. Rather, he envisages a process of negotiation in which existing problem files are extended to accommodate specific real world instances expressed in STTL and offered by researchers.

This thesis proposes a model whereby the timetabling problems are easily standardised thereby overcoming the problem of acquiring and standardising problems. As a generic constraint based ER model is proposed, we do not expect any researcher's work to be excluded.

3.6 THE GATT TIMETABLING SYSTEM

An interesting work related to standard data format for timetabling problems is included in the GATT timetabling system (Collingwood *et al,* 1996) GATT ("Genetic Algorithm Time Tabler") uses a file format for describing timetabling problems. The format claims to be able to describe any GELTP ("General Exam/Lecture Timetabling Problem") and non-educational timetabling problems. The file format is verbose and uses as main components: events, time slots, rooms, students and teachers.

The format was essentially devised for exam timetabling problems. This way, some important omissions include weeks and other periods (useful for staff allocation and university course timetabling problems), room types, event sections (and section duration), continuity and load constraints (useful to achieve good quality schedules for teachers and students), student groups (essential in school timetabling) and teacher groups.

Thus, GATT proposes a genetic algorithm based model that lacks completeness and generality and therefore needs further refinement to be acceptable widely.

3.7 BURKE ET AL'S STANDARD TIMETABLING DATA FORMAT

A more recent paper by Burke *et al* (1998), proposes a different kind of standard for timetabling instances. They include a simple but incomplete description of the data types, keywords and syntax of the language and outline some further facilities to develop. Some concepts and constructs they use are similar to those found in the Z specification language (Potter B, *et al,* 1991). The format includes as data types: classes, functions, sets, sequences, integers, floats, Booleans, chars and strings. Classes include some attributes and functions and an inheritance mechanism is provided. A useful data type of this work is sets since many of the components of timetabling instances involve groups of resources (groups of students, groups of classes, etc.). In addition to the common set operators (member, union, intersection, subset, etc.), they also include operators like for all, exists, sum and prod. All these data types make the specification language close to a kind of programming language and enable the definition of constraints in logic programming language style.

A good extension of Burke, Kingston and Pepper's work could be the use of a constraint logic programming language as the specification language and the modelling of constraints together with the resources. Logic, associated with constraints performs very well in describing and solving timetabling problems. However, logic is not suitable as an interchange format between computer scientists and educational administrators.

This thesis proposes a relational model that exploits the declarative nature of Structured Query Language as the choice of the constraint language to encapsulate the resources, constraints and evaluating functions within single model. The problem of logic being not suitable as an interchange format between computer scientists and educational administrators is overcome by implementing the solution in the easily available relational database packages that allow computer scientists to work with constraint logic while offering a suitable user-friendly decision-supporting interface for educational administrators.

Separate models have been developed for university course and examination timetabling. Different or modified algorithms need to be applied separately to course and examination timetabling. A few cases of examination timetabling problems are now studied and it is shown how the exam timetable problems are identical to course timetabling.

Burke *et al* (1998) had expressed that a library of timetabling modules would be set up on the Internet.

3.8 DATA FORMAT FOR EXAMINATION AND COURSE TIMETABLING

Commenting on course and examination timetabling, Harikleia 1999 states that "Both problems are quite similar, though they present a few differences. There is usually only one exam but several lectures for a subject in one semester. Conflicts are not allowed in exam scheduling, but some may exist in course scheduling. In general, exam scheduling is less complex than course scheduling."

Harikleia does not explain explicitly why examination timetabling is less complex. The assumption is that since there are more lectures to be scheduled in course timetabling than in examination timetabling, there is a greater chance of clashes and hence harder to satisfy clash constraints. However, in course timetabling, while there are more lectures, the number of time slices available is generally larger. A university with 200 courses may have 4 one hour lectures per course thus requiring 800 hours per week.

Typically, a week consists of some 40 hours spread over a five-day week. If there were twenty rooms available, there would be 800 timetable slots available per week. In the case of exams, 200 exams, one for each course, would need to be scheduled. Given that, each exam lasts for three hours and there are two examination sessions per day (one in the morning, one in the afternoon, with a gap in between), 40 exams can be held in a single day in the twenty rooms, with 200 exams in a week. Thus, the increased number of lectures do not really make course timetabling a more complex problem than exam timetabling.

It may be argued, that in fact, the added hard constraint for exams that no two exams can be held at the same time while in course timetabling some clashes in allocations between certain courses (for example, a first year and a third year course) maybe allowed, make examination timetabling problem more complex. This is reflected by the fact that in most institutes, the examination period is longer than a week while course lectures are scheduled within a week.

3.9 SUMMARY

A standard timetabling data format must be general – be able to express any reasonable instance of the timetabling problem, it should be complete – be able to express any reasonable instance completely and should be practicable – it should be easy to translate data from current formats into the standard format. The Reis *et a*l (2000) verbose model does not generalise the constraints but suggests a list of synonyms for various constraints and data. It helps administrators specify constraints and does not simplify the constraints. They have not shown how all reasonable instances of timetabling problems can be specified, thus lack generality. Johnson's database approach and the Slot Manager are tools that assist administrators to 'book-keep' the data. They do not develop automatically a timetable and constraints are not represented in an ER model. Reis et al (2000), Cooper and Kingston (1995), Collingwood et al, (1996) and Burke et al (1998) have all described various constraints as being different whereas it will be shown in Chapter 4 that constraints can be generalised into three main types, thus simplifying the process of building a timetable and comparing solutions.

Chapter 4 GENERALIZATION OF TIMETABLING CONSTRAINTS

(Note: Part of this chapter is based on a paper by the author A. Chand "A Constraint Based Generic Model for Specifying Complete University Timetabling Data" accepted for publication in "The Proceedings of the 5th International Conference on the Theory and Practice of Automated Timetabling" held at Pittsburgh USA, Aug 8-12th 2004)

This chapter examines the various constraint requirements and shows how the various constraints are actually specific cases of three general categories of constraints – Domain, Spread and Count Constraints. A wide variety of constraints stated by various researchers and summarized by Burke (1994) and Goltz (1999) are listed and specified as generalized constraints.

A timetable allocation is a session associated with a timetable slot. Typically, an allocation consists of the session code, room code, day code and hour code. Initially, the timetable allocation is empty except for the session code. Values of room code, day code and hour code need to be mapped onto each timetable allocation. Thus, the domain of each timetable allocation comprises of the sets of room codes, day codes and hours codes.

Existing timetabling solutions typically use a database to store domain data but not constraints since the wide variety of constraints, expressed in varying formats and languages, make it difficult to model the general university timetabling problem conceptually. The constraint models are varied and not amenable to comparison. In this chapter, a generic constraint format is proposed and it is shown how it satisfies the requirements of generality, completeness and practicability.

Several timetabling cases will be now investigated for how they have formulated their constraints, how they appear to be different, what the similarities are and how they can be generalized.

4.1 Typical Constraints

Burke *et al* (1994) delineates some of the most common types of timetabling constraints:

- **Resource Assignment**
  A resource may be assigned to a resource of a different type, or to a meeting. For example, a lecturer prefers to teach in a particular room or an exam should take place in a particular building. Specific to USP, an exam such as LL321 must be held in the language lab.
- **Time Assignment**
  A meeting or a resource may be assigned to a time. This constraint can be used to specify days on which a teacher is unavailable, or to pre-assign a time to a particular meeting. Specific to USP's examination timetabling, certain examinations such CS122 should be held on a day between Tuesday and Friday only.
- **Time Constraints between Meetings**
  Common examples of this class of constraint are that one particular lecture must take place before another one, or a set of exams must be sat simultaneously. Specific to USP, certain examinations such as CS391 and CS491 should be held together because they have common components.
- **Meeting Spread**
  Meetings should be spread out in time. For example, no student should have more than one exam in any day and lectures of a course should be spread over different days.
- **Meeting Coherence**
  These constraints are designed to produce more organised and convenient timetables, and often run contrary to "meeting spread" constraints (above). For example, a lecturer prefers to have all his lectures in three days, giving him two lecture free days.
- **Room Capacities**
  The number of students in a room must not exceed its' capacity.

- **Continuity**

  A continuity constraint is any whose main purpose is to ensure that certain features of student timetables are constant or predictable. For example, lectures for the same course should be scheduled in the same room, or at the same time of day.

Goltz *et al* (1999) states the following constraints for the Medical Faculty at the Humboldt University timetabling problem:

C 1 : The teaching activities can begin every quarter of an hour and may run for different lengths of time.

C 2 : There are restrictions with respect to the starting time for each teaching activity.

C 3 : Two lectures about a same subject must not be scheduled for the same day.

C 4 : The lectures, seminars, and practicals of each group must not overlap.

C 5 : The number of seminars and practicals that a department (research group) can conduct at a certain time is limited.

C 6 : Some teaching activities are only held during certain weeks, where the following cases are possible: all weeks, A-weeks (weeks with odd numbers: 1,3,5, …), B-weeks (weeks with even numbers: 2,4,6, …), either A-weeks or B-weeks.

C 7 : There are lectures about the same subject carried out in parallel at the parts of the faculty. Thus, a student can choose from among parallel lectures.

C 8 : Timetabling also involves allocating rooms to the individual lectures.

C 9 : Some lectures can only be held in certain specified rooms; in some cases, only one room matches the requirements.

C 10 : The varying distances between the different locations of the teaching activities must be taken into account.

C 11 : The starting time preferences for teaching activities and the preferred rooms for lectures should also be taken into account if possible.

The constraint C 11 is a soft constraint. The other constraints are hard constraints and must be satisfied.”

Often, once an initial timetable has been built, it is circulated to faculty and other members for their comments. Requests are the received for changes, which are attempted to be fulfilled. This may entail making changes to other timetable allocations for which there have been no requests made for changes.

Burke *et al* (1994) and Goltz *et al* (1999) have specified different constraints for different timetabling environments. The algorithms developed are for the specific instances of the problems. A generic classification of constraints will now be defined, such that all of the above constraints, and others, can be modelled easily as a few classes of constraints.

There are many different objects that have to be timetabled – courses, lectures, exams, tutorials etc. A course may have one or more lectures and or exams or tutorials etc. Each lecture/exam/tutorial may last for a certain period. In each timetabling instance, there is a basic period. For example at USP, the basic period is the hour. A lecture may last one, two or more hours. The same applies to exams, tutorials etc. The basic period is referred herein as a ‘session’. A lecture may consist of two sessions that should be timetable contiguously. Chapter 5 models time and explains why ‘session’ is timetabled rather than a course or examination.

According to Reis *et al* (2000), some of these words are synonymous – a lecture is synonymous with exam. A lecture may be divided into smaller periods – some lectures may have a half an hour session while others may have two half an hour sessions held one after the other. A course may have three one hour lectures with a requirement that ‘a lecture (session) be timetabled at the same time and room as one of the lectures of another course because it is taught by the same lecturer and covers the same content’.

This thesis takes the approach that students attend sessions of a course rather than courses. Let us say two courses have three sessions each with one common session,. The approach is that the first course has two sessions, say S1 and S2, the second course has another two distinct sessions, S3 and S4, while session S5 is common to both, with the number of students enrolled for S5 being the sum of students enrolled for S1 and S3. Henceforward, we will discuss timetabling of sessions, rather than courses, lectures, tutorial, etc.

We propose that constraints are of three main types: Domain, Spread and Count Constraints. Domain constraints are those that limit a session to certain values of Time Slots and Room Codes. Spread constraints are those that specify how the Timetable Allocations should be spread over time and rooms. Count constraints are those that limit the number of resources that participate in a grouping of resources. Before discussing these constraints in more detail, we will discuss a measure of constraint hardness that applies to all three categories of constraints.

4.2 MEASURE OF CONSTRAINT HARDNESS

The extent to which a constraint must be satisfied is referred to as it hardness which can be specified by giving the constraint a weight. The weights can be limited to a range of values between positive and negative limits. While the limits may be arbitrarily selected for different timetabling instances, this model proposes a standard scale of 100% and –100%. Intermediate weights indicate how soft each constraint is. For example, a weight of 100 means that the constraint must be satisfied while a weight of -100 means that satisfying the constraint must be avoided. Values in between 100 and -100 indicate that the constraint is softer, with a weight of zero indicating that it does not matter if the constraint is satisfied or not. In fact, constraints with a weight of zero are best left out of the problem because they do not make any difference to the acceptability of the solution.

Thus, the absolute value of the weight hence indicates how hard a constraint is. The higher the absolute value, the harder is the constraint. Given 5 constraints and their respective weights as (c1,100), (c2,50), (c3,0) (c4,-50), (c5,-100), c1 and c5 are equally important hard constraints as their absolute weight is same while c3, which is least important, together with c2 and c4 are soft constraints. Since c3 has a weight of zero, dropping constraint c3 reduces the search space thus increasing the efficiency of the search mechanism.

4.3 DOMAIN CONSTRAINTS

Domain constraints are those that restrict a session to certain domain values. A session may be restricted to certain rooms, days, hours or combinations thereof. Constraints such as "The number of students in a session cannot exceed the capacity of the room allocated to it", "Only computer lab classes can be held in the computer labs" and "Avoid scheduling examinations of video broadcast courses classes on Mondays" lie in the domain category. Domain constraints can be specified as

DomainConstraint((SessionList), (DomainList), Weight)

where *Domain* is replaced by the name of the actual domain such as day, hour, room etc. *SessionList* is the list of sessions that that must be allocated to a domain element that must be in the *DomainList*. *Weight* is a numeric value corresponding to the hardness of the constraint. A negative weight indicates that that no session in *SessionList* must be allocated a value in *DomainList.* Specific domain constraints are discussed next.

Room Constraints

There is a variety of room constraints. At USP, these include constraints that a room must be able to accommodate the sessions it is allocated and that certain rooms are reserved for special sessions. To a smaller extent at USP and much larger extent at larger universities, sessions should also be allocated to rooms that are closer to the students' main faculty. Certain sessions may need to be scheduled together in the same room.

Sometimes, one room may be allocated to more than one session at a time. At other times, one session may need to be allocated more than one room. Some other typical room constraints are:

a. A session may need to be allocated to a room with special facilities.
b. A session may need to be restricted to certain rooms for special reason, e.g. rooms close to the department.
c. A room may be restricted to special sessions only, e.g. only Physics laboratory classes can be held in the Physics Laboratory.

All these constraints can be represented as

RoomConstraint(SessionList, RoomList, Weight)

where *SessionList* is a set of sessions that are constrained to rooms in the *RoomList*, and *Weight* is a measure of the hardness of the constraint.
Examples of the above room constraints are given below:

The constraint that Physics sessions (PH101T1, PH101T2,) need to be allocated to a lab room R1with special facilities is specified as

RoomConstraint((PH101T1, PH101T2), (R1), 100)

Session CS121L1, CS121L2, and CS121L3 needs to be held in a room with enough capacity. Say rooms R10, R11, and R12 have enough capacity. The constraint is then specified as:

RoomConstraint((CS121L1, CS121L2, CS121L3), (R10, R11, R12), 100)

In some situations, two or more sessions may be held in the same room. In that case, *RoomConstraint* is dynamically changed by allocating attributes to Room: Maximum Capacity and Current Capacity. If a room has a maximum capacity of 200 and is allocated a session of size 50 for a particular time, then current capacity is updated to 150 for that particular time. Relational database systems typically have the capability to update current capacity automatically via a trigger whenever a session is allocated to a particular room. Non-relational database timetabling systems could use a similar event driven mechanism to update room constraints. If only one session can be held at a time in particular room, then current capacity is updated to 0 for that particular time.

Time Constraints

Time domain constraints in university timetabling involve division of the year into smaller semesters or trimesters that are further divided into weeks. Typically, the five weekdays are used for teaching activities. Each day is further divided into hours or parts of an hour. The various ways in which time is grouped results in various time constraints that appear to be difficult to standardise.

The standard time constraint proposed in this thesis is

PeriodConstraint((SessionList), (PeriodList), Weight)

where *Period* is the period being constrained, *SessionList* is the list of sessions that are being constrained, *PeriodList* is the list of periods that the sessions in the *SessionList* are constrained to and *Weight* is the hardness of the constraint.

Periods are typically divided into days and hours of the day. A session may be restricted to certain days, certain hours and/or combinations of certain days and hours. We will now examine how the generic time domain constraint can be used to specify the day, hour and combination of day and hour constraints.

Day constraints

These constraints are those that restrict the variables to certain values in the day domain. For example, all sessions must be held on a day from Monday to Friday, but not during weekends.

The constraint
DayConstraint(SessionList, DayList, Weight)
specifies that all the sessions in the *SessionList* must be timetabled on a day that is in the *DayList*.

Days can be easily represented with numbers that indicate their positional relationship with each other. The following table shows the day numbering scheme used hereinafter.

Table 4 .1 Day numbering

Day	**Number**
Monday	1
Tuesday	2
Wednesday	3
Thursday	4
Friday	5
Saturday	6
Sunday	7

Thus the constraint
DayConstraint((CS100, CS102, CS122). (1,2,3,4,5), 100)
means that the sessions CS100, CS102 and CS122 must be timetabled on a weekday.

Hour Constraints
These constraints are those that restrict the variables to certain values in the hour domain. For example, all sessions must be timetabled on the hour, from 8 am to 7 p.m., but not before 8 a.m. and not after 7 p.m.

The constraint

HourConstraint(SessionList, HourList, Weight)

specifies that the sessions in the session list must be timetabled at an hour in the hour list.

The above example can be specified as:

HourConstraint((CS100, CS102, CS122), (8,9,10,11,12,13,14,15,16,17,18), 100)

Certain sessions may need to be held at the same hour each day. If it is a specific hour (e.g. 10) then *HourConstraint(CS121), (10), 100)* suffices but if the constraint is that the sessions can be at any hour but it should be same for all (referred to by Burke (1999) as continuity constraints), then this constraint is a spread constraint that is discussed later on.

Day Hour constraints

These constraints are those that restrict the variable to certain combinations of day and hour values. A combination of a day and hour is referred to as a time slot. For example, certain sessions may need to be allocated time slots only between Monday 10 a.m. and Monday 2 p.m.

The constraint

TimeConstraint(SessionList, TimeList, Weight)

specifies that the sessions in the session list must be allocated a time in the time list. *TimeList* is a Cartesian product of *DayList* and *HourList*. Day and Hour can be integrated into a single number using various functions such as

$$T(day, hour) = day * 100 + hour$$

where day is any whole number between 1 and 7 and hour is any integer between 0 and 23.

For example, Monday 10 a.m. is allocated 110, Monday 11 a.m. is allocated 112 and so forth. Thus the constraint that the sessions CS111 and CS101 should be timetabled at any hour from 10 am to 2 pm on Monday, can be specified as

TimeConstraint ((CS111, CS101), (110, 111, 112, 113, 114), 100)

Since Time Slot is a combination of day and hour, time constraints can be specified as combinations of day and hour constraints, if each hour is identified by a unique key and each day is identified by a unique key derived from the hour's key as discussed. For example, the constraint

TimeConstraint ((CS111, CS101), (110, 111, 112, 113, 114), 100)

has the same meaning as

DayConstraint ((CS111, CS101), (1), 100 and
HourConstraint ((CS111, CS101), (10, 11, 12, 13, 14), 100)

The Day constraint specifies that the two sessions should be allocated the day 1, while the Hour constraint specifies that the two sessions should be allocated an hour between 10 and 14 inclusive. Both, Day and Hour constraints need to be satisfied simultaneously. However, the advantage of using *TimeConstraint* is that it is more intuitive. Since only one constraint is used, searching for and evaluating, a solution is expected to be more efficient.

4.4 SPREAD CONSTRAINTS

Spread constraints are those that dictate how the sessions should be spread over time relative to each other. Spread constraints also apply to other resources such as rooms as shown below. The one spread constraint can be used to specify the variety of constraints that have been categorized differently by various researchers. The various categories of spread constraints are discussed next and then it is shown how these constraints can be specified with the proposed generic spread constraint

DomainSpread((SessionList), (SpreadList),Order,Weight)

where the *domain* is one of the timetabling resources or Cartesian product of the resources with each element of the domain being denoted by a numerical value. *SessionList* is the list of sessions that need to be spread over the domain. *SpreadList* is a list of numbers that the sessions in *SessionList* should be spread by, that is the difference between the domain element allocated to a session in *SessionList* and the domain element allocated to any session appearing later in the list, should be equal to a number in the *SpreadList*. A session may need to be timetabled before or after another. For example, a theory session must be held before the practical session. The value of *Order* indicates if the order of the allocated sessions is important. A value of 1 means that the order should be maintained while a value of 0 means that the order is not important, that is if *Order* is 1 and *SessionList* is (S1, S2, S3 and S4) then S4 must be scheduled after S3 which must be schedule after S2 which must be scheduled after S1.

For example the constraint *DomainSpread((S1, S2, S3, S4, S5), (1,2,3),1,100)* specifies that if session *Sn* is allocated a domain element *Dn,* and session *Sm* is allocated a domain element *Dm* then for *n<m, Dm-Dn* equals *1, 2 or 3* and *Sm* should be timetabled after *Dn.*

No-Clash Constraints

A no-clash constraint specifies that a particular session should not clash with another given session. Some situations that create no-clash constraints are; when a session should not be scheduled at the same time as it is a co-requisite or the two sessions are taught by the same lecturer, or two sessions have a number of common students enrolled in both. Typically, sessions of the same course should not clash. For example, the sessions CS121L1, AF101L1, and MA131L1 should not be timetabled at the same time as they have common students. These constraints can be specified as

TimeSpread((CS121L1, AF101L1, MA131L1),(0),0,-100)

which means that the given sessions must not be (as indicated by the weight of –100) allocated Time Slots such that the difference between any pair of Time Slots is zero.

Concurrent Constraints

A concurrent constraint, the opposite of a no clash constraint, is one that specifies that sessions be scheduled at the same time. For example, a 3rd year session may need to be conducted at the same time as a 4th year session where the 3rd year session is part of the 4th year session. Perhaps, the examination sessions of CS391 and CS491 need to be held at the same time. These constraints can be specified as

TimeSpread((CS391, CS491),(0),0, 100).

Note that if the two examinations were not to be timetabled at the same time, the constraint would be specified as

TimeSpread((CS391, CS491),(0), -100)

which is the same as the previous constraint except that the weight is minus 100. Thus, the proposed domain spread constraint mathematically captures the meaning of the no clash and concurrent constraints and their difference.

Contiguity Constraints:
Sometimes certain sessions may need to be timetabled contiguously, that is one after the other. Such constraints have also been termed as ordering constraints. For example, a tutorial session may need to be held immediately after a lecture session. Some lectures may be longer than one session. In such cases, the two or more sessions should be timetabled contiguously.

If the sessions PH101P1, PH101P2, PH101P3 form one continuous laboratory activity then the constraint can be specified as

TimeSpread((PH101P1, PH101P2), (1),1, 100)
and *TimeSpread((PH101P2, PH101P3), (1),1, 100)*

that specify that the three PH101 sessions must be held contiguously, one after the other – PH101P2 should be allocated a time slot immediately after PH101P1 and PH101P3 should be held immediately after PH101P2. In the above case, the while the three sessions have to be contiguous, it does not matter which of the three is timetabled first, second or third. Therefore, the order the three sessions are scheduled can be chosen randomly. In cases where the order is important, such as a session on computing theory, say CS101T that is immediately followed by computer lab session CS101P, the above constraint format captures the requirement as

TimeSpread((CS101T, CS101P), (1),1, 100)

in that, the session that should be timetabled earlier is specified earlier in the session list and *Order* has a value of 1.

An important requirement in university timetabling is that certain sessions should be spread over the timetabling period. For example, the lectures of a course should be timetabled on different days of the week. In another case, the sessions may need to be distributed over the hours of a particular day.

The requirement that sessions are spread over the week is handled by the constraint

DaySpread((SessionList), (DaySpreadList),Order, Weight)

that specifies that the sessions in *SessionList* must be timetabled such that the difference between the timetabled days of any pair of sessions in *SessionList*, is equal to a number in *DaySpreadList* that is a list of numbers. For example, the constraint

DaySpread((S1, S2, S3, S4),(1,2,3,4),0,100)

specifies that the sessions S1, S2, S3 and S4 must be timetabled such that they are separated by 1, 2, 3 or 4 days, that is, they cannot be timetabled on the same day if *DayConstraint* has specified that only days 1 to 5 can be used for timetabling. Alternatively, the constraint

DaySpread((S1, S2, S3, S4),(0), 0,-100)

specifies the same constraint by stating that the difference in the timetabled day should not be zero.

Just as sessions may need to be spread over certain days, they may also need to be spread over certain hours. The constraint

HourSpread((SessionList), (SpreadList),Order, Weight)

specifies that the sessions in *SessionList* must be timetabled such that the difference between the timetabled days of any pair of sessions in *SessionList* is equal to a number in *SpreadList* that is a list of numbers. For example, perhaps a part-time lecturer desires that all of the sessions CS111L1, CS111L2, CS111L3 need to be timetabled at the same hour of the day lectures either at 12 noon or at 17 hour (5 pm) The constraint

HourSpread((CS111L1, CS111L2, CS111L3), (0), 0, 100)

specifies that the three sessions must be timetabled such that the difference between their allocated hours is zero, that is, they are timetabled at the same hour. Together with the above constraint, the following constraint

HourConstraint((CS111L1, CS111L2, CS111L3), (12, 17), 100)

specifies that the sessions CS111L1, CS111L2, CS111L3 be all scheduled at 12 noon a.m. or all at 5 p.m.

Since Time Slot is a combination of day and hour, time spread constraints can be specified as combinations of day and hour constraints. For example, the constraint

TimeSpread((PH101P2, PH101P3), (0), 0, 100)

has the same meaning as

DaySpread((PH101P2, PH101P3), (0), 0, 100) and
HourSpread((PH101P2, PH101P3), (0), 0, 100).

The *DaySpread* constraint specifies that the two sessions should be allocated the same day, while the *HourSpread* constraint specifies that the two sessions should be allocated the same hour. Together, the two constraints ensure that the sessions are allocated the same day and hour.

Room Spread Constraints.

The *RoomSpread* constraint restricts the number of rooms that can be allocated to a set of sessions and is generally a soft one. Sometimes a lecturer may prefer to use only one room. It is much easier to remember that all your lectures are in one room or perhaps two rooms. The constraint

RoomSpread(SessionList, SpreadList, 0,Weight)

specifies that number of different rooms that are allocated to the sessions in session list must be limited to a number in *SpreadList.* For example

RoomSpread((CS121L1, CS121L2, CS121L3 CS121L4),(1,2),0, 100)

specifies that the sessions CS121L1, CS121L2, CS121L3and CS121L4 must be allocated to not more than two different rooms.

A problem with evaluating room spread is that it is not straight forward to apply the formula for calculating the spread. For example the constraint

RoomSpread((S1, S2, S3, S4, S5), (1,2,3), 0, 100)

specifies that if session *sn* is allocated a domain element *dx,* and session *dy* is allocated a domain element *dm* then for *m<n dy-dx* equals *1, 2 or 3.* However, it is not viable to number rooms 1, 2 3 and so forth because if a session is allocated room 4 and another session of the same constraint spread list is allocated room 10, it does not mean that the room spread is 6. Time is fixed in eternity, it does not change and each period remains in a fixed position relative to any other period and hence periods can be assigned fixed values that indicate each period’s position from another. If a

session is allocated a period *m* and another session is allocated time period *n* then it holds that the two sessions are spread by |*m*-*n*| periods.

For non-time domain such as rooms, if a session is allocated a room numbered *m* and another session is allocated a room numbered *n* what does it mean if we say that the two rooms are separated by |*m*-*n*| rooms? If two sessions are allocated the same room then |*m*-*n*| is zero and the meaning that the sessions have been allocated the same room holds. However, if *m*<>*n* then it is incorrect to say the sessions have been allocated rooms that are separated by |*m*-*n*| rooms. It seems that while the spread constraint is meaningful for time spread constraints, it is not consistent in its meaning for room constraints. However, this superficiality is revealed when we recall that room is actual a space. Space is a three dimensional entity parts of which are enclosed by walls, ceilings, floors or other barriers to form sub-spaces such as room, sports field, and laboratory, etc. Thus, room spread constraint refers to the spread of the location of the rooms allocated to sessions. Therefore, when it is said that two sessions are separated by |*m*-*n*| rooms, the same meaning holds as in the case of time spread constraints.

However, while it is practical to divide time into basic units that form larger periods of various sizes, it is highly impractical to divided space into basic units that form rooms of various sizes. Time is linear while space is three-dimensional. Measuring time spread is quite straightforward while determining space spread involves a more complex three-dimensional geometry. Specifying room spread spatially is a topic for future research.

Meanwhile, since it is highly likely that academic administration would be disinterested in room spread constraint because the constraint that rooms that are spatially closer and more desirable for a session can be specified as a domain constraint, it is proposed to use only time spread constraints for the present. A count function can be used to determine the number of rooms (or other resources) used. A generic constraint

CountResource((SessionList), (ResourceList), CountExpression, Weight) could be used in the model, where *SessionList* is a list of sessions, *ResourceList* is a list of resources and *CountExpression* is an expression that states the requirement such as 2 (exactly two resources used) or <3 (1 or 2 resources used), or Minimise (allocate the same resource as allocated before).

Room-TimeSpread, Room-DaySpread, Room-HourSpread Constraints

Just as the *TimeConstraint* is an expression over the combined day and hour that can be allocated to a session, expressions over combinations of room, time, day and hour may also be stated. Of these, only *RoomTimeSpread* is of common interest while others are not used in the timetabling problem. None the less, the specification described below for *RoomTimeSpread* applies equally to all other combinations.

Usually only one session may be held in a room at a time. If sessions S1, S2, S3 and S4 are the four sessions that have to be timetabled and we do not wish to allocate them to the same timetable slot (same room at the same time) this requirement is fulfilled by the RoomTimeSpread constraint

RoomTimeSpread((SessionList), (RoomTimeSpreadList),0, Weight)

which specifies that the number of different timetable slots that are allocated to the sessions in session list must be (or not be) limited to a number in *RoomTimeSpreadList*. Thus, the above constraint can be specified as

RoomTimeSpreadConstraint((S1, S2, S3, S4), (0), 0, -100)

which means that sessions S1, S2, S3, S4 must not have a room time spread of zero – that is, must not be timetabled in the same room at the same time.

The *RoomTime* constraint could be expressed as two separate constraints – *RoomSpread* and *TimeSpread.* The constraint

RoomTimeSpreadConstraint((S1, S2, S3, S4), (0),0, -100) can be specified as

RoomSpreadConstraint((S1, S2, S3, S4), (0),0, -100) and
TimeSpreadConstraint((S1, S2, S3, S4), (0), 0, -100).

Recall that *TimeSpreadConstraint* can be also specified as two separate constraints – *DaySpreadConstraint* and *HourSpreadConstraint.*

Thus, *RoomTimeSpreadConstraint* may be specified as three separate constraints – *DaySpreadConstraint, HourSpreadConstraint* and *RoomSpreadConstraint.* Future research will examine and analyse the impacts that the choice of using combined or separate constraints have on building and evaluating timetables. It is not possible to split up constraints into more basic ones always. For example, if a session is restrained to either Monday 9 am of Tuesday 3 pm, the constraints

DayConstraint((S1), (1,2), 100) and
HourConstraint((S1), (9,15), 100)

do not specify the requirements since the above constraints accept Monday 3 pm and Tuesday 9 am as feasible. Only the constraint

DayHourConstraint((S1), (109, 215), 100)

correctly specifies that only Monday 9 am and Tuesday 3 pm are acceptable.

Next it is shown how the generic model can be applied to Burke's (1994) and Holtz' (1999) constraints.

4.5 APPLICATION OF MODEL TO BURKE'S AND GOLTZ'S CONSTRAINTS

This section describes how the constraints specified by Burke (1994) and Goltz (1999) are represented in the generic model.

4.5.1 Representation of Burke's Constraints

Burke (1994) specified the following category of constraints that will now be represented in the generic model

- Resource Assignment
- Time Assignment
- Time Constraints between Meetings
- Meeting Spread
- Meeting Coherence
- Room Capacities
- Continuity

- **Resource Assignment**

The typical resource in university timetabling is the room. If a lecturer prefers a particular room, say RX, for the his course which has four sessions S1, S2, S3 and S4 then this constraint is specified as

RoomConstraint((S1, S2, S3, S4), (RX), 100)

where *100* indicates that is a hard constraint.

- **Time Assignment**

This is a domain constraint. A session may be restricted to certain days, hours or Time Slots. For example, sessions S1 and S2 are restricted to hours 5 pm, 6 pm and 7 pm because the sessions are meant for employed people who are working during the day. Using the twenty four hour notation, this constraint is specified as

HourConstraint((S1, S2), (17, 18, 19), 100)

where *100* is the upper limit of the weight scale thus making the constraint a hard one.

- **Time Constraints between Meetings**

Common examples of this class of constraint are that one particular lecture must take place before another one, or a set of exams must be sat simultaneously. This is a spread constraint. If sessions S1 must take place before S2 then the constraint is specified as

TimeSpread((S1, S2), (1), 1, 100)

which means that the difference between the time allocations for S1 and S2 must be 1, S2 must follow S1 because the value of *Order* is 1, and *100* is the upper limit of the weight scale.

In the case of two exams S1 and S2 that must be sat simultaneously, the constraint is

TimeSpread((S1, S2), (0), 0, 100)

which means that the difference between the time allocations of S1 and S2 must be 0, that is, the sessions are timetabled at the same time. The value of *Order* does not matter.

- **Meeting Spread**

Meetings should be spread out in time. For example, no student should have more than one exam in any day and lectures of a course should be spread over different days. This again is a spread constraint. If a student is enrolled for sessions S1, S2, and S3 of a course or sitting three different exams, then the constraint

DaySpread((S1, S2, S3), (0), 0, –100)

specifies that the difference in the day should not be 0, that is, the three sessions should not be scheduled on the same day.

- **Meeting Coherence**

These constraints are designed to produce more organised and convenient timetables, and often run contrary to "meeting spread" constraints. For example, a lecturer prefers to have all his lectures in three days, giving him two lecture free days. If a lecturer teaches sessions S1, S2, S3, S4 and S5 the count constraint

CountResource((S1, S2, S3, S4, S5), (1, 2, 3, 4, 5), 3 , 100)

states that the required sessions must be allocated one of the five days and the total number of days allocated must be 3.

- **Room Capacities**

The case of assigning rooms to sessions such that the room is able to accommodate the number of students in that session is specified as

RoomConstraint((S1,S2,S3,S4), (R1, R4, R5), 100)

where *R1, R4* and *R5* are the rooms with sufficient capacity.

- **Continuity**

A continuity constraint is any whose main purpose is to ensure that certain features of student timetables are constant or predictable. For example, lectures for the same course should be scheduled in the same room, or at the same time of day. These are only domain, spread or count constraints. The constraint that lectures should be held in the same room is specified as

CountResource((S1, S2), (R1, R1), 1, 100)

The constraint

HourSpread((S1, S2.), (0),0, 100)

specifies that the difference in the hours allocated to the sessions should be 0, that is, they are timetabled at the same hour. Thus, the proposed model is able to describe completely the constraints stated by Burke (1994).

4.5.2 REPRESENTATION OF GOLTZ' CONSTRAINTS

Goltz *et al* (1999) states very general constraints for the Medical Faculty at the Humboldt University timetabling problem. These constraints are examined next and it is shown how they are specified in the proposed model.

C 1: The teaching activities can begin every quarter of an hour and may run for different lengths of time.

This is, in fact multiple constraints, the first being that activities can begin every quarter of an hour and the second that teaching activities may vary in length. The basic period being used in this case is the quarter hour. To accommodate the second constraint that teaching activities may vary in length, a teaching activity may be divided into smaller sessions each one quarter of an hour in length. If a teaching activity requires *n* (e.g. 3) quarters of an hour, then the activity can be divided into *n* (3) sessions with the constraint that the *n* (3) sessions should be held contiguously in the same room. In the proposed model, the above constraint C1 resolves into three constraints:

- A day spread constraint that sessions should be held on the same day
- A quarter spread constraint that the sessions should be spread by a quarter hour each
- A room constraint that the sessions should held in the same room

Recall that the generic domain spread constrain

DomainSpread((SessionList), (SpreadList), Order, Weight)

allows one to replace *Domain* with specific domain types or cross product of domain types. The above three spread constraints can be combined into one to give

QuarterDayRoom((SessionList), (SpreadList), Order, Weight)

Thus, if a teaching activity is divided into three quarter of an hour sessions, the constraints

QuarterDayRoomSpread((S1, S2),(1),1, 100)
QuarterDayRoomSpread((S2, S3),(1),1, 100)

specify that the sessions S2 must be allocated a time period that is 1 time periods away from the previous session S1 (*Order* is 1 hence the order of the sessions must be maintained), thus ensuring that S2 is timetabled immediately after S1 on the same day, and room and S3 is similarly timetabled immediately after S2.

C 2: There are restrictions with respect to the starting time for each teaching activity.

This is a domain constraint. For example, if a teaching activity's starting time is restricted to the following quarters of an hour, 1, 3, 5, the restriction can specified by the constraint

QuarterConstraint((S1, S2), (1,3,5), 100)

C 3: Two lectures about a same subject must not be scheduled for the same day.

This is a day spread constraint. If sessions S1 and S2 are lectures of the same subject then the constraint

DaySpread((S1, S2),(0), 0, -100))

specifies that the sessions must not be on the same day.

C 4: The lectures, seminars, and practicals of each group must not overlap.

This is a time spread constraint. The required sessions should not be timetabled at the same time, as specified by the constraint

DayQuarterSpread((S1, S2, ...), (0), -100)

C 5: The number of seminars and practicals that a department (research group) can conduct at a certain time is limited.

This statement is much generalized and meaning of this constraint is not very clear. Obviously, all resources are limited in timetabling problems. The reason why sessions are limited is not stated. Is it because of limited resources, or staff? In that case, domain constraints would specify the limitation. Otherwise, if the limit were a certain maximum number of resources, then a *CountResource* constraint would represent the requirement.

C 6: Some teaching activities are only held during certain weeks, where the following cases are possible: all weeks, A-weeks (weeks with odd numbers: 1,3,5,…), B-weeks (weeks with even numbers: 2,4,6,…), either A-weeks or B-weeks.

This is a domain constraint. If sessions S1 and S2 can be held any week then the constraint

WeekConstraint((S1,S2), (1,2,3,4,5,6), 1)

specifies that the sessions can be held any week, given that the timetabling weeks are 1 through 6.

If sessions S3 and S4 can be held only on odd numbered weeks then the constraint

WeekConstraint((S3,S4), (1,3,5), 1)

specifies that the sessions can be held on an odd numbered week. Similarly if sessions S5 and S6 can be held on even numbered weeks then

The constraint

WeekConstraint((S1,S2), (2,4,6), 1)

specifies that the sessions can be held on even numbered weeks.

C 7: There are lectures about the same subject carried out in parallel at different parts of the faculty. Thus, a student can choose from among parallel lectures.

This is a spread constraint that requires sessions to be held at the same time. If sessions S1 and S2 are held in parallel, then the constraint

DayQuarterSpread((S1, S2), (0), 0, 100)

specifies that the sessions must have a spread of 0, that is, timetabled at the same time.

C 8: Timetabling also involves allocating rooms to the individual lectures.

This involves room domain and room spread constraints. If session S1 can be held only in room R5 then the constraint

RoomConstraint((S1), (R1), 100)

specifies that only room R1 can be allocated to session S1

C 9: Some lectures can only be held in certain specified rooms; in some cases, only one room matches the requirements.

This is a room domain constraint same as *C8.*

C 10 : The varying distances between the different locations of the teaching activities must be taken into account.

If the rooms R1, R6 and R7 are geographically suitable for sessions S1 and S2 then the room domain constraint

RoomConstraint((S1, S2), (R1, R6,R7), 100)

specifies that only those rooms should be used. If the geographically suitable rooms are not available at all, then the constraint is relaxed by selecting a weight less than 100, the limit of the weight scale.

C 11: The starting time preferences for teaching activities and the preferred rooms for lectures should also be taken into account if possible.

This is a soft time and room domain constraints. If Rooms R1 and R2 are preferred for sessions S1 and S2 then the soft

RoomConstraint((S1, S2), (R1,R2), X)

specifies the requirement where *X* is less than 100 .

4.6 SUMMARY

Timetabling constraints vary from one problem to another. These various timetabling constraints can be specified using only three generic constraints: Domain, Spread and Count.

Domain Constraints

The generic domain constraint is

DomainConstraint((SessionList), (DomainList), Weight).

Domain constraints restrict the feasible domain for a session. Specific to USP timetabling , the domain constraints are –

- *Day((SessionList)t, (DayList), Weight)*
- *Hour((SessionList), (HourList), Weight)*
- *DayHour((SessionList), (TimeList), Weight)*
- *Room((SessionList), (RoomList), Weight)*

<u>Spread Constraints</u>

The generic spread constraint is

DomainSpread((SessionList), (DomainSpreadList), Order, Weight).

Spread constraints govern the how the feasible domain of a session is spread. The spread constraint of interest is those that involve time.

Specific to the USP timetabling environment, the spread constraints are

- *DaySpread((SessionList), (DaySpreadList), Order, Weight)*
- *HourSpread((SessionList), (HourSpreadList), Order, Weight)*
- *DayHourSpread((SessionList), (TimeSpreadList), Order, Weight)*
- *RoomDayHourSpread((SessionList), (RoomDifferenceList), Order, Weight)*

The constraint

CountResource((SessionList), (ResourceList), CountExpression, Weight)

can be used to specify all limitations on the number of resources or timetabling objects scheduled, where *SessionList* is a list of sessions. *ResourceList* is a list of resources and *CountExpression* is an expression that states the requirement such as 2 (exactly two resources used) or <3 (1 or 2 resources used), or Minimise (allocate the same resource as allocated before).

It may be argued that some meaning is lost by generalizing the constraints; for example, *DayHourSpread((S1, S2), (0), 0, -N)* (the sessions S1 and S2 should not be scheduled at the same time) may mean that the two sessions have students in common or are taught by the same lecturer. However, it should be noted that standardizing involves eliciting the general essence of a problem devoid of local causes or idiosyncrasies. The proposed model can be extended to include constraints that are more verbose. However, at the core of the timetabling solving method, they would resolve into the proposed constraints.

It has been shown that the various timetabling constraints as specified by Burke (1994) and Goltz (1999) can be represented using the proposed generic model and is applicable to both course and exam timetabling. The proposed format can be easily applied in languages such as Lisp, Prolog and constraint logic languages, as well as relational databases. In the next chapter, the timetabling resources including time and sessions will be represented using entity relational modelling and the generalized constraints represented within the same ER model.

Chapter 5 A Generic Entity Relationship Model for Timetabling

This chapter shows how the various different timetabling resources and constraints can be represented as a generic ER model. It discusses the ways in which time and other resources are treated in the various problems and how constraints form the relationships between time and the resources. After discussing a generic entity relationship model for timetabling, the USP timetabling resources and constraints will be used to illustrate that the model applies to both course and examination timetabling.

5.1 Modelling Time

In university environments, time has been categorized differently. Some universities have trimesters while others have bi-semesters per year. The number of weeks per trimester and bi-semester differ as well as the number of hours per week available for teaching. The start time and end time of each timeslot also differ – some timeslots start on the hour while others may start on the half-hour or even quarter-hour. The duration of activity also differs. Lectures may last about an hour or so while laboratory activity may last two or more hours

In addition to the various time requirements, time itself being an abstract infinite entity with no beginning or end and indivisible – analogue in nature, makes modelling time difficult. The question "what are the attributes of time?" is difficult to answer hence time is arbitrarily divided into smaller 'discrete' practical periods. In university timetabling, most activities start on the hour and finish 5 or 10 minutes before the hour and no activity lasts less than a minute. Hence selecting the minute as the smallest discrete component of time suffices for modelling university timetabling. Traditionally, minutes are grouped into hours that are grouped into days that in turn are grouped into weeks and so on. Normally, in university timetabling the group for which the timetable repeats is the week - that is, each week follows the same timetable. In certain timetabling situations such as examination timetabling the timetable is a one off occurrence and is not repeated.

Time can be more generally modelled as sub periods and super-periods. A single time unit may be grouped into another collective unit which may be named ‘Hour’ or ‘Period A’ etc. The new group may then be grouped into a bigger group that may be called ‘Day’ or ‘Period B’ etc . The duration of each time unit is determined by a start time and end time. Thus, time is seen as a sequence of timeslices, each of which is identified by a unique number that indicates its position with other timeslices.

Crow’s foot notation has been used to represent the model with the use of Visio. Visio uses standard attribute notation to identify the relationships between entities. The notation includes the following:

(<u>Underlined</u>) Primary key, which uniquely identifies each occurrence of the entity.

(FK) Foreign Key, which identifies the primary key attribute of another entity contributed by a relationship.

(AK) Alternate Key, which is an alternate unique identifier for an entity.

(IE) Inversion Entry, which is a non-unique access identifier for an entity.

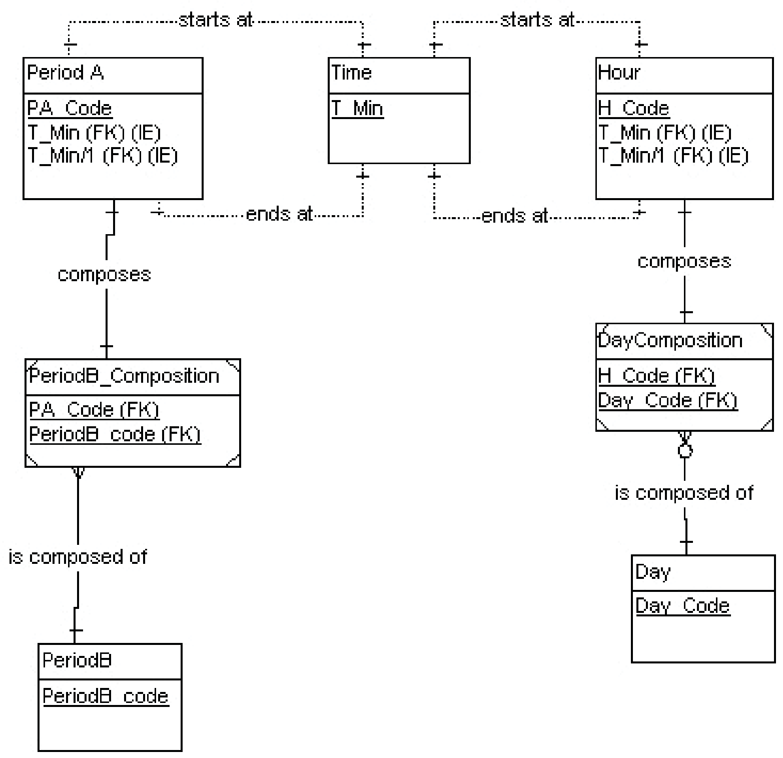


Figure 5.1 Time Model

In *Figure 5.1* the generic entities *Time, Period A* and *Period B.*depict how time is modelled. The entities *Time, Hour* and *Day* depict a real instance example of time. *T_min* is the start time of smallest timeslice unit (*min* for minimum, but also conveniently corresponds to minute). In entity Period A, *T_min* is the start time of each period and *T_min/1* is the end time of each period. If a timetable for a whole year is to be build then the 366 days of the whole year is divided into 527,040 minutes. The unique value for *T_min* can be calculated from the month, day number, hour number and minute number by using a hashing function. This would enable another function to determine the month, day, hour and minute number from *T_min*. Alternatively, the day, hour, and minute corresponding to each value of T_min ordered in sequence from 0 to 527,039 could be stored in a table and obtained by using a lookup function. A value of 0 for T_min corresponds to the first minute of the first hour of the first day of the first month. A value of 1 would refer to the second minute and so forth. The absence of the leap day in non-leap years is not a problem because in those years, the minutes corresponding to the leap day are restricted from being used by domain constraints. Similarly, domain constraints eliminate weekends and nights as need be. PA_Code and PB_Code are numeric codes that specify the different groupings of timeslices. For example, the first 60 minutes (T_Min = 0 to 59) constitute hour number 1, the second 60 minutes (T_Min = 60 to 119) constitute hour 2 and so on till the last 60 (T_Min = 526,980 to 527,039) minutes constitute hour 8784, the first 24 hours constitute day 1, the second 24 hours constitute day 2 and the last 24 hours constitute day 366.

Entities Hour, Day and Week are examples of the generic entities Period A and Period B. Week could be a further grouping corresponding to a generic entity Period C. Each period has been given its own unique code, though the start and end time could be used as an identifier.

The levels of grouping to be actually modelled are determined by the instance of the problem. If a timetable has to be repeated each week then only two groups need be modelled – one representing hour and another day, if hours and days are the units that would be used for timetabling. It is possible that a day may be divided into morning and afternoon and an event that is held in the morning may have a related event in the afternoon.

Thus, in a daily week situation there would be 5 periods (days) numbered 1 to 5 while in a half-day week situation there would be 10 periods (half-days) numbered 1 to 10. Therefore, the two situations are not different and easily represented by the same time model.

5.2 MODELLING COURSES

A course may have one or more sessions allocated to it for teaching purposes. The timetabler needs to know the enrolment details such as the student codes and the courses a student has enrolled for and the number of students enrolled per session for clash control and seating capacity.

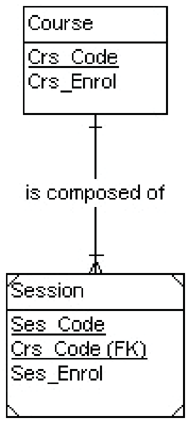


Figure 5.2 Modelling Courses.

Sometimes sessions of courses are combined together as discussed in Chapter 3 where it has been proposed that in such cases the real life situation be modelled rather than the administrative labelling. Thus, two courses A and B may have certain sessions A1, B1, in common but while officially the sessions are separate, in reality they are the same and should be modelled as such for the timetabling problem.

In *Figure 5.2* Crs_Code is the code for each course and Crs_Enrol is the number of students enrolled for that course. Each course has to have at least one session identified by the session code. Ses_Enrol is the number of students attending the particular session. A large class with, say, 400 students may be divided into two smaller sections of 200 students each. The number of students enrolled for a particular session will be used to measure constraint satisfaction.

5.3 MODELLING ROOMS

Space is analogous to time in that space has no beginning, no end, and is an analogue quantity. Just as a practical discrete time-period is selected as the basic unit, space too is measured in discrete unit such as cubic millimetre. Labelling of time into periods such as minutes, hours days and so forth does not change the nature of time and similarly placement of physically barriers such as walls and ceilings do not change the nature of space. It is commonly said that a cloud has covered the sun whereas the truth is that the cloud has covered our eye and not the sun that is too large for the cloud to cover. Similarly, physical barriers bar our access to space but do not limit space itself. It is common practice in many institutes in Fiji that rooms are separated by removable partitions to form larger gathering space. This thesis views space from this approach. For easier visualization, space is called herein, room.

Room has been defined earlier in this chapter as 'A physical space in which a session is conducted. This may be a room , a playing field, swimming pool, a laboratory or any other physical space, uniquely identified by RoomCode.' Two attributes of room – maximum capacity and current capacity have already been identified as desired information. The current capacity is used to determine if the room has enough space to accommodate another session at the same time. This is useful in cases such as when many examinations may be held in large halls.

The counter part of this requirement is that some sessions may be split into two rooms. There may not be enough space in one room to hold an examination and hence two or more rooms have to be used. For example, Rooms SO25 and SO26 maybe two adjacent rooms of capacities 50 each whose adjacent wall can be opened to form a larger room of capacity 100. The larger room is called a 'virtual room' with capacity 100. Any system will need to keep track that the virtual room is made up of the smaller rooms and update the timetabling data for the smaller room with any changes made to the timetabling data for the virtual room.

In another example, geographically remote halls H1 and H2 can be used for the same video broadcast lecture - one lecture is simultaneously send to both halls. Thus, virtual room Hall1/2 is created and timetabling booking needs to be done for each of the actual halls.

In cases where only the actual room is used without combining with any other room, the virtual room is the room itself. Allocation of a timetable slot to a session triggers an update on the timetabling data (current capacity) of the associated rooms.

Virtual rooms are needed so that two physical rooms can be joined to accommodate larger classes and so that the integrity of the timetabling data can be maintained.

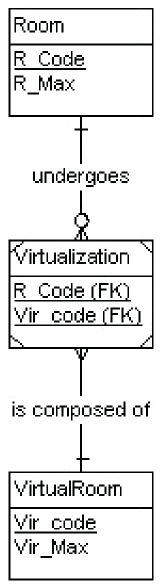


Figure 5.3 Modelling Rooms

This need can be addressed by the creation of 'virtual' rooms (in *Figure 5.3*).

R_Code is the room code of a defined space. In entity *VirtualRoom*, the attribute *Vir_Code* is the code of the virtual room. A room may be used in creating more than one virtual room and a room by itself is a virtual room. *Max* is the total number of all students in each session that can be allocated to the room. A function *current* associated with each timetable slot, calculates the total number of students in each session that has been allocated to a room at the same time. As each session is assigned to a timetable slot, the current capacity of each virtual room is updated. If room R1 has a maximum capacity of 300 and current capacity of 250 and room R2 has a maximum capacity of 100 and a current capacity of 50, the virtual room R1-R2 has a maximum capacity of 400 and current capacity of 300. The timetable solver determines from the time-room spread constraints which sessions should be not be held at the same time in the same room.

5.4 MODELLING CONSTRAINTS

The time domain constraint specifications for the various periods such as hour and day are similar. The constraints are of the general form

DomainConstraint(SessionList, DomainList, Weight).

Time constraints restrict sessions to certain period or combination of periods. Typically, the periods are days and hour of the day. The generic time domain constraint model after normalization is shown in *Figure 5.4*. An example of an actual model is given in *Figure 5.5.*

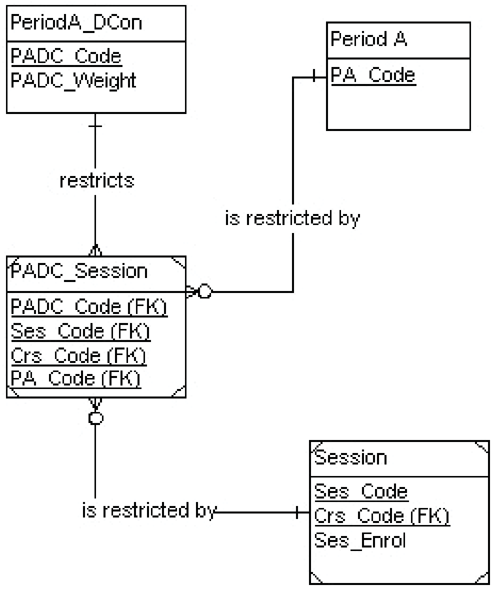


Figure 5.4. A Generic Time Domain Constraint Model

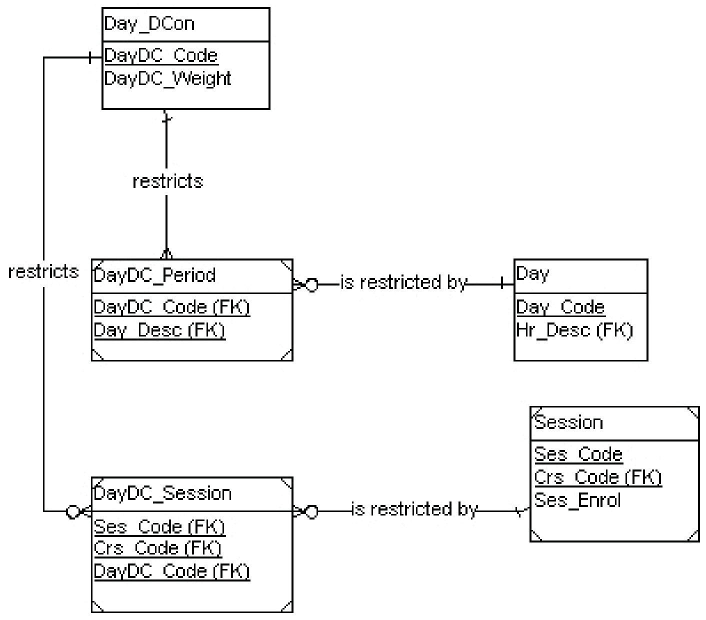


Figure 5.5 A Day Domain Constraint Model for the USP timetabling system.

Our discussion of the time model has shown that the basic time unit maybe clustered into larger groups. Typically, hours are clustered into days. However, there are cases when hours may be clustered into morning and afternoon hours. For example, a large class such as CS121 may be divided into two smaller groups with the first group having lectures CS121L1, CS121L2, CS121L3, CS121L4 during the morning and the second group having lectures CS121L5, CS121L6, CS121L7, CS121L8 during the afternoon Thus, it is permissible to have a session CS121L1 in the morning and another session CS121L5 in the afternoon. In such cases, morning and afternoons comprise a different grouping in between the day and hour groupings and a morning spread constraint is introduced to specify that the relevant lectures for the first group should have a morning spread of more than zero (should not be timetabled in the same morning). Similarly, an afternoon spread constraint is introduced for the second group of lectures.

It should be noted that, in the above example, a day spread constraint for each set of lectures becomes redundant since spreading lectures over different mornings guarantees that lectures are spread over different days. Also, keeping in line with the convention that reality should be modelled, the lectures CS121L1, CS121L2, CS121L3, CS121L4 are treated as sessions of one course, say, CS121A and the other lectures are treated as sessions of course CS121B. Since the sessions codes are unique, labelling of the sessions CS121L1 to CS121L8 is sufficient to identify the lectures as those of two separate courses though changing the labels to CS121AL1 to CS121AL4 and CS121BL1 to CS121BL4 would make the labels more informative where A and B denote that they are two sections of the same course.

Apart from being constrained to one individual domain, a session may be constrained to values from combinations of domains. For example, sessions CS491L1 and CS492L2 which serves part-time students who work during the day, may have to be scheduled after 5 pm on any day from Monday to Thursday (Friday afternoon is for shopping). The ER model for this case is same as that of an individual domain where the domain is the Cartesian product of the two domains. The generic case is depicted in *Figure 5.6* while an actual instance is shown in *Figure 5.7*

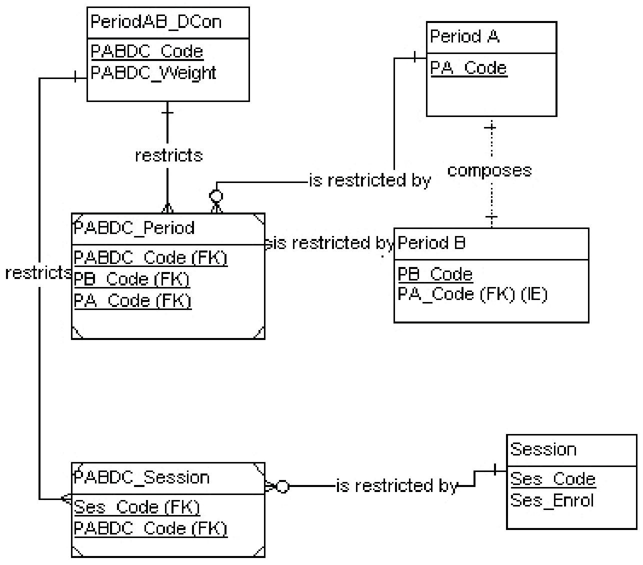


Figure 5.6. A Generic ER Model for Timetabling Constraints over Two Domains.

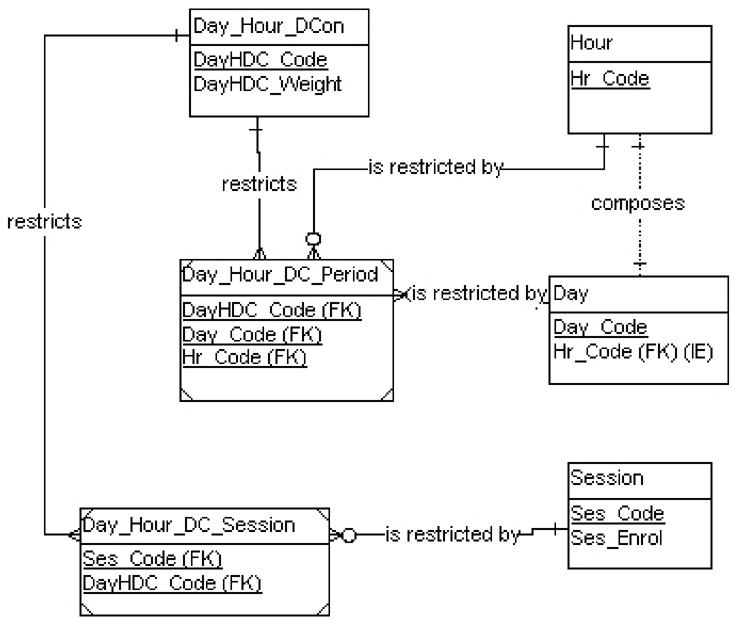


Figure 5.7 An ER Model for Time Constraints over two domains for USP timetabling.

Recall that in modelling time, we have suggested that each time unit be given a code that stores the sequential position of the unit in time - for example, Monday 8 am is allocated 0108 while Tuesday 9 am is allocated 0209. Thus, the DayHour constraint for the particular sessions is

DayHourConstraint((CS491, CS492), (01017, 0118, 0119, 02017, 0218, 0219, 03017, 0318, 0319, 04017, 0418, 0419),100) corresponding to the general format

DomainConstraint(Session List, DomainList, Weight)

which when normalized becomes

DomainConstraint(Constraint_Id, Weight)
DomainConstraintSession(Constraint_Id, SessionCode)
DomainConstraintDomain(Constraint_Id, DomainCode).

For the instance in the example, the normalized constraint data is

DayHourConstraint

Constraint_Id	**Weight**
1	100

DayHourSession

Constraint_Id	**Session**
1	CS491
1	CS492

DayHourDomain

Constraint_id	**Domain_code**
1	0117
1	0118
1	0119
1	0217
1	0218
1	0219
1	0317
1	0318
1	0319
1	0417
1	0418
1	0419

Room domain constraints are specified in a similar format. A large class such as CS211 with 200 students is restricted only to the following venues that have sufficient seating – 093-001, 093-002, 084-008. The normalized constraint data for this case is

RoomConstraint

Constraint_Id	Weight
1	100

RoomConstraintSession

Constraint_id	Domain_code
1	CS211L1
1	CS211L2
1	CS211L3

RoomConstraintRoom

Constraint_id	Room_code
1	092001
1	092003
1	084008

MODELLING RESOURCES AS DOMAIN CONSTRAINTS

All the timetabling data formats reviewed in Chapter 3 model only resources and not constraints or model resources separately from constraints. We now examine typical statements about timetabling resources and then formulate them as domain constraint statements.

Statement 1: The teaching period is the week and a session of a course can be timetabled on the hour on any weekday from 8 am to 9 pm.

Statement 2: The examination period is two weeks and an examination may be scheduled on any morning or afternoon, beginning at 9 am or 2 pm of any weekday.

The above two statements are similar in that teaching period is equivalent to examination period and session of course is equivalent to examination. Therefore, statement two will be used to demonstrate that domains can be specified as constraints.

A straightforward domain constraint such as the allocation of a day between Tuesday to Friday to the video broadcast course CS122 is easily understood as domain constraint and stored as the following tables.

DayConstraint

Constraint_Id	**Weight**
1	1

DayConstraintSession

Constraint_Id	**Session_Code**
1	CS122

(Note that as examination is being scheduled, and there is only one examination per course, there is only one session to be scheduled per course).

DayConstraintDay

Constraint_Id	**Day_Code**
1	2
1	3
1	4
1	5
1	9
1	10
1	11
1	12

(Note that Monday is day 1, days 6 and 7 are the weekend days, and day 8 is another Monday).

The Day domain constraint with Id 1 states that the day allocated to CS122 must be one of 2, 3, 4, 5, 9, 10, 11, and 12. DayC1 must be satisfied to solve the examination-timetabling problem successfully as indicated by its weight of 1. We can now (or even have done so earlier) introduce another day domain constraint DayC2 that states that the day allocated to CS122 must be one of 1,2,3,4,5,8,9,10,11,12. This constraint includes Mondays and encompasses the entire domain of the timetabling problem. Storing this constraint entails adding a new row the DayConstraint table with a constraint id of 2 and weight of 1 and corresponding rows in the other two tables.

By definition of the solution of the timetabling problem, all constraints must be satisfied to solve the problem successfully. Hence, both constraints DayC1 and DayC2 must be satisfied and the only way to satisfy both constraints is to satisfy DayC1 that guarantees that DayC2 is satisfied.

MODELLING SPREAD CONSTRAINTS

As in the case of domain constraints, spread constraints are also similar in format to each other, the general form being

DomainSpread (SessionList, SpreadList, Order, Weight)

After applying normalization techniques, the spread constraints are resolved into the following tables.

DomainSpread (Constraint_Id, Order, Weight)
DomainSpreadSession(Constraint_Id, Session_Code)
DomainSpreadSpread(Constraint_Id, Spread)

The generic ER model is shown in figure 5.8

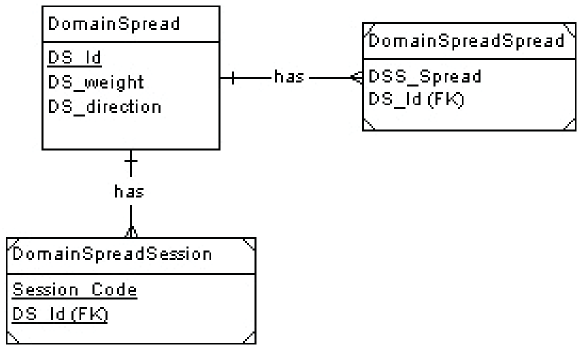
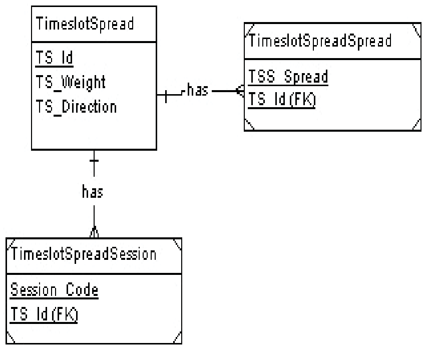


Figure 5.8 A generic ER Model for Domain Spread Constraints

Entity DomainSpread is the spread constraint. DomainSpreadSession specifies the sessions associated with each constraint. DomainSpreadSpread is the set of spread values associated with each constraint. The term "domain" may be replaced with the name of the actual domain for each timetabling problem. Typically, the domains are rooms, day and hour. An example of an instance of the generic day spread constraint is shown in Figure 5.9.

Figure 5.10. ER Model for timeslot spread constraints in university timetabling.



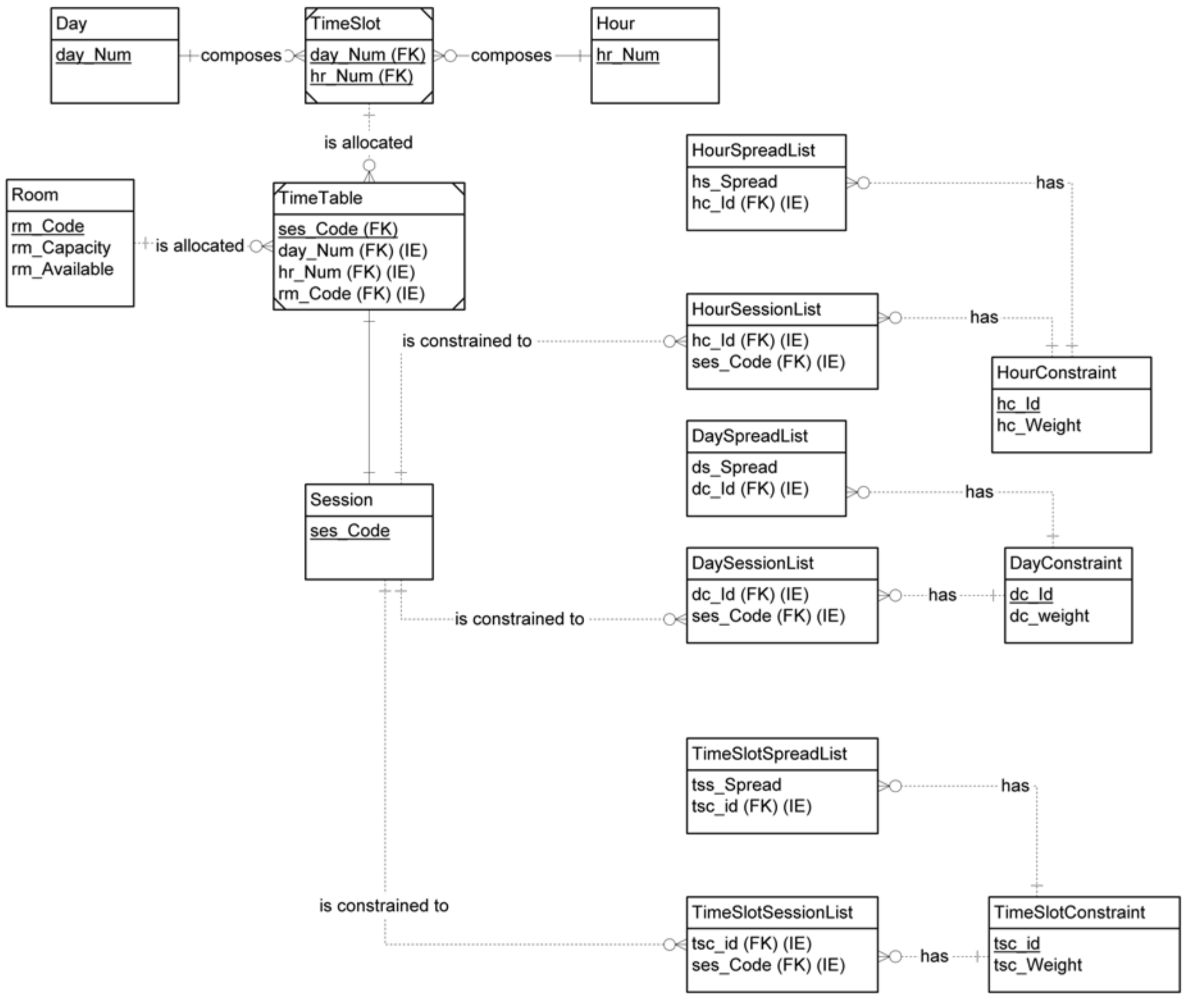
The most important spread constraint is that no two sessions must be scheduled at the same day and hour (timeslot) if they have students in common. This constraint involves two domains – day and hour. It is represented by the same model as the single domain spread constraint. Figure 5.10 that shows the model, for the day hour spread constraint, is the same as Figure 5.9 with Day being replaced by TimeSlot.

So far we have examined ERMs of the different resources, domain and spread constraints. In the next section, we will examine the overall relationships between the resources and constraints.

5.5 CONSTRAINT DIRECTED TIMETABLING SYSTEM MODEL

In this section, we discuss the relationships between the sessions, the domain, the constraints and the timetable. The complete entity relational model is shown in Figure 5.11. We observe that the structure of the two constraints, domain and spread

Figure 5.11 The Integrated Domain and Constraint ERM for University Timetabling



The user interface can incorporate a verbose standard as proposed by Reis et al (2000) to translate constraints expressed in layman's language into the relational database equivalent thus allowing educational administrators to specify the various constraints easily. The constraint revolver is independent of the verbose format of constraints allowing different algorithms to act on the same generic database of resources and constraints. This provides a common platform to compare the timetables provided by different solutions.

While relational databases are widespread, the proposed data format in this thesis is analogous to sets and lists and hence very amenable to languages such as Lisp and Prolog that process sets or lists. Data in a relational database is very easily exported to a variety of other formats. Thus, practicability is achieved.

5.6 SUMMARY

In this chapter, we developed an entity relational model for the various timetabling objects such as sessions and resources, and for all the constraints. The generic model incorporates a standard model for time that allows different groupings and regroupings of time to be represented. It has been shown that spread constraints can be adequately represented in a generic ER model. The USP timetabling data including constraints are adequately represented by the model. The model will be embedded in a constraint directed reasoning system that uses the constraints to produce feasible timetabling domains for the sessions.

In the next chapter, a constraint based heuristic algorithm that can be applied to reasoning system to generate a timetable automatically will be described

Chapter 6 A Constraint based Heuristic Algorithm for University Timetabling.

(Note: This chapter is based on a paper published by the author: Atish Chand (2002) A Heuristic Approach to Constraint Optimization in Timetabling, South Pacific Journal of Natural Science, Vol. 20, pp 64 –67 and a conference presentation: A Chand, An Heuristic Approach to Constraint Optimization in Examination Time Tabling, 2001 New Zealand Mathematics Colloquium, Massey University, Palmerston-North, NEW ZEALAND, 3-6 December , 2001.)

This chapter describes a constraint-based heuristic that is designed to eliminate the need to backtrack when no suitable timetable allocation can be found for a session by ordering constraints and resources. Another heuristic of dividing sessions into cliques is examined and a search algorithm described. A function that can be used to weigh constraint satisfaction is proposed.

6.1 Introduction

In the manual examination timetabling system, one of the major problems is dealing with clashes and finding clash free slots. Making a change requires that one has to undo previous exam allocation and look for a new allocation. This creates a series of backtracks which are difficult to resolve. In the manual system, it has been observed that exams that have fewer constraints imposed upon them are easier to shift around in the timetable than more heavily constrained ones. Resources that are less constrained are also more easily available. The heuristic logic is that if most constrained exams were allocated the least constrained resources first, then it would be easier to fulfil the leftover requirements as the leftover exams are less constrained and the resources used can be more easily allocated to other exams as the used resources are less constrained. Based on this heuristic logic, a heuristic algorithm was developed to avoid the problem of having to undo previously generated partial timetable allocations when the exam being currently assigned does not have any feasible solution.

6.2 SEARCH HEURISTICS.

Domain specific heuristics have been implemented for timetabling solutions as early as the 1960s (Foxley, 1968), (Cole 1964). While heuristic based methods typically produce quite acceptable results in a very small CPU time, they are not as optimised as the more intensive search methods. In the next few sections, a simple efficient technique based on heuristics that includes constraint ordering to build the solution is described. A careful evaluation of constraints before building the timetable means that this simple technique will select solutions that violate the constraints least. Hence, these solutions are more optimal.

It has already been shown that university course and examination timetabling problems are in essence the same. The fundamental constraints that must be satisfied by any feasible exam timetable are:

- any two exams allocated the same timeslot should not have any students enrolled for both exams.
- seating capacity for any timetable slot should not be exceeded.

Other constraints may be imposed on exams, for example:

- If two conflicting exams are scheduled on the same day, then there should be a full timeslot between them.
- the number of conflicting exams being scheduled on the same day should be minimised
- One exam may need to be scheduled before another, related, exam. For example, the theoretical drama exam may need to be scheduled before the practical drama exam.
- Two exams may need to be scheduled at the same time, as they contain similar material.

With most exam timetabling problems, the spread constraints present the greatest difficulty as the other constraints usually arise much less often.

6.3 Heuristic Methods

A common heuristic approach is the sequential method, where a sequencing strategy is applied to the problem. Approaches of this kind have been used for timetabling problems since the sixties (Foxley1968, Cole1964) and can produce quite acceptable results with minimal equipment requirements. The method involves using a heuristic to estimate how difficult an exam will be to schedule. Then, we can order the events by decreasing difficulty. Scheduling in this order means that a considerably better timetable should be found than if the events were scheduled in a random order. When scheduling the events there may be a choice of periods in which to schedule, in which case another strategy for period selection is required. Examples of strategies could be 'placing in the first available period', 'placing in the period with lowest cost' or maybe 'placing in the period with the most similar exams.

Session Ordering

Most of the sessions are affected by more than one constraint. Allocating a value to one session reduces the domain of other sessions affected by the same constraint. For example, a spread constraint *TimeSpread((S1,S2),(1),1,80)* may require that session S1 is held before preferably before session S2, a domain constraint *Hour((S1),(9, 10),1,60)* may specify that session S1 is held preferably during morning hours, and spread constraint *TimeSpread((S1,S6),(0),0,-100)* may specify that S1 must not be timetabled at the same time as S6. Thus, session S1 is constrained by three constraints. The first two constraints are soft while the third is a hard one. The third constraint must to be satisfied for a timetable to be feasible while the first two constraints will be used to determine how acceptable a timetable is. According to the domain constraint evaluation function

$$D = \sum_{c=1}^{c=n} abs(w_c / (L - abs(w_c)) * E$$

(where *D* is the sum of Domain constraint violations, *n* is the number of domain constraints *wc* is the weight of each constrain *c*, *abs(wc)* is the absolute value of *wc*, *E* is the number of students affected and *L* is the upper limit of the constraint weight) and spread evaluation function

$$S = \sum_{c=1}^{c=n} abs(w_c/(L - abs(w_c)) * E$$

(where S is the sum of Spread constraint violations, n is the number of spread constraints *wc* is the weight of each constrain *c*, *abs(wc)* is the absolute value of *wc*, E is the number of students affected and L is the upper limit of the constraint weight)

the maximum possible soft constraint violations for session S1 is the sum of violation due to the first two constraints. (The violation due to the hard constraint is infinite). Table 6.1 shows the number of common students enrolled in pairs of sessions. The values in the diagonal are the number of students enrolled in each session.

Table 6.1 Number of clashes per exam pair

	S1	**S2**	**S3**	**S4**	**S5**	**S6**
S1	35	12	14	10	16	8
S2		40	16	11	7	14
S3			37	16	10	11
S4				52	15	17
S5					39	12
S6						55

Let these sessions be restrained by the constraint
TimeSpread((S1, S2, S3, S4, S5), (0), 0, 90)

Table 6.2 shows that time spread constraint measure for each session calculated according to the spread evaluation function.

Table 6.2 Measure of Time Spread Constraint

Session Code	Constraint Measure
S1	5940
S2	5940
S3	6633
S4	7722
S5	5940

The most constrained session is S4 followed by S3, S5, and then by S1, S2.and S5 on equal ranking. The timetable problem solver uses a session ordering to decide which session to timetable next by selecting the most constrained session.

Domain Ordering

Each resource such as a room, timeslot or timetable slot may have one or more constraints placed on it. Continuing the above example, the large rooms R1 and R2, (capacities 60 and 40 respectively) may be available for the sessions S1, S3, S4, S5 and S6. However, since R2 is the smaller of the two it cannot accommodate the sessions S4 and S6.which have an enrolment greater than 40, the capacity of R2 Thus R1 is constrained by 6 sessions while R2 is constrained by 4 sessions. It is expected that resources that are less constrained would be more easily timetabled. The constraint

RoomConstraint((S4, S6), (R1), 100)

specifies that the sessions *S4* and *S6* must be allocated room R1 while the constraint

RoomConstraint((S1, S2, S3, S5), (R1, R2), 100)

specifies that the sessions S1, S2, S3 and S5 must be allocated either room R1 or R2. In evaluating a timetable for constraint satisfaction, the weight of 100 for hard constraints yields a value that cannot be evaluated by the evaluation functions since it involves division by zero. When a room is selected, the smaller feasible room that is less constrained will be favoured ahead of a larger room.

Divide and Conquer

Certain sets of sessions may be mutually time exclusive. They may have no time constraints in common. Hence, they can be timetabled at the same time. These mutually exclusive sets of constraints can be timetabled independently. Once mutually exclusive sets of sessions are identified, the problem solver searches the domain to filter all values that do not satisfy the associated constraints, using the ordering heuristics discussed above.

A constraint is not satisfiable when all values in a domain for a given session are eliminated as being feasible. In such a case, the options available in order to reach an acceptable stage are either to modify the unsatisfied constraint or to modify the domain. The domain itself may be modified in two ways; firstly, new values can be added to the domain and secondly, previous timetable allocations could be unallocated so that certain values are returned to the domain.

In order to reduce the need to undo previous allocations, that is, to avoid cases where all values in a domain are eliminated, domain ordering heuristic is used. Since it is predicted that the ordering heuristics would reduce or avoid the need to backtrack, the following sequential search is used, that does not do any backtracking.

6.4 SEARCH ALGORITHM

The heuristic non-backtracking method used here is similar to the approach used by Burke *et al* (1995) without the backtracking and with an additional grouping of resources according to how constraint each resource is relative to each other, including the sessions that are to be timetabled. Their method is described in more detail in Chapter 7.

The basic constrained based heuristic algorithm is:

- Order all the sessions from most constrained to least constrained
- Order each category of resources (typically time and rooms) from most constrained to least constrained
- For each unallocated session, allocate the least constrained resource to the most constrained session such that the associated constraints are satisfied.

A lower level algorithm is:

(Remarks on lines starting with /* are in verbose English while remarks on lines without /* are in pseudo code.

```
/* State the timetabling objects, resources and constraints
Input (exams, resources, constraints.)
/*create blank timetable
Timetable = new(exams, resources, constraints)
/* order the exams from most constrained to least
OrderedExamList =OrderExams(Exams,  constraints)
/* order the  resources from least constrained to most
OrderedResources = OrderResources(Resources, Exams, Constraints)
/* order the  domain constraints from most constrained to least
OrderedDomain = OrderDomain(Resources, Exams Constraints)

/* Populate Timetable
 For all exams in OrderedExamList do
        /* Determine the spread constraints that affect the current exam
        Spread_Constraint_List = CreateSpread_Constraint_Listt(CurrentExam, Constraints)

        /* Determine the resources available for exam using the domain constraints
        /* to  filter the resources available
        Resources_Available = CreateResAvail(CurrentExam, OrderedResources, Constraints)
        /* Use the spread constraints to determine a grouping of resources that does
        /* not violate any constraint

        /* Determine all the resources required for current exam
        While a session has resources left to be allocated
        /*while a resource has not been found for the current exam and there is a
        /*resource  left

        resource_found = false
        While  not(resource_found) and resource_left do
                /* get the next available resource
                CurrentResource = pop(Resources_Available)

                /* If current resource does not violate any constraint then
                /* allocate it to the exam
                If NoClash(CurrentResources, Clashlist, Timetable) then
                ResourceFound = true
                AssignToTT(CurrentResource CurrentExam, TimeTable)
                end if.
        /* if a resource was not find then try next resource
        Loop While
/* determine next category of resource for the current exam
loop while
/* determine the resources for next exam
Next Exam.
/* timetabling process is now complete
Output Timetable
```

A timetable is built by selecting values from the filtered domain. The filtered domain is the initial domain that has elements removed by the domain constraints. As values are selected from the domain for one variable, the domain for other variables may be affected in that those domains may be further filtered. Therefore, as one timetable allocation is made, new ordered list of constrained resources for each exam is determined.

In the process of eliminating elements from a domain, there may be a situation where the filtered domain is empty. In such cases, it would be necessary to undo one or more of the previous allocations in order to populate an empty domain. It is suspected that backtracking is necessary only for soft constraints because experimental results with test data showed that all hard constraints are satisfied. If a hard constraint cannot be satisfied, then the above algorithm does not allocate any timetable slot to an exam. However, results with the USP examination timetabling problem showed that there is no need to backtrack. Since the above algorithm assigns the most constrained exams to the least constrained resources first, it is expected that any backtracking will involve assigning a more constrained exam to a more constrained resource thus increasing the probability for that resource to be unavailable to other exams and resulting in a less acceptable timetable.

The two resources that require allocation to an exam are timeslot and room. A heuristic technique is to first group all exams into those that can be timetabled together, that is no two exams in the same group have any conflicts with each other. The exams in each group may then be easily assigned rooms. The above algorithm implicitly achieves this by nesting the room allocation search loop within the timeslot allocation search loop. The algorithm was implemented in MSAccess for USP's examination timetabling problem. The exams, resources and constraints were constructed as tables while queries were used to filter and order the resources and exams.

In Section 6.3, the divide and conquer approach to timetabling was discussed. During the implementation of the above algorithm, it was observed that if it is assumed that there are only spread constraints and no domain constraints, then sets of sessions are produced that can be timetabled independently of each other, that is, sessions in each set do not have any common students. Similarly, if it is assumed that are only domain constraints and no spread constraints, then sets of sessions are produced that do not have conflicting resources. While this behaviour can be used as a heuristic to divide sessions into independent cliques, we propose to investigate this in further research where a calculus of timetabling constraints will be explored.

6.5 SUMMARY

This chapter described a constraint based heuristic that attempts to eliminate the need to backtrack when no suitable timetable allocation can be found for a session by ordering constraints and resources. It was shown that the domain and spread constraints could be ordered from most constraining (important) to least constraining. Resources and sessions were also ordered according to most constrained to least constrained. The basic heuristic of assigning the least constrained resource to the most constrained session to satisfy the most important constraint was used to develop a search algorithm. Another heuristic of dividing sessions into independent cliques was described. A function that can be used to measure relative soft constraint satisfaction while at the same time provide a yardstick that depicts the absolute nature of hard constraints was proposed.

Chapter 7 Application and Comparison of a Heuristic Constrained Based System (COMBAT) to a Manual University Timetabling Instance and the Optime System.

This chapter describes the timetabling environment at USP. The proposed constrained directed reasoning system based on the generic model and heuristic algorithm discussed in the previous chapters is implemented as the Constraint Based Automated Timetabling System (COMBAT), a relational database that captures the domain and the constraints. The structured query language of the relational database is utilised in the constraint resolver. A heuristic algorithm in ordering constraints first discussed in Chapter 6, is further discussed and implemented in the system to solve the USP examination timetabling problem for semester one, 1999. A timetabling solution for USP's examination timetabling problem generated by the new model is evaluated for constraint satisfaction and compared with USP's manually built timetable. The COMBAT solution is also compared with the solution generated by OPTIME, an examination timetabling system developed by University of Nottingham's Automated Scheduling and Planning Research Group.

7.1 The University Of The South Pacific Timetabling Environment

The first computerized system for USP examination timetabling was developed by David Johnson (1990). His work focused on the book keeping aspects of timetabling and the use of heuristic algorithm to generate a timetable, but did not model constraints within a database and neither discussed nor proposed a standard data model. The USP timetabling environment is described next. Some of the description overlaps with Johnson's (1990).

USP serves 12 countries of the South Pacific. It operates three campuses – a main campus in Fiji with the School of Education, School of Pure and Applied Science and School of Social and Economic Development; a School of Agriculture in Western Samoa and a School of Law in a Vanuatu. There is no timetabling interaction between the courses offered by each school due to the large distances between the three countries. The USP also offers courses by distance and flexible learning modes. Distance learning courses are not timetabled with the on campus courses except for a few video broadcast courses and tutorials. Video broadcast courses (VBC) are offered at the main Fiji campus and transmitted live by video broadcast to remote distance and flexible learning support centres (DFLC). Some foundation level and first-year students of the Samoa and Vanuatu spend the foundation and/or the first year on the Fiji campus where they take courses offered by other schools. The only courses they may then take from their own school are online courses that do not need be timetabled.

USP has two main semesters per year for all graduate and postgraduate courses with each semester lasting 18 weeks. USP also offers a Masters of Business Administration programme. The students enrolled in this programme cannot enrol in courses in any other USP programmes at the same time. Hence, MBA courses do not influence the general USP timetabling process.

Each school at USP consists of different departments. Each department is responsible for the teaching of one or more disciplines. Each discipline has different courses, which are identified by codes. Each course can be offered at a different level, 1, 2 and 3 representing the year of study, for undergraduate course, level 4 for post-graduate courses, 5, 6, and 7 for Masters and PhDs. For example, CS111, is a Computing Science course at the 100 level, AF208, is and Accounting and Financial Management course at the 200 level while PH306 is a Physics course at the 300 level.

Each course has sessions allocated for teaching activities[1]. Lecturers, tutors, lab assistants, invigilators or supervisors are assigned to each session to be conducted. Those conducting a session will be called lecturer[2] in general. Each student is allowed to enrol for at least one and at most four courses per semester.

[1] Teaching activities are synonymous with examinations

[2] The term lecturer is synonymous with tutor, lab assistant, invigilator or supervisor.

Sessions are conducted in rooms, lecture theatres, laboratories, workshops and special places such as the swimming pool. For timetabling purposes, each of these venues is called a room. Rooms are identified by unique code numbers, for example, N111, U8 and S024.

Teaching activities begin on the hour at USP and last for 50 minutes after which there is a 10-minute break to allow students and lecturers to refresh themselves and move to other rooms for the next teaching activity. Some contiguous sessions such as three hour long science laboratory activities and examinations do not have a mandatory 10 minute break. However, the absence of the break makes no difference to the timetabling problem. The basic time unit (henceforward referred to as time-slot) is a combination of a day number and an hour number; for example, Thursday 10 am is day 4 hour 10 and Tuesday 3 p.m. is day 2 hour 15. A timetable slot is a combination of a room and a timeslot. A timetable is a set of sessions associated with timetable slots.

USP's course timetable is created by the chairperson of the Timetable Committee while its examination timetable is created by the academic administrative assistant. The course timetable is created before the students enrol. The students are given the timetable beforehand and are expected to select courses that do not clash. However, some clashes are inevitable such as those arising when a student has failed a 100 level elective course and wishes to repeat the course in the second year while doing other 200 level courses. A 200 level course may clash with a 100 level course if a student does not fulfil the general recommendation that students should finish all 100 level courses before embarking on higher-level courses.

In contrast, the USP examination timetable is built after the students have enrolled. While in course timetabling some clashes between courses are permissible, clashes between examinations must be avoided. The course and examination timetabling constraints are discussed next.

7.2 THE USP TIMETABLING CONSTRAINTS

This section discusses the USP course and examination timetabling constraints and shows how each constraint is specified in the proposed constraint based relational model.

At USP, the course timetable is developed before the student enrolments are known. The constraints used are developed from the structure of the programme of courses, the inter-relationship of courses between related disciplines, university regulations such as that all required 100 level courses should be finished before a student proceeds with a 200 level course, and previous enrolments trends of popular combination of courses. The popular combination of courses stem from popular choices of majors and minors. These include history and politics, accounting and economics, accounting and management, accounting and information systems,, biology and chemistry, computing science and information systems, computing science and mathematics, mathematics and physics, and so forth.

The following constraints were specified by the USP timetabling committee:

1. No lecturer can be at two places at the same time
2. There should be no clashes between the courses of the same level for specified disciplines. For example, it is unacceptable to schedule a 100-level mathematics course at the same time as 100-level accounting course where the mathematics course is a service course for the accounting course.
3. It is unacceptable to schedule a 200 and 300-level course for certain courses at the same time, for example a 200-level computing science course should not be scheduled at the same time as a 300-level information system course
4. Clashes between 100-level and higher-level courses can be permitted to certain extent since students are expected to finish all 100-level courses before embarking on higher-level courses.
5. Sessions of a course should be spread over the week
6. Sessions of a course should suitably be spread over the hours of the day. Students tend to be tired at the end of the day and hence it would unfair to timetable all the sessions of a particular course at the end of each day.

7. Some courses that are taught by part-time lecturers should be timetabled at certain times. For example, a part-time lecturer may be available only after 4 pm each day.
8. Some courses have certain lectures in common that should be timetabled at the same time.
9. Sessions that have large number of common students enrolled in both should not be timetabled at the same time.
10. Courses with many part-time students should be scheduled between 4 p.m. and 6 p.m.
11. VBC courses should be scheduled in VBC capable rooms.

The constraints for examination timetabling for semester 1 1999 were elicited from the academic administrative assistant. This particular semester has been selected for comparing the manually and the automatically generated timetables, because after semester 1, 1999, the academic assistant received computerized help in building the examination timetable. Semester 1, 1999 was the last occasion when the examination timetable was built completely manually. David Johnson (1990) has also enumerated a number of factors/constraints (the first five specified below) that should be considered for the examination timetabling problem for USP for semester 1, 1988. These constraints also apply to Semester 1, 1999.

1. The timetable must avoid all conflicts, so that no student has more than one examination at the same time; otherwise special invigilation arrangements have to be made. In practice, this is not difficult to achieve if enough examination-room capacity is available.
2. All examinations have to be completed in at most two weeks (20 sessions). This is an important consideration because time has to be allowed for marking the scripts and processing the results before the beginning of the next semester or the end of the academic year. The university administration is firmly of the view that the examination period cannot be easily extended.
3. It must be possible to accommodate all the candidates for the various examination rooms available. In theory, many examination rooms can be used; but in practice, only the larger rooms tend to be used, for ease of

administration and to avoid excessive invigilation requirements. The use of smaller rooms is more difficult to organise and administer and involves a proportionately larger number of invigilators. In 1988, the university had one large hall that could accommodate 400 students under examination conditions. Other large lecture rooms were used, mainly for DFLC studies examinations, but the aim was to confine the full time students to the large hall if possible at all.

4. Those examinations with a large number of students should come earlier in the examination period to allow maximum time for marking. The number of students per course varies from 400 for certain common first year courses (700 in year 2002) down to some five or seven for some final year options. To allow time for the examination results to be processed through the various boards and committees, all scripts should be marked by the middle of the next week following the examinations. For those examinations coming at the end of the examination period, only 4 days (including the weekend) are available for marking and it would be unrealistic to expect (say) 400 scripts (700 in 2002) to be marked in such a short time. Lecturers with large courses expect their courses to be early as possible and certainly within the first week.
5. Where a student is taking more than one examination, the examinations should be spread out over the two-week period if possible so that there is some time for preparation before each examination. In practice, this is virtually impossible to achieve for all the students, but at the very least they should not have to take two exams on the same day, and perhaps also an exam on the morning after one of the previous afternoon. Again, given enough examination-room capacity, same day conflicts can be usually eliminated but with limited capacity, they become more problematic.

Constraints 1, 2 and 3 are hard constraints that must be satisfied. Constraint 4 is desirable (soft) from the lecturer's point of view and constraint 5 is desirable from the student's point of view. Constraints 4 and 5 are conflicting constraints as the larger classes are popular classes with many students enrolled in two or more of the larger popular classes. From the lecturer's point of view, these large popular classes should be timetabled in the first week at least, while from the student's point of view, these large popular classes should be spread over the two-week period. A trade off between the lecturer and student satisfaction has to be determined.

7.3 THE MANUAL USP TIMETABLING METHOD

Before 1999, the USP examination timetabling system was computed manual except for the instance of semester 1, 1988. However, a check with the academic office did not reveal that any computerized system was used thereafter. University course timetabling was partly computerised but the building of the timetable was mostly manual. The computerisation consisted of storing timetabling data in an electronic format and using reports to produce lists of courses that had timetable clashes and list of rooms that were available for timetabling. The reports were used manually to modify the timetable to reduce conflicts and determine availability of rooms.

For both examination and course timetabling, an initial timetable for a particular semester was drafted from the corresponding semester's timetable of the previous year. Old sessions that were no longer needed were deleted first. New sessions were then added and allocated timetable slots while trying to satisfy the timetabling constraints. The manual system used two sets COUNT and CLASH as input. COUNT is a set of course codes and the number of students enrolled for each particular course. CLASH is a set of pairs of course codes and the number of students common to both courses in each pair. COUNT and CLASH are used in determining the number of clashes for a particular timetable schedule and to predict the extent of clash before candidate timetable allocations are attempted.
The manual timetabling process involved:

- Selecting a session to be timetabled
    - Selecting a set of candidate rooms
    - Selecting a set of candidate (empty) timeslot of the room (timetable slots)

    - Determining the sessions that were being held at the same timetable slot for each timetable slot in the candidate timetable slot list.
    - Determining the total number of clashes for each timetable slot.
    - Selecting a timetable slot that has no clashes
- Repeating above for other sessions

Clash lists provided by the academic office sometimes had inconsistencies arising from cases where a student had changed course enrolment but the academic records were not updated. Time and effort was spent in discovering and correcting these problems. There was no immediate online feedback on possible conflicts when selecting a timetable slot for a course. Clashes are determined by examining hardcopy reports. Due to the nature of the manual method, some clashes may be overlooked or deliberately scheduled at a time that the timetabler wrongly assumes has minimum number of clashes.

The timetabler has to determine the suitability of a timetable manually in respect to its constraints by examining the clash list and a room-booking list. This is a very tedious and time-consuming process. Some constraints may be neglected or not wholly evaluated. While heuristics may be used to search for suitable timetable slots manually to reduce the solution space, it is still difficult to apply the heuristic consistently over the large problem space manually.

7.4 A COMPARISON OF COMBAT WITH OTHER TIMETABLING SOLUTIONS

In this section, the COMBAT generated timetable is compared with USP's manual timetabling and with the University of Nottingham's OPTIME exam scheduler for satisfaction of the hard and soft constraints.

The examination timetable for semester one 1999 was built using COMBAT and compared with the manually prepared timetable of the same semester for the two important constraints - Day Spread and Time Spread. The Day Spread soft constraints state that examinations should be scheduled so that the event that a student has to sit two examinations on the same day is avoided. The Time Spread hard constraints state that examinations should be scheduled such that a student does not have to sit for more than one exam at the same time. The automatically generated timetable did not have any exams scheduled such that there were clashes at the same time. It reduced the number of same day clashes.

The timetabling data – the number of students per exam and the clash list is available at http://www.usp.ac.fj/macs/combat/ The total number of examination scripts was 17703.
The results are shown in Table 7.1.

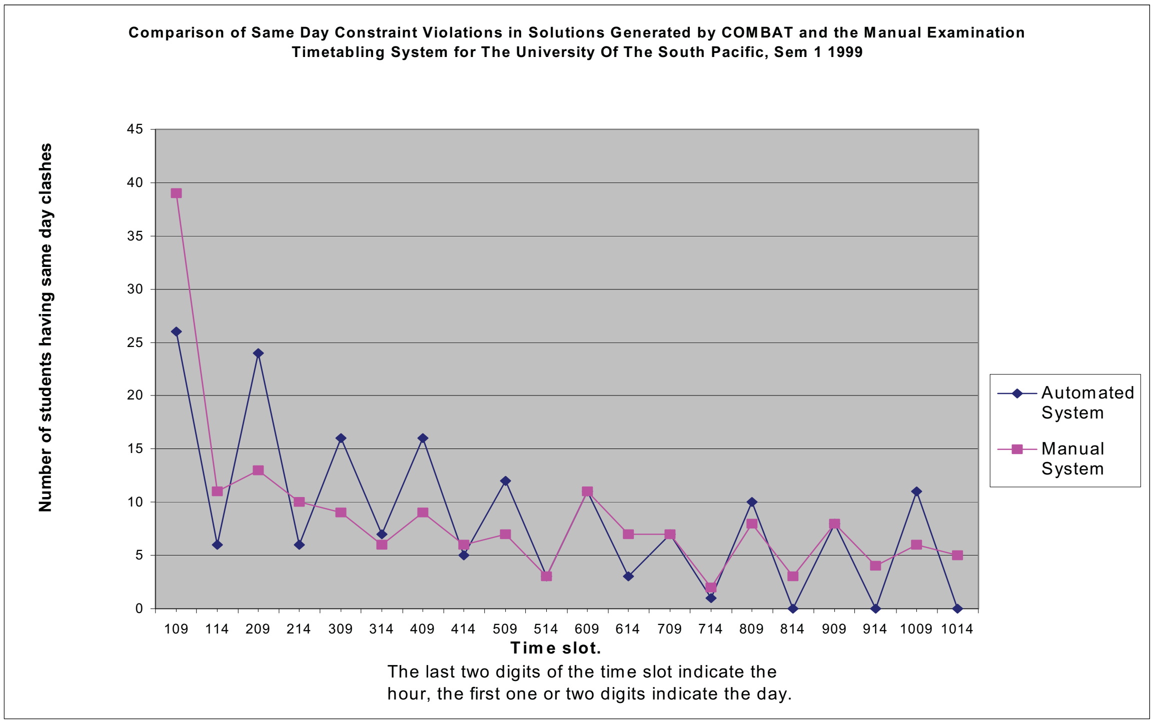
Table 7.1 Comparison of constraint satisfaction of USP's exam time table, semester 1 1999.

Number of exam scripts 17703

Number of clashing scripts			**Percentage of total scripts**	
	Manual	**Automated**	**Manual**	**Automated**
Same time conflicts	548	0	4.48	0
Same day conflicts. (for example, two exams on Monday, includes same time conflicts)	1170	399	9.56	3.26
Same day conflicts excluding same time conflicts (one exam in the morning and one in the afternoon)	622	399	5.08	3.26

In the manually prepared timetable, there were 1170 instances of students having to sit for two or more exams on the same day while in the automated timetable there were 399 instances only. Instances of students having to sit for two or more exams at the same time dropped from 548 in the manual system to zero in the automated system.

An important consideration in examination timetabling is to allocate exams to morning sessions when students are more alert and fresh. The automated timetable



allocates 82.0 percent of exams to morning sessions while the manual timetable allocates 62.7 % percent.

Figure7.1 Comparison of the COMBAT Solution with the Manual Solution

It is also desirable that exams are scheduled as soon as possible to allow more time for marking the scripts. In the automated timetable, 70 % of the exams are scheduled in the first week compared to the 65% of the manual timetable. Overall, the COMBAT solution satisfies all hard constraints that no student should have two or more exams scheduled at the same time, and reduces the number of exams scheduled on the same day. It also satisfies other satisfies constraints such as scheduling large classes the first week and lecturer preferences of exam dates.

The OPTIME timetabling scheduler is based on a simple heuristically guided randomised search described by Burke, Newall and Weare (1995). They outline a simple efficient technique based on heuristics that includes a variable degree of nondeterminism to facilitate a search through the solution space

The Heuristically Guided Randomised Search Method

Firstly, they order all the events by their sequencing strategy that dictates the following heuristics:

• "*Largest Degree First*". They schedule those events with the greatest number of conflicting events first. They expect these events to be more difficult to schedule.

• "*Largest Colour Degree First*". In this strategy, they schedule first those events with the greatest number of conflicting events that have already been scheduled. Generally, these would be more difficult to place in the timetable than those with little or no conflicting events already in the timetable. Largest Degree first is used to break ties.

• *"Least Saturation Degree First"*. They schedule first those events with the least number of valid periods currently available. This give priority to those events that have very few periods of time available, perhaps because a large number of their conflicting events have already been scheduled, or maybe because the number of suitable periods was limited to begin with. In either case, such events may prove difficult or impossible to schedule later on in the process. Largest degree first is again used to break ties. Having decided on their sequencing strategy, they take each event in turn and schedule it in the valid period where that event will cause least penalty (in terms of our objective function). If there is no valid period in which to schedule an event, they consider unscheduling some of the events already scheduled in order to make room for it. In that case, each period of the timetable is examined and falls into one of the following three cases:

1. The current event could not be scheduled in the period even if there were no conflicting events already in that period. Therefore, the period is no longer considered.
2. The current event could be scheduled into the period if a number of already existing events were to be removed from the period. If this is the case, the cost of scheduling in this period is the number of events that would have to be bumped from the timetable.
3. The current event could be scheduled into the period if a number of already existing events were removed. However, one of these events has already been bumped by the current event and will not be permitted to do so a second time. Therefore, this period is no longer considered. This is to avoid the

possibility of looping. From the set of periods that fall into case 2, they select the one that involves bumping the least number of events. If no periods at all fall into case 2, then the event is left unscheduled. On scheduling in this period, the existing events that were removed from the period are placed at the head of our sequence for rescheduling.

They proposed that, while a sequencing method such as the one above provides a quite acceptable solution, at least with a good choice of heuristic, it suffers in that it only produces a single solution which may not be as good as it could be. One exception to this is a technique that involves embedding a random element into the process to break ties. This meant that the entire process could be iterated a number of times with the best solution being selected. By introducing a random element into the pure heuristic method, they propose to be able to improve on the results found. This approach differs from the above method. Firstly, they do not implement a backtracking procedure, as this will not be as vital as when they generate a single solution and will add considerable extra time requirement in an iterated method. Secondly, instead of selecting the best event according to their heuristic, they use one of the following methods to select the next event.

• *Tournament Selection*. They generate a random subset of the events not yet scheduled and then choose the best event according to their heuristic from within this subset. For these experiments, they use tournament sizes of 5%, 10% and 20% of the full set of events.

• *Bias Selection*. Here they order their events according to their sequencing strategy but rather than picking the best event every time they select an event at random from the best n. Here they select from either the best 2, best 5 or best 10 events. As before they then schedule the chosen event in the valid period causing least penalty, or if no valid period exists, they simply leave it unscheduled. If there is a tie between periods, they choose the earlier of the periods. They then iterate this entire procedure a number of times, keeping track of the best solution found so far.

The OPTIME Constraint Model.

The OPTIME system was chosen to compare with COMBAT because it was the only one that we could obtain after an intensive search on the internet at the time of the experiment and moreover, OPTIME has been developed by the Automated Scheduling and Planning Group at the University of Nottingham. This group is the leading component of the foremost international series of conferences on the Practice and Theory of Automated Timetabling. Papers submitted to the conference are refereed and published in the proceedings of the conference. After the conference, selected papers are submitted for a second round of refereeing and successful papers are then published in Lecture Notes in Computing Science by Springer Verlag. For these reasons, we have selected the OPTIME system to compare with COMBAT.

The basic information about each exam to be scheduled by OPTIME is described next followed by the various ways in which each exam may be constrained.
Each exam has the following attributes:

- CODE
    - A code uniquely identifying each exam.
- TITLE
    - The verbose description of the exam paper.
- DEPARTMENT
    - The department code for with which the exam is associated.
- DURATION
    - The duration of the exam in hours and minutes. An exam can only be scheduled in a period of equal or greater duration.
- NUMBER OF STUDENTS
    - The number of students currently enrolled on the exam.

In addition, each exam may be constrained by the following:

- CONFLICTS WITH
    - This lists all other exams that the current exam conflicts with, together with the number of students involved in that conflict.
- PERIOD

  The period of an exam can be constrained as follows:
    - Use Any Period

    The exam will use any suitable period of the timetable.
    - Period Priority

This is a numeric preference for how early or late in the timetable you would prefer an exam to be scheduled. A value of 0 indicates the start of the timetable, 50 the middle and 100 the end etc.

- Use only Given Periods

  The exam may be given a list of periods in which it can be scheduled. The algorithm will then not schedule outside these periods under any circumstances.

- ROOM

  The room of an exam can be limited as follows:

  - Use any room

    The exam is allowed to use any room that is suitable and available.

  - Use only Given Rooms

    An exam may be limited to use only a certain room or number of rooms. If a room has been indicated as being only for use on request but appears in this list, it is assumed the algorithm is authorised to use that room.

  - Can use Given Rooms

    This merely indicates that the exam is allowed to use the listed rooms (which might otherwise be reserved). The exam however is not limited to the given rooms.

  - Use only Given Zones

    An exam may also be limited to use a given zone (a collection of rooms). You may wish to do this if you require an exam to take place in a certain area of campus, or if you have a splittable zone that you do not wish exams to use generally.

  - Requires exclusive access

    This option indicates that an exam must have a room to itself. This might be appropriate for instance, when an exam includes a multimedia presentation element.

- COINCIDENCES

  While these are referred to coincidences, this section includes other exam to exam constraints. These constraints are categorized as Spread constraints by the COMBAT model. The possible exam to exam relationships are:

- Schedule Before

  The current exam must be scheduled before the exams listed. The system will not allow the exam to be scheduled at the same time or later than those exams shown.
- Coincidences

  Coincidences are "Same time as..." relationships. The exam will be scheduled the same time as any exams listed here.
- Schedule After

  The converse of "Schedule Before"
- Exclusions

  This indicates that the current exam SHOULD NOT be scheduled the same time as the other exams listed. This is essentially the opposite of coincidences.

The exam timetabling data was fed into COMBAT and OPTIME with varying room constraints and four different timetables for the following four cases were built for each model. Cases 1, 2 and 3 are hypothetical cases while case 4 was the real-time case for the USP examination timetabling problem for semester one, 1999.

- **Case 1: One very large room, capacity 25000**
  - It is assumed that one very large room with capacity of 25,000 is available. This capacity is more than enough to accommodate all the examinations (17703) if there were no other constraints. Thus, this situation assumes that only clash and spread constraints and not room constraints will affect the final timetable. In this situation, the following questions will be answered: Given that, a room is also available at any given time, can either the COMBAT and OPTIME or both the models completely satisfy the hard and soft constraints? If not, then to what extent? This case also illustrates the ‘divide and conquer’ approach discussed in Chapter 6. The timetabling results of this case are expected to be sets of exams that have no common spread constraints.

- **Case 2: One very large room, capacity 1200**
    - Again a very large room is use, but not large enough to accommodate all the exams at one time, but more than large enough to accommodate the largest exam under one roof. In this situation, the following question will be answered: Given that a room is available to accommodate an exam of any size, can either or both the COMBAT and OPTIME models completely satisfy the hard and soft constraints? If not, then to what extent?
- **Case 3: Three rooms capacities 701, 440 and 220**
    - The room of size 701 is just large enough to accommodate the largest class. In this situation, the answers to the previous questions will be examined to compare if and how the increased room constraints affect the solutions built by each model.
- **Case 4: Three rooms, capacities 450, 440 and 220**
    - In this case, no room is large enough to accommodate the large exams. Exams with more than 450 students must be split into two or more rooms. In this situation, the following question will be answered: Given that, no room is available to accommodate the exams of size greater than 450, can either, or both the COMBAT and OPTIME models completely satisfy the hard and soft constraints? If not, then to what extent? This case represents the real-time domain and spread constraints of the USP examination timetabling problem of semester one 1999.

The timetables generated for each of the above cases were analysed for satisfaction of the following hard and soft spread constraints:

Hard Constraint

Same-time constraint: No student should have two or more exams scheduled at the same time. This constraint is the most important of all. It must be satisfied.

Soft Constraints

Avoid same-day exams - avoid scheduling exams with common students on the same day.

Avoid overnight exams - avoid scheduling exams with common students such that one exam is the last one and the other exam is the first one next morning. While it is possible for a student sit two exams at different times on the same day, it is better to spread the exams to allow time for rest and more study. While same-day and overnight exams are allowable, it is more important to avoid same-day exams then overnight exams.

It is expected that an acceptable timetable will satisfy the constraints in the following order:

- Same-time constraints (all satisfied)
- Same-day constraint (minimised violations, less than overnight constraints)
- Overnight constraints (minimised violations, but more than same-day constraints)

Results

The results for each of the four cases above are given in *Table 7.1* that shows the raw data and *Table 7.2* that shows the percentage of clashes of the total number of exam scripts (17,703).

Table 7.2 Constraint Violations by COMBAT and OPTIME exam timetable solutions

	Case 1		**Case 2**		**Case 3**		**Case 4**	
	COMBAT	**OPTIME**	**COMBAT**	**OPTIME**	**COMBAT**	**OPTIME**	**COMBAT**	**OPTIME**
Number of Students with unscheduled or clashing Exams	0	1117	0	1149	0	1489	0	3387
Number of Students affected by all same day clashes	630	327	562	482	588	369	432	308
Number of students with overnight exams	364	0	551	0	766	0	575	0

Table 7.3 Constraint violations by timetable solutions build by COMBAT and OPTIME

	Case 1		Case 2		Case 3		Case 4	
	COMBAT	**OPTIME**	**COMBAT**	**OPTIME**	**COMBAT**	**OPTIME**	**COMBAT**	**OPTIME**
% Students with Unscheduled or Clashing Exams %	0	6.3	0	6.5	0	8.4	0	19.1
% Students affected by same day clashes %	3.6	1.8	3.2	2.7	3.3	2.1	2.4	1.7
% Students with overnight exams %	2.1	0	3.1	0	4.3	0	3.2	0

Only COMBAT satisfies all the hard constraints. Although OPTIME satisfies the soft constraint of avoiding same day clashes better than COMBAT for cases 1 and 2, the difference is only 1.8% for Case 1 and 0.5% for Case 2. COMBAT satisfies the same day constraints better than OPTIME under the more constrained cases 3 and 4. COMBAT is 1.2% better in Case 3 and 0.7% better in Case 4. . Though OPTIME completely satisfies the least important soft constraint of avoiding overnight clashes, the difference is between 2.1 % and 4.3 %. However, COMBAT is between 6.3 % and 19.1 % better in satisfying the hard constraints. The results show that as the room constraints become more constraining, COMBAT continues to produce a consistently acceptable timetable while OPTIME solutions rapidly degenerate.
The COMBAT timetables satisfy the hard and soft constraints in the expected order of most important to least important, while OPTIME timetables satisfy the constraints, surprisingly, in the reverse order.

Satisfaction of same-time constraints

The timetable build by COMBAT did not have any exams scheduled such that any student had two exams scheduled at the same time, and all exams were allocated a time and venue except for certain postgraduate courses that did not have any clashes. These courses were left out of the timetabling problem, allowing the respective courses co-ordinators to select their own preference of time. Since these courses are

a very small number and USP has many small rooms, allocating a room to these exams is not a problem. The timetable build by OPTIME did not have any exams scheduled such that any student had two exams scheduled at the same time. However, it had several exams that were not scheduled. It appears that when OPTIME is not able to determine a clash-free timetable allocation, it leaves the exam unscheduled for the timetabler to decide the preferred timetable allocation manually. Figure 7.2 shows the number of students with no exams scheduled for COMBAT and OPTIME tables for varying room
constraints.

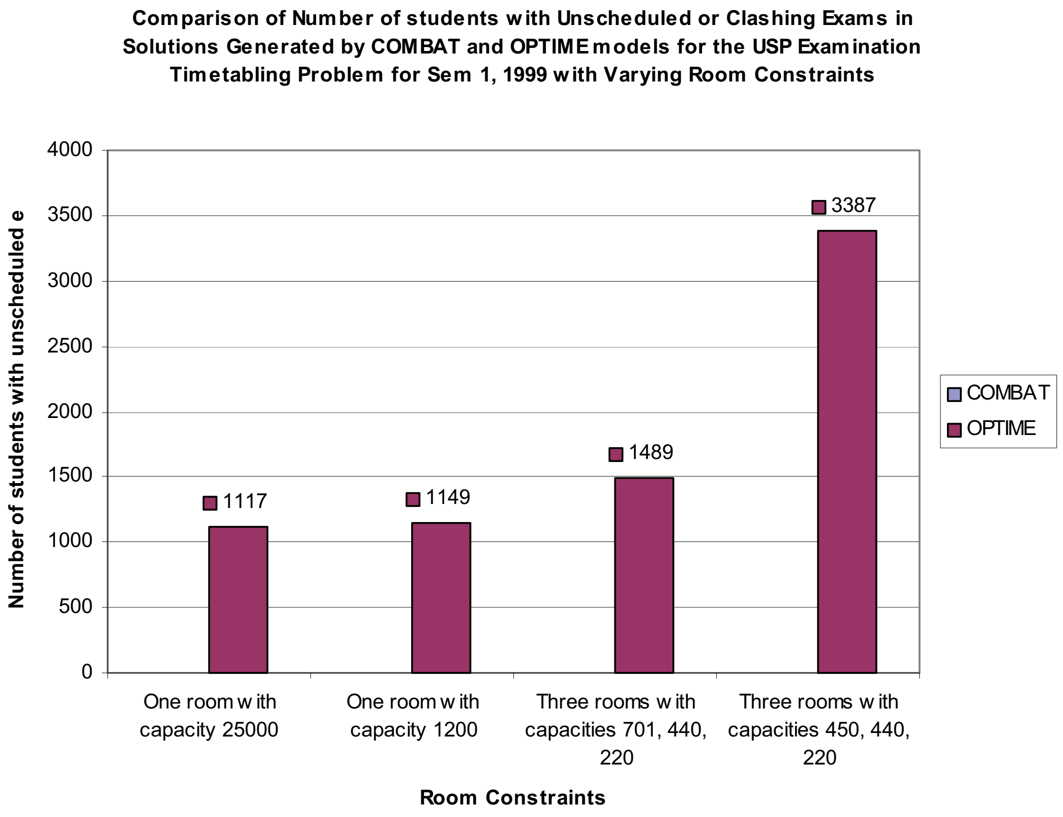


Figure 7.2 Comparison of number of students having unscheduled exams or clashing exams between COMBAT and OPTIME timetabling solutions.(note: COMBAT did not have any unscheduled students or any student with clashing exams)

Satisfaction of Overnight Constraints

As shown in *Figure 7.4*, OPTIME outperforms COMBAT in satisfying these overnight constraints. However, these are the least important constraints while COMBAT has satisfied the most important constraints by an even better difference of 6.3 % to 19.1 students with clashing exams. OPTIME has about 1100 students with unscheduled exams in Case 1. This figure increases dramatically when the number of large rooms decreases and number of smaller rooms increases. COMBAT has satisfied all hard constraints while OPTIME has not.

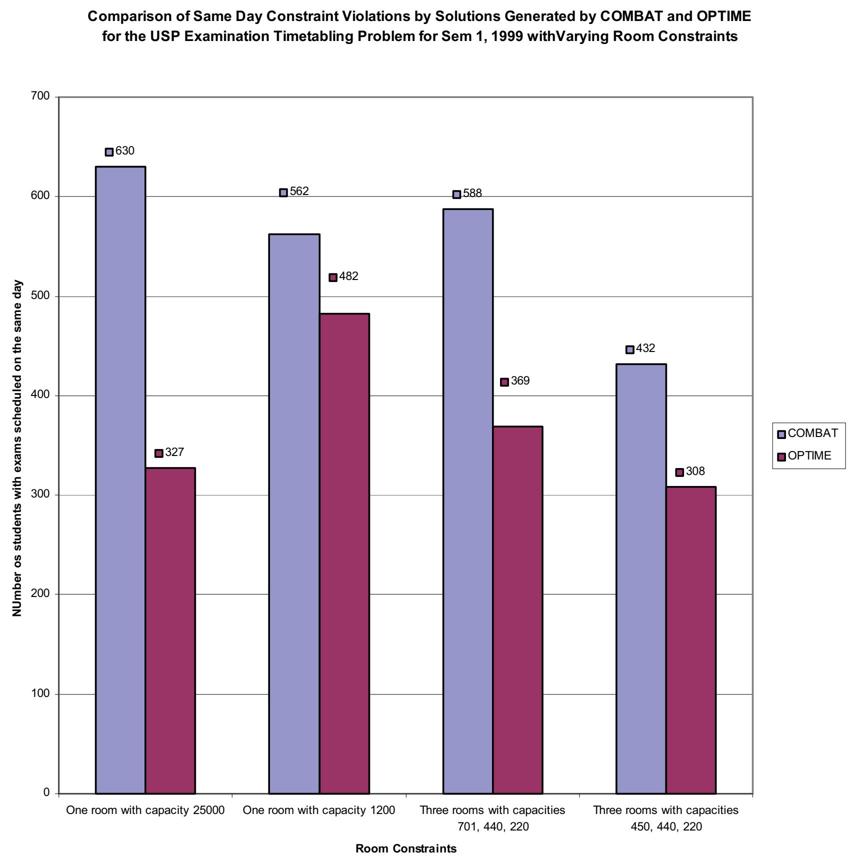


Figure 7.3 Comparison of same day constraint violations by COMBAT and OPTIME.

Satisfaction of Same-day Constraints

Figure 7.3 shows the result for the satisfaction of the first set of soft constraints that the number of students having two or more exams on the same day should be

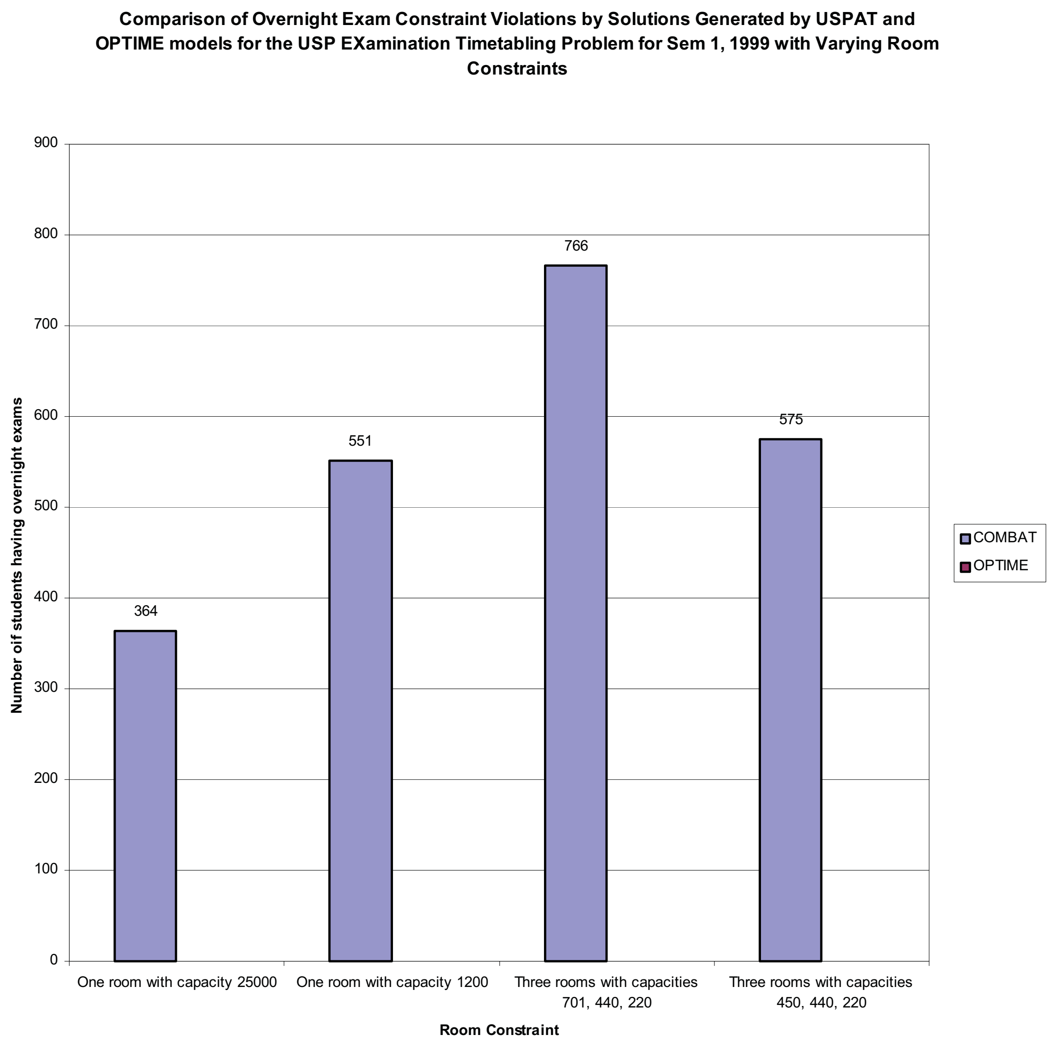


Figure 7.4 Comparison of overnight exam violations by COMBAT and OPTIME (note: OPTIME had no overnight violations).

7.5 SUMMARY

In the USP timetabling environment, the course timetable is developed before the students enrol and allows for some clashes. However, the examination timetable is built after the students have enrolled. It is imperative to avoid clashes in examination timetabling as opposed to course timetabling where some clashes are acceptable. The proposed constrained directed reasoning system based on the generic model and heuristic algorithm discussed in the previous chapters was implemented as COMBAT in a relational database that captures the domain and the constraints. The Structured Query Language of the relational database was utilised in the constraint resolver. A heuristic algorithm utilizing constraint ordering as discussed in the previous chapter was implemented in the system to solve the USP examination timetabling problem for semester one, 1999. A timetabling solution for USP's examination timetabling problem generated by the proposed model (COMBAT) was evaluated for constraint satisfaction and compared with USP's manually built timetable. The COMBAT solution satisfied all hard constraints (no same time exam clashes) and had far less same-day clashes than the manual timetable. It also scheduled all the large classes in the first week as required and had more students sitting exams in the morning than the manual timetable. The COMBAT solution was also compared with the solution generated by OPTIME -an examination timetabling system developed by University of Nottingham's Automated Scheduling and Planning Research Group based in its' School of Computer Science and IT. The COMBAT timetable satisfied all hard constraints whereas the OPTIME solution had many students with unscheduled exams. The COMBAT timetable satisfied the hard and soft constraints in the preferred ordered while the OPTIME solution satisfied them in the reverse order. As the room constraints were made more difficult, COMBAT consistently generated timetables with all hard constraints satisfied while OPTIME generated solutions where the number of constraint violations increased rapidly.

Chapter 8 CONCLUSION

This thesis examined various algorithms that have been used to solve timetabling problems. The algorithms act on computational models that are different for the various problems. In the models studied, only the timetabling resources were modelled but not the constraints. There exists a need for a standard timetabling model that incorporates both the resources as well as constraints, that can be subjected to different algorithms. This thesis showed that the various timetabling constraints could be specified using only three generic constraints: Domain, Spread and Count.

Domain Constraints

The generic domain constraint is

Domain((SessionList), (DomainList), Weight).

Domain constraints restrict the feasible domain for a session. Specific to USP timetabling , the domain constraints are –

- *Day((SessionList)t, (DayList), Weight)*
- *Hour((SessionList), (HourList), Weight)*
- *Time((SessionList), (TimeList), Weight)*
- *Room((SessionList), (RoomList), Weight)*

Spread Constraints

The generic spread constraint is

DomainSpread((SessionList), (DomainSpreadList), Order, Weight).

Spread constraints govern the how the feasible domain of a session is spread. The spread constraints of interest are those that involve time. Non-time spread constraints can be specified as domain constraints and hence are utilized to restrict the number of domain elements allocated to a group of sessions.

Specific to the USP timetabling environment, the spread constraints are

- *DaySpread((SessionList), (DaySpreadList), Order, Weight)*
- *HourSpread((SessionList), (HourSpreadList), Order, Weight)*
- *DayHourSpread((SessionList), (TimeSpreadList), Order, Weight)*
- *RoomDayHourSpread((SessionList), (RoomDifferenceList), Order, Weight)*

While *DayHourSpread* constraint can be specified as Day and Hour constraints, it is being used because it is more intuitive and results in a smaller number of constraints.

Count Constraints

The constraint

CountResource((SessionList), (ResourceList), CountExpression, Weight)

can be used to specify all limitations on the number of resources or timetabling objects scheduled.

It may be argued that some meaning is lost by generalizing constraints, for example *TimeSpread((S1, S2), (0), 0, -N)* (the sessions should be scheduled at the same time) may mean that the two sessions have students in common or are taught by the same lecturer. However, it should be noted that standardizing involves eliciting the general essence of a problem devoid of local factors or idiosyncrasies. The proposed model can be extended to include constraints that are more verbose. However, they would resolve into the core proposed constraints.

It has been shown that the various timetabling constraints can be specified using the proposed generic model and is applicable to both course and exam timetabling. The proposed format can be easily applied in languages such as Lisp, Prolog and constraint logic languages, as well as relational databases. The model was successfully implemented for the University of South Pacific's examination timetabling problem for the last few years using Ms Access and Visual Basic. The author hopes to build a database of instances of timetabling constraints specified in the proposed model and converters for translating data to and from the different current formats.

The proposed model allows constraints to be easily compared. This property is exploited by developing a constraint based heuristic reasoning method that eliminates the need to back track. The system (COMBAT) was successfully applied to the USP examination timetabling problem that led to the reduction of same time clashes to nil, and large reduction in the number of same day clashes compared to the manual timetable. In comparison with OPTIME, COMBAT satisfied all hard constraints while OPTIME was not able to scheduled certain exams

Thus, the answers to the research questions are

- Is it possible to standardise the university timetabling data format such that it is applicable to all reasonable university timetabling instances?
    - Yes, a generic model COMBAT was developed that standardised all reasonable types of timetabling data including the grouping and sub groupings of time and rooms.
- Is it possible to standardise university timetabling constraints such that all reasonable constraints in one timetabling instance can be expressed in any other timetabling instance?
    - Yes, COMBAT allows constraints to be standardised such that all reasonable constraints from different timetabling instances are expressed in COMBAT.
- Can a generic model of the university timetabling problem that incorporates both the timetabling data and constraints be developed?
    - Yes, both, timetabling data and constraints are represented within COMBAT, within a single database.
- Can a timetabling algorithm be successfully applied to such a generic model?
    - Yes, a constraint based heuristic algorithm was applied by COMBAT to solve a timetabling problem.

Further work involves extending the proposed generic model to include secondary school class timetabling and developing an intelligent interface that allows timetable developers to state constraints in everyday verbose language that are translated into

COMBAT format. Work is in progress for a calculus of university timetabling constraints to allow the simplification of constraints and pre-processing of constraints to determine constraint conflicts and non-satisfiability of constraints.

BIBLIOGRAPHY

1. Abramson D, Dang H, School Timetables: A Case Study Using Simulated Annealing, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 1993

2. Bardadym A, Computer-Aided Lessons Timetables Construction. A Survey, Management Systems and Computers Vol 8, 1991

3. Bardadym A, Computer-Aided School and University Timetabling: The New Wave. Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling, 1995

4. Barham A, Westwood J, Simple Heuristic to Facilitate Course Timetabling, Journal of the Operational Research, Vol. 29, 1978

5. Burke E, Petrovic S, Recent Research Directions in Automated Timetabling European Journal of Operational Research – EJOR, 2002

6. Burke E, Elliman D Weare R, A Genetic Algorithm for University Timetabling, Proceedings of the AISB workshop on Evolutionary Computing, University of Leeds, 1994

7. Burke E, Elliman D, Ford P, Weare R, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling, 1995

8. Burke E, Jackson J, Kingston J, Weare R, Automated Timetabling: The State of the Art, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling, 1995

9. Burke E, Newall J, Weare R, A Simple Heuristcally Guide Search for the Timetable Problem. Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling, 1995

10. Burke, E, Kingston J, Pepper P, A standard data format for timetabling for timetabling instances Practice and Theory of Automated Timetabling II Springer-Verlag LNCS 1998

11. Burke, E.K., Newall, J.P., Weare, R.F., A Simple Heuristically Guide Search for The Timetable Problem. Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling 1995

12. Carter M, A Survey of Practical Applications of Examination Timetabling Algorithms, Operations Research, Vol. 34 1986

13. Carter M, Laporte G, Recent Developments in Practical Examination Timetabling, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling, 1995
14. Chang K, Jeung W, Investigations of a Constraint Logic Programming Approach to University Time tabling, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling, 1995
15. Cole A.J. The Preparation of Examination Timetables using a Small Store Computer Comp Jrnl 1964
16. Collingwood E, Ross P, Corne D, A Guide to GATT University of Edinburgh, 1996
17. Cooper T, Kingston J, The Complexity of Timetable Construction Problems, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling, 1995
18. Corne D, Ross P Fang H, Fast Practical Evolutionary Timetabling, Proceedings of the AISB Workshop on Evolutionary Computing, 1994
19. Fahrion R, Dollanski G, Construction of University Faculty Timetables Using Logic Programming Techniques, Discrete Applied Mathematics, Vol. 35, 1992
20. Fahrion, R. and Dollanski G. Construction of University Faculty Timetables Using Logic Programming Techniques, Discrete Applied Mathematics, Vol. 35 1992
21. Foxley E., Lockyer K, The Construction of Examination timetable by Computer TheComputer Journal, 11 1968
22. Garey M, Johnson D, Computers and intractability - A Guide to The Theory of NP-Completeness, W.H. Fremann and Company 1979
23. Goltz H, Matzke D, University Timetabling Using Constraint Logic Programming Practical Aspects of Declarative Languages, LNCS 1551, SpringerVerlag 1999
24. Gudes E, Kuflik T, Meisels A, On Resource Allocation by an Expert System, Engineering Applications of Artificial Intelligence, Vol. 3, 1990
25. Gudes E.; Kuflik T. and Meisels A. On Resource Allocation by an Expert System, Engineering Applications of Artificial Intelligence, Vol. 3, pp. 101-109, 1990

26. Gueret C, et al Building University Timetables Using Constraint Logic Programming, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling, 1995

APPENDIX: TIMETABLING CONSTRAINTS.

Burke *et al* (1994) delineates some of the most common types of timetabling constraints:

- **Resource Assignment**
  A resource may be assigned to a resource of a different type, or to a meeting. For example, a lecturer prefers to teach in a particular room (for whatever reason), or an exam should take place in a particular building. Specific to USP, an exam such as LL321 must be held in the language lab.
- **Time Assignment**
  A meeting or a resource may be assigned to a time. This constraint can be used to specify days on which a teacher is unavailable, or to pre-assign a time to a particular meeting. Specific to USP's examination timetabling, certain examinations such CS122 should be held on a day between Tuesday and Friday only.
- **Time Constraints between Meetings**
  Common examples of this class of constraint are that one particular lecture must take place before another one, or a set of exams must be sat simultaneously. Specific to USP, certain examinations such as CS391 and CS491 should be held together because they have common components.
- **Meeting Spread**
  Meetings should be spread out in time. For example, no student should have more than one exam in any day and lectures of a course should be spread over different days.
- **Meeting Coherence**
  These constraints are designed to produce more organised and convenient timetables, and often run contrary to "meeting spread" constraints (above). For example, a lecturer prefers to have all his lectures in three days, giving him two lecture free days.
- **Room Capacities**
  The number of students in a room may not exceed the room's capacity.

- **Continuity**

  A continuity constraint is any whose main purpose is to ensure that certain features of student timetables are constant or predictable. For example, lectures for the same course should be scheduled in the same room, or at the same time of day.

Goltz *et al* (1999) states the following constraints for the Medical Faculty at the Humboldt University timetabling problem:

C 1: The teaching activities can begin every quarter of an hour and may run for different lengths of time.

C 2: There are restrictions with respect to the starting time for each teaching activity.

C 3: Two lectures about a same subject must not be scheduled for the same day.

C 4: The lectures, seminars, and practicals of each group must not overlap.

C 5: The number of seminars and practicals that a department (research group) can conduct at a certain time is limited.

C 6: Some teaching activities are only held during certain weeks, where the following cases are possible: all weeks, A-weeks (weeks with odd numbers: 1, 3, 5,…), B-weeks (weeks with even numbers: 2, 4, 6,…), either A-weeks or B-weeks.

C 7: There are lectures about the same subject carried out in parallel at the parts of the faculty. Thus, a student can choose from among parallel lectures.

C 8: Timetabling also involves allocating rooms to the individual lectures.

C 9: Some lectures can only be held in certain specified rooms; in some cases, only one room matches the requirements.

C 10: The varying distances between the different locations of the teaching activities must be taken into account.

C 11: The starting time preferences for teaching activities and the preferred rooms for lectures should also be taken into account if possible.

The following constraints were specified by the USP timetabling committee:

1. No lecturer can be at two places at the same time
2. There should be no clashes between the courses of the same level for specified disciplines, for example, it is unacceptable to schedule a 100 level

mathematics course at the same time as a 100 level accounting course where the mathematics course is a service course for the accounting course.

3. It is unacceptable to schedule a 200 and 300 level course for certain courses at the same time, for example a 200 level computing science course should not be scheduled at the same time as a 300 level information system course
4. Clashes between 100 level and higher-level courses can be permitted to certain extent since students are expected to finish all 100 level courses before embarking on higher-level courses.
5. Sessions of a course should be spread over the week
6. Sessions of a course should suitably be spread over the hours of the day. Students tend to be tired at the end of the day and hence it would unfair to timetable all the sessions of a particular course at the end of each day.
7. Some courses that are taught by part-time lecturers should be timetabled at certain times. For example, a part-time lecturer may be available only after 4 pm each day.
8. Some courses have certain lectures in common that should be timetabled at the same time.
9. Sessions that have large number of common students enrolled in both should not be timetabled at the same time.
10. Courses with many part-time students should be scheduled between 4 p.m. and 6 p.m.
11. VBC courses should be scheduled in VBC capable rooms.