NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

A LOGIC APPROACH TO CONFLICT RESOLUTION IN UNIVERSITY TIMETABLING

By

Le Kang

Thesis Submitted to
the School of Graduate Studies
in Partial Fulfillment of the Requirement for
the Degree of Master of Computer Science*

at the
University of Ottawa

*The Master of Computer Science program is a joint program of the University of Ottawa and Carleton University, administrated by the Ottawa-Carleton Institute for Computer Science

ISBN 0-315-60003-9

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ABSTRACT

A computerized timetabling system developed at University of Ottawa is presented. The system is built on a logic programming model which uses first order logic to define first order and second order constraints in timetabling. Information about courses, professors, and student programs is collected for each academic year and used in the process of constructing timetables. The time schedule produced by the system takes into account course conflicts, professor availability, professor teaching preferences, pre-assignments, classroom location choices, and many other important factors that affect its user satisfaction level. Along with the system's ability to include factors such as professor availability, professor teaching preferences and classroom location, the analysis of test results at the University of Ottawa shows a large improvement in the time utilization and seating usage of classrooms, compared to the corresponding timetables that are produced by the traditional manual processes.

Chapter 1 introduces the problem of timetabling and gives a overview of the background of the thesis. In chapter 2, literature published in the field is reviewed. Theoretical and practical details about the project are given in chapters 3 and 4. Aspects about the testing stage are included in chapter 5 and finally in chapter 6, some future research directions are proposed.

ACKNOWLEDGMENTS

I just do not know how to express my thanks towards my supervisor, Dr. George White. Without all the guidance, advice and encouragement that he has given me throughout my graduate studies, I would not have been able to finish my project, let alone this thesis.

I am grateful to Ms Pauline Belanger, the scheduling officer of the University of Ottawa, for her enthusiasm and effort put into the project.

I would like to thank Mr. George H. von Schoenberg, the Registrar of the University of Ottawa, for his support, co-operation and many useful discussions.

I acknowledge the financial support in the form of research assistantship from the Registrar's Office at the University of Ottawa. This support makes my study at the University of Ottawa possible.

I owe all that I have accomplished in my life to my parents, Mr. Hengzhong Kang and Mrs Mengli Liu, who are both ten thousand kilometers away on the other side of the earth physically and right beside me emotionally. They have taught me to put my heart in everything I do and to be cheerful in life.

Finally, I want to express my deepest thanks to my wife, Ning Wang, for her patience, support and confidence, especially during the more than two years when we were separated by the Pacific Ocean. I would not have survived the life alone in Canada without knowing that she is always with me wherever I go.

TABLE OF CONTENTS

Chapter 1 INTRODUCTION

Timetabling, or constructing timetables, is a task which arises frequently in organizations where groups of people have to schedule a number of meetings at various times in a set of rooms, such as offices where meetings have to be organized, and in educational institutions where classes must be scheduled. *[TT 88]* The basic timetabling problem, closely related to *examination timetabling*, can be defined as the problem of assigning a set of courses to a set of rooms during a fixed number of time periods such that no *critical conflicts* exist. Critical conflicts, or first order constraints, are conflicts that are commonly agreed as unacceptable such as two different classes being assigned to a room at the same time, or two required courses of a student being scheduled at same time. This type of conflicts also includes those conflicts that are defined as unacceptable according to actual circumstances and institutional policy. Non-critical conflicts, or secondary constraints, have been introduced in most practical applications. These conflicts depend mainly on local practical conditions and can not be completely eliminated but should be reduced as much as possible.

It is very important for educational institutions, especially large ones like universities, to build timetables which use resources efficiently and achieve a high degree of satisfaction among users, i.e. professors and students. The efficiency of resource usage can be evaluated by two indices, **seating usage**, which tells how much room space is used when the room is occupied, and **time utilization**, which evaluates the proportion of time the room is occupied. By improving these two indices, an institution can either schedule more classes or keep fewer rooms for teaching and convert surplus classrooms to space for other purpose. Given the fact that most North American universities are experiencing space shortages because of increasing enrollment or other kinds of expansions,

improving the efficiency of timetables has, not surprisingly, become one of the major concerns of many university administrations.

Building good timetables in educational institutions, especially in large ones like universities or colleges is an unsolved problem that has been studied for many years, from both theoretical and practical aspects. Manual timetabling is the approach that has been used for many years by all universities, and it is still the major way of scheduling in most universities. The number of teachers and courses along with the resulting conflicts in university timetabling is so large that building a new timetable each year/term becomes a task that is too complex and too large for manual processing. It has also been proved from the past experience that putting more people into the manual scheduling process can neither guarantee a better timetable nor justify the extra cost. With the amount of work and time demanded, people have come to settle for some heuristic ways of getting things done at the price of efficiency and user satisfaction. One very popular way is using timetables of previous years with some minor adjustment. The assumption of this method is that academic programs offered by a university are basically the same for years, so the courses offered in each year should also be almost the same as those of previous years. It is very obvious that some serious defects exist in this method. First of all, the efficiency, as well as the level of user satisfaction, of the timetable will be undoubtly lowered each time this method is applied. If a course has changed teacher(s), or the attendance of a course is larger than that of the previous year, the timetable will fail to accommodate the new requirements resulting from these changes. Secondly, this method may leave many new courses unschedulable. Due to the inter-connected nature of timetables, a small change in academic programs can bring a wide swing in the new timetable. This usually requires re-scheduling a large part of the old timetable to maintain the schedulability of those courses whose characteristics have been changed. So the task of accommodating small changes in academic programs can be too complex to be handled manually. Finally, this method is inapplicable in situations where all or a large portion of the courses of an institution have to be re-scheduled, for example, when a university changes its academic calendar from quarters to semesters.

With the rapid development in computer science in the last thirty years, almost all large educational institutions have large-scale computers and many areas in the administration have already gained an advantage from these powerful tools. Personnel records management, among others, can be cited as a good example. Many institutions and scientists have put a lot of effort into the field of automating, or computerizing, the process of constructing timetables, hoping to reduce the amount of work required by the manual process and to provide timetables which use teaching resources more efficiently and satisfy more user requirements. Results have been generally disappointing, though a considerable amount of work has been done in the area. This situation is reflected in the current

practice of constructing timetables at North American universities, where the majority are still using the traditional manual approach and subsequentially producing timetables which allocate teaching resources inefficiently and causing a lot of unhappiness among teachers and students. Some universities have developed their own computerized timetabling systems which either are not complete or ignore a large number of constraints from both administration and user level. The status of the commercial software market also tells the same story. Not only is the number of software packages on the market very small, but none of them offers a satisfactory solution that is acceptable to a reasonable number of institutions. Most of these systems still require considerable manual time and effort. *[TT 13]*

The main motivation of this thesis is the development and implementation of a computerized timetabling system at the University of Ottawa. Since the summer of 1987, sponsored by the Office of the Registrar at the University of Ottawa, we have been working towards building a computerized timetabling system for the University. We have developed a new approach which is based on a logic programming model and the result is very promising. It is expected that the system will be put into practical use in very near future, possibly the next academic year, 1990-1991, at the University of Ottawa.

To help one to obtain a good overview of this thesis, we will now first introduce the background of our project, present the motivation, and then describe the organization of the thesis. Analysis of the nature of the timetabling problem and reviewing of work done in the field are given in the next chapter.

1.1. TIMETABLING AT THE UNIVERSITY OF OTTAWA

The University of Ottawa, the oldest and largest bilingual (English and French) institution in North America, has about 2000 teaching staff and an enrollment of 25,000 students. Within every academic year, from September to the following August, there are three academic semesters, Fall, Winter, and Spring. A two month Summer term, July to August, exists for mainly part-time and co-operative students. There are around 5000 sections of 3000 courses offered in each of these three semesters. The University's Scheduling Office, which is under the supervision of the Registrar's Office, is responsible for creating timetables for the university community.

The current timetabling at University of Ottawa is still largely a manual approach, in which initial timetables are built manually and some in-house developed computer programs are used to help

schedule individual courses or events interactively. It is an on-going process which involves two full time staff at the university level and at least one secretary from every department/faculty office on a part time basis throughout every academic year.

Under the current system, timetabling is carried out in two stages, time scheduling and room allocation. At the first stage, only courses and their teaching time are decided, while at the later stage classrooms are assigned to these courses. A new timetable is first defined by copying the courses and their teaching times from the timetable of the previous year. Copies of the report of this information are sent to all departments and faculties. Timetabling officers at department or faculty level will make changes on this report and send it back to the Scheduling Office for verification and data entry. Numerous phone calls between the Office and departments have to be made to resolve questions and to allocate new changes.

The teaching space allocation, i.e. assigning proper classrooms (including lecture rooms and laboratories) to classes, is done by the timetabling officer at the Scheduling Office, after agreements have been reached with departments/faculties about what and when courses are going to be offered. This process is largely a manual one that requires a lot of time and tedious work. With some help from several software packages which search for un-occupied rooms and display their information, the officer has to spend a lot of time on selecting suitable rooms for all classes and do a large amount of tedious checking work to avoid conflicts in the assignment.

One factor that has given timetabling officers a lot of headaches is the multi-length time slot system. Currently, there are more than a dozen course formats available. Though most of the course fall into the the slot system of one and half hour a block, many other different length formats exist, such as one-hour, two-hour, two-and-half-hour, three-and-half-hour, four-hour, four-and-half, five-hour, six-hour, seven-hour, eight-hour, eight-and-half-hour, nine-hour, and three-times-one-hour. The university has been striving towards a standard slot system, which will require all courses to follow the one-and-half hour block system, i.e. the length of a course has to be a multiple of one-and-half hour.

Three editions of timetables will be published in every academic year. The first edition is published in the early April of the academic year, i.e. on April 1, 1989 for the academic year 1989-1990. It contains all courses offered at the undergraduate and graduate levels during the fall and winter sessions, as well as courses with room changes for January and half-courses beginning in the Winter session. This edition is mainly designed to inform students what courses are going to be offered and when they are taught. Room information is not included in the edition, as it is not available at the

publication time. The second edition will be published in the late August. Besides information contained in the previous edition, it includes the room numbers where fall classes are given. The third and last edition of the university timetable, published in early December, contains regular academic year full courses continuing through to next April with room changes in January and all half-courses beginning in January. The room numbers for lectures and laboratories are included.

All changes occurring as well as new requests for classrooms through the academic year should be reported to and handled by the Scheduling Office, which coordinates with timetabling officers at the department/faculty level in making changes in timetables and allocating rooms to non-regular activities such as meetings or special seminars.

1.2. MOTIVATION OF THE PROJECT

With a proposal from the Registrar's Office at the University of Ottawa, we started the project of developing a Computerized Timetabling System(CTS) for the University in the summer of 1987. One of the major objectives of the project is to achieve better opportunities for students to register in the courses they need to finish their programs, without encountering scheduling conflicts. The large number of conflicts created by the large number of courses and different student programs make it almost impossible for manual scheduling to find a feasible solution that accommodates requests from all student programs. We believe that a computerized system can solve this problem faster and better than the manual process does.

Achieving more efficient utilization of the available classrooms is another motivation for which CTS has been developed. The room utilization and seating usage of the timetable currently being used at the University of Ottawa are 50% and 69.5% respectively. "Classrooms are considered effectively utilized at 70 percent or more of the available hours" and "Optimum seat utilization is in the 80 to 95 percent range". *[TT 30]* These two figures show that "there is surplus space in the University of Ottawa classroom pool". *[TT 30]* Our project intends to raise these two indices and give the University the opportunity to convert the surplus space into other urgently needed space.

CTS is also designed to enable optimal placements in terms of campus geography. Limitation of manual scheduling prohibits our current timetables from taking into account the factor of campus geography. In CTS, courses are placed into classrooms according to "road maps" defined by departments and faculties.

The final objective for which our project has been undertaken is to reduce the manual work for departments, faculties and the Scheduling Office. By handing this tedious checking work to computers, CTS will eliminate a large part of the manual work currently carried out by timetabling officers. Timetabling officers will have more time to deal with professors and students, get to know their requirements better, and prepare data that reflects these needs more precisely. Consequently the user satisfaction level will be improved, in addition to other positive results.

Most research related to the timetabling problem is based on the extensively studied *vertex coloring problem* in graph theory and almost all practical applications are based on some modified graph coloring heuristic. *[TT 13]*. Other approaches include *mathematical programming* and *semi-interactive methods*. In our project CTS, we have adopted an approach of applying logic programming with constraints in building timetables. CTS has some very useful characteristics, such as:

(i) *modularity:* The central part of a timetabling system is the structure which describes constraints and how these constraints are activated during the scheduling process. CTS is well modularized so that constraints can be easily modified, added, deleted, or tested.

(ii) *portability:* CTS is implemented using Waterloo-Prolog, a very simple and basic version of Prolog. As only standard Prolog functions are used, the system could be easily transformed into other versions of the Prolog language. Coupled with modularity, this feature enables CTS to be easily adopted by other institutions.

(iii) *applicability:* One major intention of developing computerized timetabling systems is to improve the efficiency of teaching resource usage as well as the level of user satisfaction. CTS, according to our tests at University of Ottawa, produce timetables that behave better in both aspects than those built manually.

CTS is specifically designed for the Registrar's Office of the University of Ottawa. Several other programs have been developed as supporting packages so that CTS fits into the current scheduling system and takes advantage of the existing software at the University. Not only will the new timetables not have to copy from the previous ones and sacrifice the efficiency and user satisfaction, but also the University has a tool to test policies and patterns of room facility distributions by running CTS with sets of data with different attributes.

1.3. ORGANIZATION OF THE THESIS

Chapter 1 gives a brief review of the background of the timetabling problem, describes the current practice of building timetables at University of Ottawa, and introduces the general methodology of our approach to the problem.

In Chapter 2, literature on related work is surveyed. All major algorithms researched before are described and both their advantages and disadvantages are evaluated. We will also review some major practical applications which either are currently advertised on the market or are reported as being used in one or more universities. We do not intend to include all work done in the field, we have, however, made effort toward a relatively complete and comprehensive review of all significant algorithms and applications.

Chapter 3 presents the model of our computerized timetabling system. All first and second order constraints and their expression in first order logic are described in detail. Our solution to these constraints is also presented. The structure of the implementation system, the detailed algorithm and its characteristics are given in Chapter 4.

All matters concerning the test stage are given in Chapter 5. These includes the test background, the test result, and the result analysis. Comparisons with several other major algorithms are also reported in this chapter.

Finally in Chapter 6, we conclude the thesis and propose some possible future research in this field. Chapter 7 is the bibliography. The detailed system structure, the flowchart of the main scheduling program, and the input file structures are included in the appendices.

Chapter 2 LITERATURE REVIEW

2.1. INTRODUCTION

The timetabling problem has been receiving considerable attention from researchers for a very long time; so it is not surprising to see that much literature has been published on the subject. By reviewing previous work, we hope to establish a background for introducing our approach to the problem and comparing our system to other practical applications as well as giving a deeper insight of the nature of the problem.

The first reported computer programs to attack the timetabling problem are closely related to *manual timetabling*. These programs are actually a kind of bookkeeping program, which maintain the lists and tables classically used. Elementary insert and remove techniques were designed and applied to the basic data structures used by these programs. The famous process of reducing the availability array proposed by Gotlieb might be the first nonheuristic attack on the problem. *[TT 21] [TT 40]* Even after later refinements, Lions and Dempster were able to give examples showing that it could not guarantee the elimination of all pseudo-availabilities, or the production of results with all reasonable input data. *[TT 24] [TT 52]* Many elaborate heuristic techniques appeared when people started realizing the complexity of the timetabling problem. Many of the early heuristics were under the influence of the Gotlieb approach. *[TT 5] [TT 6]* This research simulated the process of manual tabletabling and seems to be the first group who applied interchange operations if scheduling a class failed at the first attempt. There are many other systems which follow the same direction of heuristic approach. In fact, "*an amount of heuristics may be found in every known timetabling system*". *[TT 70]*

In an effort to reduce the complexity of the problem, manual timetabling is usually carried out in two separate stages, *time scheduling*, where courses are scheduled into a fixed number of time periods, and *room scheduling*, which assigns suitable classrooms to classes. This is especially true in large institutions. Many research models proposed for computerization reflect this division by focusing on only one of the two stages, while others cover both. Naturally we could classify computation models for timetabling into three classes, a) those focusing on time scheduling; b) those concerning only room scheduling; and c) those dealing with both aspects. The well-known examination timetabling problem can be included into the first group, time scheduling, as it only deals with courses and examination time, or time periods.

Another traditional way of classifying groups the models by the algorithms proposed along with them. With this classification, all models fall into one of four classes, graph theoretical methods, linear optimization methods, interactive methods, and logic programming, which is the approach we are using in designing and implementing our system.

In the following sections of the Chapter, literature related to the problem of timetabling is reviewed. The first of the two classifications will be used in reviewing the models and the second for reviewing the algorithms. In Section 2, we will review several typical models of the timetabling problem and discuss the factors that contribute to the problem's complexity. Section 3 contains an overview of different algorithms proposed and studied in the literature and Section 4 presents brief descriptions of a number of practical applications.

2.2. MODELS OF TIMETABLING

Since we are focusing on the factors included in the computing models, the coverage or extent of a model becomes more important than the algorithm used to implement this model. We will group models using the first classification.

2.2.1. Models of Time Scheduling

The Course Scheduling problem is the simplest type of Time Scheduling. The basic problem consists in assigning each lecture to some period of the week in such a way that no student is required to take more than one lecture at a time. Assuming that students are classified into groups that follow exactly the same curriculum, this type of problem only deals with courses and time

periods. It is quite similar to the *examination scheduling problem*, which is to cast an examination timetable such that all students may be examined and no student is required to write more than one examination simultaneously. Only courses and time periods are concerned in this type of problem and the sets of teachers and rooms are irrelevant.

Given a set of m courses C, course c_i consists of t_i lectures of one period each; the number of available periods is p; the students are divided into r groups S_1, ..., S_r such that in each S_j all students take exactly the same courses; and, l_k is the maximum number of lectures which can be scheduled at period k, the course scheduling problem will be to satisfy the following constraints: (*CS*)

$$\sum_{k=1}^{p} x_{ik} = t_i \qquad (i = 1, ..., m)$$

$$\sum_{i=1}^{m} x_{ik} \le l_k \qquad (k = 1, ..., p)$$

$$\sum_{i \in S_j} x_{ik} \le 1 \qquad (j = 1, ..., r, k = 1, ..., p)$$

where $x_{ik} = 1$ if a lecture of course c_i is scheduled at period k, and $x_{ik} = 0$ otherwise.

Another basic model for time scheduling is the class-teacher model, in which teachers as well as courses and time periods are involved *[TT 82]* A *class* will consist of a set of students who follow exactly the same program. Each class has a number of lectures which are taught by certain teachers. With a set of classes, a set of teachers, and a set of time periods, the purpose of this scheduling model is to assign each lecture to some periods in such a way that no teacher is involved in more than one lecture at a time. This has become known as the simple timetabling problem, *T(R)*.

Let $C=\{c_1, ..., c_m\}$ be a set of classes and $T=\{t_1,...,t_n\}$ a set of teachers. Assume that all lectures have the same duration. We have an $m \times n$ requirement matrix $R=(r_{ij})$ where r_{ij} is the number of lectures involving class c_i and teacher t_j. If we define x_{ijk} to be 1 if class c_i and teacher t_j meet at one time k and 0 otherwise, the basic class-teacher model will be to solve the following problem: *[TT 82]*

$$\sum_{k=1}^{p} x_{ijk} = r_{ij} \qquad (i=1, ..., m; j=1, ..., n)$$

$$\sum_{j=1}^{n} x_{ijk} \le 1 \qquad (i=1, ..., m; k=1, ..., p)$$

$$\sum_{i=1}^{m} x_{ijk} \le 1 \qquad (j=1, ..., m; k=1, ..., p)$$

$$x_{ijk} = 0 \text{ or } 1 \qquad (i=1, ..., m; j=1, ..., n; k=1, ..., p)$$

The necessary and sufficient conditions for the existence of a solution to the above problem are:

$$\sum_{i=1}^{m} r_{ij} \le p \qquad (j=1, ..., n)$$

$$\sum_{j=1}^{n} r_{ij} \le p \qquad (i=1, ..., m)$$

Thus there is a timetable in p periods if and only if no teacher (no class) is involved in more than p lectures.

The basic *T(R)* problem can be extended to allow for classes and teachers to be available for more than once per period. *[TT 17] [TT 82]* The *period* in *T(R)* may stand for some hour of the day in a daily scheduling problem or some day of the week in a weekly scheduling problem. So the extension allows classes and teachers to be available for more than once in any day of the week in a weekly scheduling problem. In the extension, there is an a_i for each class c_i representing the maximum number of lectures in which c_i may be involved during any one of the p days (respectively, there is a b_j for each teacher t_j). The solution of this problem will be finding values x_{ijk} satisfying: [*T(R)*]

$$\sum_{k=1}^{p} x_{ijk} = r_{ij} \qquad (i=1, ..., m; j=1, ..., n)$$

$$\sum_{j=1}^{n} x_{ijk} \le a_i \qquad (i=1, ..., m; k=1, ..., p)$$

$$\sum_{i=1}^{m} x_{ijk} \le b_j \qquad (j=1, ..., n; k=1, ..., p)$$

$$x_{ijk} \ge 0 \ \ integer \qquad (i=1, ..., m; j=1, ..., n; k=1, ..., p)$$

The necessary and sufficient conditions for the existence of a solution to such a problem is that:

$$\sum_{i=1}^{m} r_{ij} \le pb_j \qquad (j=1, ..., n)$$

$$\sum_{j=1}^{n} r_{ij} \le pa_i \qquad (i=1, ..., m)$$

This extended model still does not fully reflect the reality in Time Scheduling. In practice there are additional requirements which have to be taken into account.

De Werra proposed a formulation of daily scheduling problem that takes into account the requirements of *pre-assignments* and *unavailabilities*. *[TT 82]* In the formulation, the constraints of

unavailabilities are reduced to preassignments by stating that if t_j is unavailable at period k, there is a meeting of t_j with a dummy class c_i^* at period k and similarly for a class c_i which is unavailable at some periods.

Let $\bar{x}_{ijk} = 1$ if class c_i and teacher t_j meet at period k for a lecture which is not a preassignment and $\bar{x}_{ijk} = 0$ otherwise. Now the constraints for scheduling become:

$$\sum_{k=1}^{p} \bar{x}_{ijk} = \bar{r}_{ij} \qquad (i=1, ..., m; j=1, ..., n)$$

$$\sum_{j=1}^{n} \bar{x}_{ijk} \le \bar{b}_{ik} \qquad (i=1, ..., m; k=1, ..., p)$$

$$\sum_{i=1}^{m} x_{ijk} \le \bar{c}_{jk} \qquad (j=1, ..., m; k=1, ..., p)$$

$$\bar{x}_{ijk} = 0 \text{ or } 1 \qquad (i=1, ..., m; j=1, ..., n; k=1, ..., p)$$

where

$$\bar{r}_{ij} = r_{ij} - \sum_{k=1}^{p} r_{ijk}$$

$\bar{b}_{ik} = 1$ if c_i is available and not preassigned at period k, 0 otherwise

$\bar{c}_{jk} = 1$ if t_j is available and not preassigned at period k, 0 otherwise.

Though in some very special cases, it can be decided if such a timetable exists or not *[TT 82]*, Evan, Itai and Shamir have proved that the problem of deciding whether a solution exists for the scheduling problem is NP-complete. *[TT 31]* In other words, there is no algorithm that can be guaranteed to terminate in polynomial time. This fact explains the phenomenon that "*work to date on timetabling problem has mostly concentrated on finding heuristic approximations to the solution which give reasonable values using reasonable amounts of computing time.*" *[TT 88]*

2.2.2. Models of Room Scheduling

Using teaching utilities, especially space resources, has become a much more important issue in most universities than ever before. Not too long ago, many institutions could still ignore the problem of room limitation when producing a timetable. The only thing they were concerned with was the time conflicts of courses and professors. Past experience told them that they would almost always find a proper classroom for a course in such a timetable because the space resources were more than enough. Things have changed. In many institutions, assignment of rooms "*has become the dominant difficulty since the epoch of building programs has ended but the enrollment increases have not.*" *[TT 88]*

Assignment of classrooms is usually carried out along with the process of time scheduling, but it has been studied independently from time scheduling in some practical applications. *[TT 14] [TT 16] [TT 35] [TT 38] [TT 39]*

The formulation of the problem is normally expressed in terms of the variable x_{ij}, a 0-1 valued variable that is equal to 1 when class c_i is assigned to room r_j. Constraints for such a problem are: *[TT 14]*

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad (i = 1, 2, ..., n)$$

$$\sum_{i \in p_k} x_{ij} \leq 1 \qquad (j = 1, ..., m; k = 1, ..., p)$$

where p_k represents the set of all classes that meet in period k. The first constraint ensures that every class is assigned to some room while the second constraint prevents any room from having more than one class in the same period.

This problem is NP-hard even when the problem has only two periods. *[TT 16]* There are several heuristics for solving this problem proposed in the literature. They will be reviewed in Section 2 of this chapter.

2.2.3. Models of Timetabling (Time-Room Scheduling)

Timetabling is a process which assigns classes to appropriate time periods and classrooms. It combines the processes of *Time Scheduling* and *Room Scheduling*.

The timetabling problem, in its most general form, involves a set of courses, a set of time periods, a set of student programs which represent sets of courses that certain groups of students have to take, a set of teachers and a set of classrooms. The elements of these sets should be scheduled in such a way that:

(a) no teacher is required to be in two or more places simultaneously;

(b) all classes are scheduled;

(c) all classrooms are large enough to hold the classes assigned into them.

(d) no two or more courses of one student program are scheduled into the same time period, i.e. no student is required to be in two or more places simultaneously;

White and Wong have given a precise definition of the problem *[TT 88]* : let there be m periods, $k = 1, ..., m$; n_2 teachers, $j = 1, ..., n_2$; n_1 classes, $i = 1, ..., n_1$; and n_r rooms, $l = 1, ..., n_r$. There exists a requirements matrix $R = (r_{ij})$ and two vectors. E is an n_1-vector whose ith element is the enrollment in class i and C is an n_r-vector whose ith element gives the capacity of room i. The problem will be to determine whether there is a function

$$S = \{ x_{ijkl} \}: \{ 1, ..., m \} \times \{ 1, ..., n_1 \} \times \{ 1, ..., n_2 \} \times \{ 1, ..., n_r \} \rightarrow \{ 0, 1 \}$$

where $x_{ijkl} = 1$ if and only if teacher j meets class i in room l during period k such that:

$$\sum_l \sum_k x_{ijkl} = r_{ij} \qquad (i = 1, ..., n_1; j = 1, ..., n_2) \tag{1}$$

$$\sum_j x_{ijkl} \leq 1 \qquad (i = 1, ..., n_1; k = 1, ..., m; l = 1, ..., n_r) \tag{2}$$

$$\sum_i x_{ijkl} \leq 1 \qquad (j = 1, ..., n_2; k = 1, ..., m; l = 1, ..., n_r) \tag{3}$$

$$\sum_l x_{ijkl} \leq 1 \qquad (i = 1, ..., n_1; j = 1, ..., n_2; k = 1, ..., m) \tag{4}$$

$$\sum_j \sum_k E_i \, x_{ijkl} \leq \sum_j \sum_k C_i \, x_{ijkl} \qquad (i = 1, ..., n_1; l = 1, ..., n_r) \tag{5}$$

In this formulation, constraint (1) requires that the number of meetings of class i and teacher j conforms to the specifications of the requirements matrix; constraint (2) specifies that each period-class-room has at most one teacher involved; constraint (3) specifies that each teacher-period-room has at most one class in it; constraint (4) specifies that each teacher-period-class occupies at most one room; and constraint (5) specifies that all rooms are large enough to contain the classes scheduled into them.

It is obvious that this is also an NP-complete problem. *[TT 31] [TT 88]* Many different algorithms, or heuristics, have been published, giving solutions for some special cases. *[TT 13] [TT 18] [TT 37] [TT 52] [TT 57] [TT 61] [TT 62] [TT 71] [TT 72] [TT 73] [TT 79] [TT 87]* In these special cases, new models are formed by adding different local constraints over and above those discussed in this general model proposed by White and Wong. Of these algorithms many have never left the experimental stage while some have had some actual use. In the next sub-section, we will discuss the major factors contributing to the complexity and difficulty of the problem.

2.2.4. Timetabling Complexity

From the discussion in above sections, it is very easy to get the impression how complex even a very restricted model can be and how difficult the computation of finding a solution may get. Actually the problem of timetabling is of such size and complexity that no known method can guarantee optimality by any reasonable criteria. Indeed, apart from exhaustive search, no known method can guarantee the existence of even one solution.

There are several factors contributing to the complexity of timetabling.

i) The number of computations is very large. When the number of course sections is relatively large, (>1000), the number of constraints becomes so large that many algorithms become unpractical or impossible to implement. *[TT 46]* Furthermore, if some constraints, such as professor preference and professor availability, are taken into consideration, things get worse and the computing time may become unacceptably long;

ii) Building a timetable involves some psychological as well as pedagogical constraints. A good timetable should incorporate not only a feasible arrangement of physical resources, such as classroom and other academic utilities but also resolution of psychological constraints which can be illustrated by the case where a professor may want to teach only in a specific classroom at a specific time. Furthermore, almost any resolution of timetabling problems require some compromises, between departments, for example; *[TT 20] [TT 52]*

iii) Local variance is so large that it is almost impossible for two institutions to have the same model. In a practical environment, each institution has its own regulations about producing timetables. Not only are the procedures different, but also the fundamentals of timetables such as the format of the timeslot system, vary from one university to another.

White and Wong have given a detailed list of problems inhibiting the practical implementation of algorithms into the real world: *[TT 88]*

(i) Some classes must be held in specially equipped rooms. Typically these are laboratory subjects, such as physics and chemistry, as well as physical education and music and some others. Thus certain subsets of courses must be scheduled into certain subsets of rooms;

(ii) Several courses may be grouped for a certain fraction of the year. Conversely, certain courses may be split into several sections for tutorial purposes or for practical reasons;

(iii) Some teachers and some rooms may not be available for certain periods because of fixed administrative or other duties;

(iv) Some courses may be required to be held at the same time to enforce a prerequisite requirement or for another reason. If course A is a prerequisite for course B, then scheduling A and B at the same time prohibits students from registering in both courses simultaneously;

(v) Some courses must be considered as pre-assigned. These are courses given by some external lecturers and core courses which are taken by large numbers of students in diverse programs. Many mathematics course fall into this category, especially introductory calculus and algebra course taken by engineering, science and administration students;

(vi) Some institutions cast their timetables after students have registered for courses. The timetable then attempts to arrange things so that each student can follow his desired course of studies. This is unsuitable for students who have commitments elsewhere and for whom the time of day is the determining factor in whether to register in a course or not. This may not be a factor for secondary schools but for large city universities with a large working part-time student constituency, it may be very important;

(vii) Courses that meet more than once per scheduling cycle (usually this means more than once a week) may have problems with the span of time between sessions. Some institutions make no restrictions at all; some prohibit a span which includes the noon meal while others require that all sessions of a course fall in a time "slot";

(viii) Some institutions put some restrictions on their timetables that students should not be forced to attend classes consecutively more than a certain number of times and that lectures of a professor should be scheduled in a way that the professor does not have to teach more than a certain number of times in a row. Some other institution may have some more relaxed rules which allow classes (or lectures) arranged in consecutive time periods but these classes (lectures) should be in classrooms within a certain distance so that students (teachers) who are taking (teaching) these classes (lectures) together do not have to rush a long way to another location after they finish one class (lecture);

(ix) In some institutions, more than one time slot system are allowed. This not only reduces the efficiency of the final timetable in which there are many "holes" left unused, but also increases the difficulty of the timetabling work;

(x) Modern institutions live in a world of changing budgets, mobile staff and uncertain students. These changes often bring changes to the timetabling system and sometimes may force much work of changing to be done manually.

Of course, this is not a complete list and we could add a lot more to it. But it should be able to show how complex the task of designing a model and an algorithm to find a solution can be.

2.3. ALGORITHMS FOR TIMETABLING

Algorithms designed along with different timetabling models can be divided into four types:

i) graph theory-based algorithms;

ii) mathematical programming-based algorithms;

iii) those involving certain "human factors", or interactive approaches;

iv) logic programming-based algorithms.

Applying logic programming in the field of computerized timetabling is a new approach which appeared in recent years and there is not much literature published on this subject, though logic programming itself has received intensive study. The project that will be described in detail by the thesis follows this concept. In a later part, we will pay a lot of attention to logic programming as well as to the literature published on the subject. So only the other three types of algorithms are discussed here.

2.3.1. Graph Colouring Heuristics

Many different algorithms have appeared in the literature. Most of them are based on or are related to the extensively studied vertex coloring and edge coloring problem in graph theory. Given the basic *time scheduling* problem $T(R)$, we could construct a corresponding graph as follows:

i) each vertex, or node, represents a class c_i or a teacher t_j;

ii) vertex c_i and vertex t_j are linked by r_{ij} parallel edges.

If each time period corresponds to a colour, the problem consists in finding an assignment of one among p colours to each edge of the graph in such a way that no two adjacent *edges* have the same color. This is an *edge* coloring problem.

Some models can be formulated as *node* or *vertex* coloring problems. With the *course scheduling problem, CS*, we can construct a graph as follows:

i) each lecture is represented by a vertex;

ii) an edge connects two vertices if the corresponding courses have been taken by at least one student group.

An acceptable solution will be that each node receives one of p colors and no two adjacent nodes share the same color.

The graph coloring problem is usually formulated in one of two ways. *[TT 87]*

(1) Can the vertices of a graph be colored using a set of p colors such that no two adjacent nodes or edges are assigned different colors?

(2) To determine the minimum number of colors required to completely color the nodes (edges) of a graph such that no two adjacent nodes (edges) share the same color. This unique integer is called the *chromatic number* of the graph.

Some methods have been designed to find a coloring of a graph with p colors if such a coloring exists. *[TT 8] [TT 64] [TT 66]* They are called *exact algorithms*. These methods are feasible for implementation only when the number of courses, n, is very small as they enumerate implicitly all possible colorings, which increases exponentially as n increases.

Besides these exact algorithms, various heuristic methods have been developed for coloring a timetabling graph. An excellent review on graph coloring heuristics can be found in *[TT 28]*. The heuristics discussed below are those relevant to practical applications.

As they color one node (or edge) at a time, most of these coloring heuristics may be considered as based on an assignment strategy: *[TT 82]* At each step a lecture is selected according to some rule and then a time period is assigned to this lecture according to some criterion. Different methods differ from each other in their lecture selection rules and period selection criteria.

The *degree of freedom* is the most commonly used metric. In a graph, the degree of freedom of a vertex or node is the number of edges adjacent to this vertex. In practice, the degree of freedom of a course may have different meanings. It can be, for instance, the number of courses which may not be scheduled into same time period with the current one; or it may be taken as the remaining number of periods that this course could be scheduled into.

A well-known method, which is called *largest degree first*, consists in ordering the nodes according to the number of their neighbors, the degree of freedom, from large to small. *[81]* Based on the rationale that the nodes with most neighbors will be the hardest to color if their neighbors are colored before them, the coloring process proceeds by selecting courses from the top of the list and assigning the "lowest numbered" non-conflicting color.

In a method proposed by Peck and Williams the nodes are also sorted by degree, the list is scanned through from top to bottom to place as many courses as possible in the first time slot, which is the lowest colour. *[TT 63]* Further scans then go back to the top of the list and fill the second period, etc. This method is called *largest degree first: fill from top*. There is a similar algorithm

called *largest degree first recursive: fill from top*. It is slightly different in that as each node is removed from the list, all the remaining nodes are re-evaluated, their degrees of freedom are re-calculated, and the list is resorted using the new data.

Williams stated that the degree of freedom was not sufficient to determine how difficult a course was to schedule and that a course was critical only if a large number of its neighbors were critical. *[TT 89]* He proposed a new formula for computing the critical property of a node:

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

where A_i is the set of nodes adjacent to the node v_i and $d_0(v_j)$ is the degree of node v_j. Hence $d_1(v_i)$ is the sum of the degrees of the nodes adjacent to v_i. He normalizes the $d_1(v_i)$ values and then computes $d_2(v_i)$, $d_3(v_i)$, ..., $d_{k+1}(v_i)$ recursively. Their values will stabilize after a while. Williams shows that this is more expensive, but better than the simple *largest first* heuristic in terms of the number of colors used. This method is named as *largest modified degree first*. It has been shown that the first formula, $d_1(v_i) = \sum_{j \in A_i} d_0(v_j)$, was sufficient to beat the simple largest first methods, but further recursion for $d_k(v_i)$ did not result in significant improvements. *[TT 28]*

Smallest degree last methods are based on the same rationale as the *largest degree first* methods in that nodes of lowest degree are easy to color and they should be colored last in sequence. The basic procedures of this heuristic type are to remove those nodes with lowest degree from the graph and place them at the end of the list, the degrees of the remaining nodes are recalculated, and then the lowest degree nodes are removed and repositioned. When the list is complete, the coloring process proceeds from the top of list as in the *largest degree first* heuristics. *[TT 52]*

The performance of the above method can be improved when it is combined with a bichromatic interchange procedure. The interchange rule is as follows: *[TT 55]*

(i) the node on the top of the list is assigned to the lowest numbered conflict-free color which has already been used;

(ii) if the node conflicts with all of the current colors, find a color k_i for which there is only one conflicting course c_j. If course c_j can be recolored, it is given another color. Otherwise, search for a "bichromatic interchange" involving a subset of courses for each of two colors (one of which should contain c_j) such that interchanging the subsets is feasible, and none of the courses in the second subset, which will be recolored to color k_i, conflict with the new course which is under consideration. If no such interchange can be found, a new color for the course is created and the whole process continues.

Among the simple heuristic methods, the *DSATUR* method seems to perform best on average. Brelaz initially colored any node with the largest possible number of neighbors with the smallest possible color. *[TT 8]* For each node x there is a set $F(x)$ of forbidden colors (these may be colors which have already been used by neighbors of x); at each step we choose a node y for which the cardinality $|F(x)|$ of $F(x)$ is maximum and color it with the smallest possible color. The rationale is that the node with largest cardinality of $F(x)$ is the node with the least number of possible colors to choose from, and that the node of highest degree with the least number of remaining colors is the most difficult and should be colored first. Computational results on large random graphs done by Brelaz has proven that this method is superior to the previous heuristic algorithm in terms of the number of colors required. Metha demonstrates that the number of colors required to color the graph may be further reduced if a bichromatic interchange procedure is used along with this method. *[TT 55]*

2.3.2. Algorithms Based on Mathematical Programming

If the timetabling problem is looked at from the perspective of reducing existing constraint violations, i.e., first order and/or second order conflicts, it can be presented as an optimizing problem and many special requirements can be easily included into the formulation. *[TT 1] [TT 7] [TT 16] [TT 35] [TT 38] [TT 39] [TT 51] [TT 56]* This consists of a group of algorithms based on mathematical programming methods (linear programming, integer programming, transportation problem, and network flow problem). These algorithms are also called *improvement algorithms*. *[TT 4]*

The major drawback of this approach is that the number of variables and constraints becomes unmanageably large for practical size problems. In an attempt to reduce the problem size, researchers have proposed several different models. For example, Bloomfield and McSharry, Almond, and Selim propose heuristic methods. *[TT 2] [TT 7] [TT 72]* Mulvey suggests a network model, *[TT 57]* Akkoyunlu, Carter, and Tripathy use Lagrangian relaxation approach, *[TT 1] [TT 14] [TT 80]* and Ferland and Roy have proposed a formulation that leads to a quadratic assignment problem. *[TT 35]*

The coverage of these models are also different. Mulvey's model combines the problems of time scheduling and room scheduling, *[TT 57]* Ferland and Roy propose solving them in sequence and iterate if necessary, *[TT 35]* while Carter's formulation concerns itself only with room scheduling. *[TT 14]*

Here we give several typical formulations which represent the different approaches of algorithms based on mathematical programming.

Lawrie formulated the timetabling problem as an *integer linear programming* model in terms of *layouts*. *[TT 51]*

"The layout is a statement of the curriculum and its organization for a group of students, generally a year group, i.e. all the students in the first or other year of the school." All data is expressed in a number of layouts, generally between 4 and 6 in total. Once layouts have been obtained, the problem of constructing the timetable is solved in two separate stages:

1) Producing an "outline timetable" by overlapping layouts with one another in such a way that restrictions are not violated during any period of the week.
2) Permuting the outline timetable to meet requirements on distribution of classes and sets in various subjects over the week.

An arrangement is defined as a set of columns, one from each layout, which does not violate any restriction. Then the overlapping procedure can be seen as finding as many arrangements, not necessarily all different, as there are periods in the week. Let b_i = the number of occurrences of column i specified in its layout; $a_{ij} = 1$ if column i appears in arrangement j, 0 otherwise. The linear programming formulation for the problem becomes:

$$\sum_{j=1}^{N} a_{ij} x_j = b_i \qquad (i = 1, ..., m)$$

where the matrix of coefficients (a_{ij}) is a zero-one matrix and has in general between 85% and 90% zeros.

Lawrie presented two computing options. His first version generates all variables prior to the linear programming run which incorporates Gomory's Method of Integer Forms and uses an objective function, *maximise* $\sum_{j=1}^{N} c_j x_j$, where c_j is defined as the minimum of the b_i values of the columns in arrangement j and hence is an upper bound for x_j. *[TT 51]*

In the second version, variables are generated only as required, no objective function is used, and Gomory's Method of Integer Forms is supplemented by an *ad hoc* procedure which first rounds a rational solution to integer values and then completes the partial solution thus obtained using an enumerative procedure.

Tripathy in 1980 proposed a model which exploits the underlying network structure of the timetabling problem. *[TT 80]*

Let $K=K_1, ..., K_q$ be a set of q courses and K_i consist of k_i lectures of one period each; There are p periods; Students are divided into r groups $S_1, ..., S_r$ such that in each S_l all students take exactly the same courses; l_k corresponds to the number of available classrooms, i.e. the maximum number of classes which can be given at period k; and $y_{ik} = 1$ if a class of course K_i is scheduled at period K and $y_{ik} = 0$ otherwise; then we have an objective function:

$$Max \sum_{i=1}^{q} \sum_{k=1}^{p} C_{ik}\, y_{ik} \qquad \text{T(1)}$$

$$s.t. \sum_{k=1}^{p} y_{ik} = k_i \qquad (i = 1, ..., q) \qquad \text{T(2)}$$

$$\sum_{i=1}^{q} y_{ik} \le l_k \qquad (k = 1, ..., p) \qquad \text{T(3)}$$

$$\sum_{i \in S_l} y_{ik} \le 1 \qquad (l = 1, ..., r, k = 1, ..., p) \qquad \text{T(4)}$$

$$y_{ik} = 0 \ \ or \ 1 \qquad \text{T(5)}$$

The C_{ik} in $T(1)$ are costs measuring the desirability of $y_{ik} = 1$; in other words, if assigning a class of course K_i into period k is highly desired, C_{ik} should be given a high value.

The Lagrangean relaxation technique can be used when constraint T(4) is incorporated in the objective function T(1):

$$\max \sum_{k=1}^{q} \sum_{k=1}^{p} C_{ik}\, y_{ik} + \sum_{l} \sum_{k} \lambda_{kl} \left(1 - \sum_{i \in S_l} y_{ik} \right) \qquad \text{T(6)}$$

This objective function combined with constraints $T(2)$, $T(3)$ and $T(5)$, consists of a capacitated *transportation problem*. In Tripathy's algorithm, the values λ_{kl} are obtained with a subgradient optimization method combined with a Branch and Bound procedure.

The timetabling problem is decomposed into a time scheduling problem and a room scheduling problem in Ferland and Roy's project. *[TT 35]* First a time schedule is determined, then a feasible room assignment table is attempted. If an assignment cannot be found for the time schedule, the input data for time scheduling are modified accordingly and a new schedule is generated. Each subproblem is formulated as a problem of assigning conflicting activities to resources, or a 0-1 mathematical programming problem, which is transformed into an equivalent *quadratic assignment program* with 0-1 variables by introducing the complicating constraints into the objective function via penalties.

As the two subproblems have similar formulation, only that for time scheduling is reviewed here. For each class (activity) i and time period (resource) j, the decision variable x_{ij} is defined as: $x_{ij} = 1$ if class i is assigned to time period j and $x_{ij} = 0$ otherwise. The formulation is as

follows:

$$Min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}\, x_{ij}$$

$$s.t. \quad \sum_{j=1}^{m} x_{ij} = 1 \qquad 1 \le i \le n \qquad \text{F(1)}$$

$$x_{ij} + x_{kl} \le 1 \qquad l \in J_{ijk},\ k > i,\ 1 \le i \le n,\ 1 \le j \le m \qquad \text{F(2)}$$

$$x_{ij} = 0 \ \text{or}\ 1, \qquad 1 \le i \le n,\ 1 \le j \le m \qquad \text{F(3)}$$

where J_{ijk} is the set of time periods for class k which generates a conflict with the assignment of class i to time period j; c_{ij} is the preference for the assignment of class i to time period j:

$$c_{ij} = \begin{cases} k & \text{if } \textit{period } j \textit{ is ranked } k \text{ for } \textit{class period } i,\ 1 \le k \le 3 \\ n \cdot m+1 & \text{if } \textit{period } j \textit{ is ranked } 4 \text{ for } \textit{class period } i \end{cases} \qquad \text{F(4)}$$

Constraint F(2) is incorporated into the objective function F(1) via an exact penalty approach:

$$Min\ z(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left[c_{ij}\, x_{ij} + \frac{1}{2} \sum_{k=1}^{n} \sum_{l=1}^{m} p_{ijkl}\, x_{ij}\, x_{kl} \right] \qquad \text{F(T)}$$

$$s.t. \quad \sum_{j=1}^{m} x_{ij} = 1 \qquad 1 \le i \le n$$

$$x_{ij} = 0 \ \text{or}\ 1 \qquad 1 \le i \le n,\ 1 \le j \le m$$

where

$$p_{ijkl} = \begin{cases} (n \cdot m+1) \cdot n+1 & \textit{if assigning } k \textit{ to } l \textit{ generates a conflict when } i \textit{ is assigned to } j \\ 0 & \textit{otherwise} \end{cases}$$

An extension version of the procedure proposed by Carlson and Nemhauser is used to solve the above problem: *[TT 82]*

(1) Determine a feasible solution for $F(T)$.

(2) Improve the objective function with single reassignments.

(3) Proceed to improve the objective function with double reassignments.

(4) If the solution is a local minimum, then terminate.

(5) Execute single reassignments inducing no improvement of the objective function. If the objective function can be improved with single or double reassignments, repeat Step 2 or Step 3, respectively. Otherwise terminate.

Carter described a quite similar formulation which centered on room scheduling. *[TT 14]* The room scheduling problem is formulated as a series of assignment problems with a set of side

constraints. A set of "Room Jumping Elimination" constraints are included in the formulation to force classes to stay in the same room in each period of the day:

Define $x_{ij}^{k} = 1$ if class i is assigned to room j in period k. Then the Room Jumping Elimination constraints can be defined as:

$$x_{ij_1}^{k_1} + x_{ij_2}^{k_2} \leq 1 \qquad i = 1, ..., n \text{ for all } j_1 = j_2 \text{ for all } k_1 = j_2$$

where n represents the number of classes in a subproblem. The mathematical formulation is the same as Ferland and Roy's, but special attention has been given to the design of the cost function c_{ij}. *Space Utilization, special facilities, preferred rooms, time fragmentation* and *non-standard timeslots* are taken into consideration while assigning the c_{ij} values.

2.3.3. Interactive Algorithms

White and Wong proposed that timetabling can also be classified by the degree of interaction of the user with the system. *[TT 88]* Besides those early papers on "bookkeeping" type of programs, most literature describes batch oriented systems. However, the nature of the timetabling problem suggests that the aspect of user interaction plays a vital role in the success of any system. Colijn suggested that the reasons for the generally disappointing results from most application systems probably include "*...the generally batch-oriented approach taken by these systems*". *[TT 19]*

There are a few research projects reported using the interactive approach. *[TT 19] [TT 26] [TT 33] [TT 34] [TT 88]* Generally an interactive system is composed of two phases, a preparatory phase, which is batch-oriented and creates a database, and an interactive phase which includes one or more interactive sessions during which the database created in the preparatory phase is modified by user(s) and the final timetable is produced.

In the algorithm reported by Colijn, the computer has two roles: keeping records (the database) and making suggestions for time and room assignment. *[TT 19]* The user decides either to use the computer's suggestions or to ignore or override them depending on environmental factors around the timetabling system. In the preparatory phase, several sets of information, which can either be constructed from pre-registration files or from historical data, are entered into computer and made available to the timetabling system before scheduling takes place. These include course and room data, the conflict matrix, etc. During the scheduling process, the computer makes heuristic suggestions and keeps track of assignments already made. The user first asks the computer to make suggestions for times which are conflict free, or at least low in terms of time-conflicts, for a given course, and for rooms that are available. Then he chooses one of the suggested (or other) times and rooms and

makes an assignment. The process can be repeated as often as necessary to complete a portion of or the whole timetable.

Special techniques have been used to keep the search process reasonably efficient and to protect the database. At the beginning of an interactive session, a small group of courses, the so-called reference group which typically consists of courses which have pairwise many students in common, or which must share some facilities, is identified. Searching in the interactive session is done in reference to these courses.

White and Wong's algorithm takes its data from two sources, a set of prepared files of stable information about student programs of studies, the staff available and the room inventory, and information entered interactively by users. *[TT 88]* The output is a set of files providing timetable of each course, teacher, and room. The complete timetable, which is conflict free, does not have to be constructed in one interactive session. Updated data is stored and read in automatically in the next session.

The algorithm attempts heuristically, via operator intervention, to construct a timetable which strictly adheres to the five constraints of the basic timetabling model T(R). Several other practical constraints which are solved by the system are,

(i) specially equipped rooms are scheduled only when the operator schedules them;
(ii) the grouping of courses is not handled but split or sectioned courses do not present problems because they are treated as separate courses. They can be scheduled simultaneously or otherwise as required;
(iii) there is no forced inhibition of certain teacher or room periods. Dummy periods or rooms may be created for this purpose;
(iv) any number of courses may be scheduled simultaneously provided suitable rooms are available and there are no student or teacher conflicts;
(v) pre-assignments are easily handled by placing the information into the pre-assingment file;
(vi) the oddities of various timetable timeslot layouts and spanning times are met by the flexible timeslot definitions;
(vii) changes in the timetable can be made with minimal disruption.

The algorithm begins by first scheduling the preassigned courses followed by scheduling professor-course pairs. When scheduling the professor-course pair, the user may make changes to the courses already scheduled if conflicts occur. The operator decides on the timeslots (classrooms) for courses by choosing from lists of available timeslots (classrooms) displayed in screen menu by the

computer. Unlike other algorithms, there is no formula to compute the conflict rating for each course as the aim of the algorithm is to construct a *conflict free* timetable.

2.4. TIMETABLING APPLICATIONS

It is noteworthy that one closely related area, *examination timetabling,* has been receiving considerable attention and there are many practical examination scheduling systems successfully running at various institutions. An extensive literature can be found on this subject in *[TT 15]*. *"The state-of-art for the course timetabling problem is primitive by comparison"*. *[TT 15]* There is no product available on the market that can solve the general timetabling problem satisfactorily. People are still trying to build their own systems.

While almost all of the North American universities and colleges are currently constructing timetables with some kind of help from computers, these scheduling systems involving the computer vary among each other greatly in both the nature of the questions they intend to solve and the methods or tools they use. However, they can be grouped into several groups according to the extent computers are used.

In most of the early systems using computers, the computer is purely used to aid the manual scheduling process. It may be used to store the scheduling data, facilitate the data modification, and even help to produce (print) the final products, but it never participates in the process of real scheduling, i.e., make decisions (like "this course will be assigned to that room on the certain time") or help making decisions by giving suggestions (such as "certain classrooms are free on that time"). A large number of institutions are still using this kind of systems.

Systems in which the computer plays a more important role may be grouped into another class. In this type of systems, the computer does not only mechanical labor such as storing and managing the data or printing the result, but also some intellectual work. It may give a list of classrooms that are free at certain time, suggest conflicts-free time periods, and/or check if certain assignment is following the scheduling standards, i.e. is conflict-free. The distinguishing feature of these systems is that it is the user who makes the decision of each assignment based on the program's suggestions.

One typical and also maybe one of the earliest systems is the Purdue Academic Scheduling System (PASS), which was developed at Purdue University in the early 1960's and is still used at Purdue. *[TT 15]* PASS reports courses which create significant number of conflicts by comparing

the initial timetable with the student course requests. The user adjusts the timetable based on this information and puts the new schedule into PASS. Then this process repeats until the user feels the resulting timetable meets the standards. Other early examples are the GASP system developed at M.I.T. and the Stanford School Scheduling System (SSSS), created at Stanford University. *[TT 57] [TT 60]* All these early systems "*tend to produce relatively poor solutions and require considerable manual time and effort.*" *[TT 15]*

One of the recently reported systems is the one being used at the Brigham Young University. *[TT 10] [TT 11]* The computer provides vital information that helps the academic scheduling officer in assessing needs, assigning classrooms, and making adjustments to the class timetables. The use of computer has simplified calculating, organizing, and displaying information, and even helped to create faculty awareness of scheduling difficulties, system capabilities, and the impact of curriculum change, but at the bottom it is still the human operator who makes every single scheduling assignment.

Most of the systems adopting the interactive approach also fall into this category. One example is CATS, the Computer Assisted Timetabling System, that was reported by Colijn in 1977. *[TT 19]* Its algorithm has been described early in 1.3.3 *Interactive Algorithms*.

The other type of systems are those in which the computer produces the final complete timetable with or without some user intervention. Different from the second type of system, the computer will make the decision of making individual assignments though the user may decide whether the resulting timetable is acceptable or not on the whole.

One early system of this type is the famous Ontario School Scheduling Program reported by Lions. *[TT 52]* The program was theoretically based on the model proposed and revised by Gotlieb and Csima with minor modification. *[TT 21] [TT 40]* It has been successfully used for constructing timetables for some Ontario high schools.

Most systems available on the commercial market fall into this category. *SCHEDULE25*, advertised by the Universal Algorithms Incorporated, is a room scheduling program that has been widely distributed in United States and Canada. *[TT 45] [TT 95]* Though there are no reports on its technical detail and algorithm structure, the company claims that "*SCHEDULE25 is a classroom scheduling software system containing a proprietary, unique, and highly sophisticated algorithm*" which "*running on a high speed computer, permits SCHEDULE25 to assign classes to classrooms so that a near optimal result is achieved.*" *[TT 94]* From the fact that it has gained

a market in more than 30 American major institutions such as Carnegie-Mellon, SUNY-Binghamton, and Virginia Tech, we can say that at least it has solved the room scheduling problem to a certain degree of satisfaction.

Another system new on the market is the Integrated Student Administrative System by ExCellere Associates, Inc. *[IT 29]* In this software, the scheduling functions have been spread into three separated modules, Class Schedule, Room Scheduling, and Student Scheduling. Though we can not find, as with most commercial products, any reliable data to support the claims made in the brochures, this software has attracted over a dozen clients such as University of Kentucky, Ball State University and University of Virginia.

Chapter 3 PROJECT MODEL

3.1. INTRODUCTION

The use of logic as a very high level programming language is an emerging approach to computer science in which computation is viewed as controlled deduction. Since PROLOG (PROgramming in LOGic), a theorem prover/program executor, was designed and implemented by Colmerauer and Roussel in 1972, logic programming has become a separate and rapidly developing research field. The large number of research activities has lead to many implementations of PROLOG and practical applications of logic programming in a wide range of areas, such as expert systems, artificial intelligence, natural language processing, image processing, digital circuits design and education.

The development of logic programming has also brought a new promising research topic in the field of computerized timetabling. Applying the results of logic programming research to the timetabling problem has produced a new area in which little literature has yet appeared. When we started to design CTS, the Computerized Timetabling System at the University of Ottawa, we could find virtually no publications on this subject.

We have built our system upon a logic programming model with a specific implementation of the PROLOG language, WPROLOG. There are two reasons for our decision to choose logic programming as the system specification and implementation tool. The first one is that from our survey of the literature (Chapter 2), we concluded that traditional approaches to the timetabling problem had been widely studied and the results were generally disappointing. The second reason is that logic programming systems have some very special characteristics which have been shown to be very useful

and powerful in solving the type of constraint satisfaction problem, into which timetabling problem can also be modeled. These characteristics will be summarized when we introduce the basic concepts of logic programming later in this chapter.

In this chapter, a basic introduction to logic programming and PROLOG as well as the model of CTS will be presented. In the first section, we will first briefly review the fundamentals of logic programming; then examine the features of PROLOG, the realization of logic programming, and finally we will discuss the available applications of logic programming in timetabling. The second section covers the logic model of our system, and will include timetabling constraints and their resolution in CTS.

3.2. LOGIC PROGRAMMING AND PROLOG

3.2.1. Fundamentals of Logic Programming

Logic programs consist of a set of axioms in Horn Clause form. Each axiom may be considered a procedure to be executed by an interpreter. Computation is the result of top-down inference using a proof procedure which is known to be sound and complete for Horn Clauses. The proof procedure may also be viewed as an interpreter that calls procedures represented by the clauses of the logic program. Logic programming systems have a built-in and fully general pattern matching facility based upon the unification principle. *[LP 16]* They are also equipped with an automatic backtracking facility for the exploration of alternative paths to a solution.

The Horn clause subset of logic was named after Alfred Horn, who first investigated its properties. More specifically, it is a subset of predicate logic. "Horn clauses are both exceedingly simple yet surprisingly expressive". *[LP 11]* They are the basic elements of PROLOG.

Pattern matching and *backtracking* are two important facilities of logic systems, in which they are used to find solutions for queries (proving goals).

A goal, or query, is proved either by matching it with a fact, i.e. assertion, or by matching it with the conclusion part of an implication (can also be viewed as an assertion) and then proving all the conditions of this implication. This matching is called pattern matching. It might be possible to match a goal with more than one assertion or implication. Thus logic systems must be able to try all possible combinations if necessary to prove the goal, or to find all possible ways it can be proved.

The most popular method used by logic systems in implementing this facility is *backtracking*.

3.2.2. The Logic Programming Language PROLOG

3.2.2.1. The features of PROLOG

Since its first implementation by Colmerauer in Marseilles, PROLOG has spread all over the world in a few years, bringing along with it the concept of logic programming. Now it is the most widely used logic programming language and is considered as the practical realization of logic programming.

PROLOG owes its success largely to its logic basis. As it is based on logic programming, it has inherited most of its features.

In logic programming, programs are logic statements about relations and the execution of a program corresponds to the proof of a particular conclusion. As a result, logic programs generally have logic interpretations. This brings the most attractive property of a logic programming language, its level of abstraction. It is possible to write declarative programs in PROLOG. Programmers can program in a more declarative style -- saying what a program should compute, rather than how to compute it. "The declarative style of a programming language encourages the programmer to think about the *intent* of a program, about the *static* description of relationships and properties that are to hold in a program, without having to worry about dynamic changes in the store of a computer. It deemphasizes the role of time in understanding programs and allows parts of a program to be examined and understood in isolation." *[LP 14]* Another advantage of programs in a declarative language is that they "are easier to modify because we can add more constraints without having to worry about the timing of checking those constraints". *[LP 14]* So with logic programming systems, one will be able to write both clear and efficient programs.

The basis of formal logic makes logic programming systems an obvious choice for writing programs that try to capture any kind of reasoning, because they enable one to "draw upon well understood and powerful ideas of formal logic in thinking about new ideas, for example inductive inference, planning, database design, natural language understanding, meta-level inference, or concurrent processing". *[LP 15]* PROLOG has found successful applications in these areas.

Logic programming systems have totally separated the implementation of inference procedures from writing applications. Since the inference procedure of logic programming has been shown to be sound and complete, if bad conclusions are produced, the axioms, not the inference procedure, should be changed. "This shifts the burden of writing intelligent computer programs from redesigning the inference procedure to better ways of structuring knowledge. Moreover, one can explore, independently of applications, techniques of implementing the inference procedure in progressively better ways." *[LP 15]*

Another advantage of logic programming systems originates from pattern matching. Pattern matching based on unification gives tremendous expressive power to logic programs which can be used to represent knowledge in a wide range of domains. "Moreover, pattern recognition is also one activity humans do very well. Hence it seems that a language based upon pattern recognition may have an excellent change of providing tools for modeling intelligent human performance." *[LP 15]*

PROLOG does not achieve all the features of an ideal logic programming language. In addition to Horn clauses, PROLOG provides "several extra-logical features which extend its power but conflict with the classical semantics of logic". *[LP 11]* These features are usually necessary to control the backtracking search strategy, (such as the *cut* predicate or utility), to provide clause manipulation during the program execution, and to create certain side-effects (such as write to the screen). As a result, not all PROLOG programs have a simple logical interpretation. But on the other hand, by means of these facilities, it is possible to define negation by failure and other extensions of Horn clauses in PROLOG.

3.2.2.2. PROLOG as a constraint programming language

The properties such as relational forms, non-deterministic computations and the unification mechanism of logic programming make it, as well as PROLOG, an appropriate and powerful tool for discrete combinatorial problems, specifically problems requiring constraint satisfaction.

In PROLOG, predicates are used to express relationships between their arguments and declarative statements about a problem are given in the form of Horn Clauses. Sussman and Steele *[LP 19]* have shown that constraint problems can be naturally expressed in terms of a set of declarative statements of relationships. Actually PROLOG has two ways of expressing constraints:

- *predicates* which enforce relationships between arguments;

. *unification* mechanism which creates and solves equality constraints.

When unifying two variables in PROLOG, the system does not wait until they are instantiated to test this equality constraint. Rather the constraint is "solved" by binding the variables to each other. Whenever one of them is given a value, this value is immediately propagated to the other variable. So unification in PROLOG can be considered as a constructive operation: it creates and solves equality constraints.

The two ways of expressing constraints and its non-deterministic computation which is controlled by a backtracking process make PROLOG a good candidate for a constraint programming language.

3.2.3. Applying Logic Programming in Timetabling

Application of logic programming in creating computerized timetabling systems is a new research topic. There are very few publications on the subject available. A closely related research area is the study of a class of problems named as the constraint satisfaction problems. Constraint satisfaction problems are widely studied in the field of artificial intelligence. *[LP 6] [LP 7] [LP 10]*

One of the reports on projects of computerized timetabling using the logic programming techniques was published in 1988 by Carmel and Itzovitz. *[TT 12]*

The project was called *Comprehensive University Planner (CUP)*, which was designed to support all administrative activities related to a university teaching system. Timetabling is divided into following functions: courses scheduling, rooms allocations and teaching assignments.

The implementation of CUP was done with the help of the so-called "the fifth generation software" *SAPIENS*, a software tool combining the power of an application generator and data base management system. Specifications of CUP were actually transformed through SAPIENS into a set of rules linked together by an inference engine. Any changes in the database will trigger some set of rules. So CUP is a rule-based system.

Recently Feldman and Golumbic reported a project of using constraint satisfiability algorithms for interactive student scheduling, a problem related to timetabling. *[TT 32]* The project was to determine which algorithms perform best in solving the student scheduling problem and under what conditions.

The authors noted that the order in which variables are assigned values significantly affects the performance of such algorithms and they investigated certain probabilistic techniques.

The student scheduling problem is the problem of searching for suitable schedules for students from the general university timetable. In the student scheduling problem (SSP) defined by the authors, two types of constraints are considered, which must be satisfied and which can be relaxed according to certain priorities. (We refer to them as *primary* and *secondary* constraints. See § 3.3) The procedures or algorithms of solving this problem are divided into four stages:

I) Preprocessing before starting the search;
II) Checks done during the search to choose the next instantiation;
III) Procedures done in each stage of the search process after a new instantiation is made;
IV) Checks done against the final schedules.

After testing three algorithms, regular backtracking, forward checking (sorting of the variables by increasing order of their domain sizes after each stage), and word-wise forward checking, for finding a schedule, the authors concluded that with large number of courses the word-wise algorithm was clearly better than the other two and regular backtracking is the worst while if the number of courses is smaller than 19, regular backtracking becomes the best. Also under each algorithm, three different instantiation orders were compared.

3.3. LOGIC MODEL OF CTS

In our system, the problem of timetabling is considered as one special case of Constraint Satisfaction Problems(CSP), which usually involves finding an assignment of a set of values to a set of variables that satisfies a set of constraints. The class of CSPs also includes many problems like graph coloring, boolean satisfiability, scene and edge labeling and logic puzzles among others as particular cases.

The formal definition of timetabling as a constraint satisfaction problem will be presented first, following by detailed descriptions of all the primary and secondary constraints as well as their resolution precedence and expression in first order logic.

3.3.1. Timetabling as a Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) can be formally defined as following: *[LP 10]*
For a finite set of variables, $V = X_1, X_2, ..., X_n$, which take their values respectively from their corresponding domains $D_1, D_2, ..., D_n$, and a finite set of constraints of the form $c(X_{i_1}, ..., X_{i_k})$ among k variables from V representing a subset of the cartesian product $D_{i_1} \times \cdots \times D_{i_k}$ which specifies which values of the variables are compatible with each other, a solution to a CSP is an assignment of values to all variables which satisfy all the constraints. [CSP]

In designing the Computerized Timetabling System at the University of Ottawa, we have formulated the problem of timetabling as a CSP. Our model is an extension of the [CSP]: the set of constraints in the original model is called first order or secondary constraints and another set of constraints, secondary constraints, is added to express those constraints in timetabling problem that should be satisfied as much as possible but could be neglected if necessary.

3.3.2. Variables of CTS Model

There are five variables taken into account in CTS, *course, time, room, professor,* and *program.* Each of these variables normally has several arguments representing its properties, which are used in *constraints* to define the comparability among the values of different variables.

course: represents the course variable. Its domain contains all the courses as well as labs and discussion groups to be scheduled in the system. A course is defined by several properties: student enrollment, type, format, teaching professor, and etc.

time: or time period, represents the time variable in the timetabling system. Its domain, T, is the set of all the time blocks permitted in the time slot system. Currently at the University of Ottawa, most of the courses fall into one major slot system while many other formats are also allowed. In designing CTS, with the agreement of the Registrar's Office at the University, we have adapted one standard slot system in which course lengths are limited to times of one-and-half hour. Our timetable is a weekly timetable. One week is divided into six working days (some courses have to be taught on Saturday) and each day (except Saturday with one block) is divided into 9 one-and-half hour blocks. So the domain of the *time* variable in our model, T, has a set of 46 values. [See Appendix B.2]

room: represents the room variable in the system. Its domain, R, contains all the classrooms available at the University for scheduling purposes. The arguments of a *room* value include room type, furniture type, seating type, room size as well as a list of reserved time blocks. [See Appendix D.1]

professor: represents teachers for courses. Its domain, P, is the set of all professors involved in teaching *course*. A professor may have a list of *time* values which state the time he cannot teach. [See Appendix D.2]

student program: represents student academic programs. This variable has two properties, required courses and recommended courses. The domain of this variable, S, is the set of all student programs in all the departments and faculties at the University. Generally a group of students of the same year and studying the same major area at one department will be considered as being in the same academic program. [See Appendix D.3]

3.3.3. Constraints of CTS Model

A solution to a timetabling problem involves the resolution of many constraints. CTS encounters four classes of constraints:

- Administrative constraints which are limitations pertaining to the University's policy and procedure;
- Environmental constraints which are limitations pertaining to the University's physical resources(classrooms and their equipment) and how they are related to the courses taught;
- Preferential constraints which are limitations pertaining to the needs, wants and wishes of the students and professors;
- Systemic constraints which are limitations pertaining to computer capacity and the relevant software.

These constraints can also be classified according to their functions in the CTS model as first order constraints which *must* be satisfied and secondary constraints which *should* be satisfied.

3.3.3.1. First Order Constraints of CTS Model

First order constraints of a timetabling system are those constraints that must be satisfied by the final timetable.

0.01 *No two or more required courses of any year of a student program can be scheduled such that they overlap in time.* In another words, no student should be required to show up at two places simultaneously. This restraint guarantees that students are able to take their required courses.

0.02 *No recommended course of a student program can be scheduled in such a way that it will conflict in time with any one of the required courses of the same student program.*

0.03 *No two or more courses of a professor are scheduled such that they overlap in time.* That is, no professor should be required to teach two different classes simultaneously.

0.04 *Daytime courses must be scheduled into daytime periods and evening courses must be taught in the evening time.* Courses are divided into daytime courses and evening courses. Most evening courses are for part-time students who are not free during daytime, and often, the time that a course is offered during day or night is a deciding factor in the course selection process of the students.

0.05 *The classrooms assigned to courses must have capacities equal or larger than the enrollments of the courses scheduled into them.*

0.06 *Classrooms must not be scheduled during their reserved time.* Some classrooms are reserved for certain time for other academic or non-academic use and during these time periods they cannot be assigned to any courses.

0.07 *Courses cannot be scheduled into their professors' unavailable time.* Each professor has certain time blocks when he may be occupied by other activities. In addition, each department may set apart one time period each week for staff meetings. The system should take these factors into consideration and not schedule courses in the time when professors have other duties.

0.08 *Preassigned courses must be scheduled into their preassigned classrooms at the preassigned time.* There are three cases of preassignment. The first case is that a course's teaching time and classrooms are all fixed before the scheduling. These courses are called *completely-preassigned*

courses. In the second case, only the teaching time of a course is preassigned, the classroom is to be decided by the system. We call it a *time-preassigned* course. In contrast, some courses have to be held in certain rooms but the time can be scheduled by the system. In this case, we have a set of *room-preassigned* courses.

0.09 *Courses that should be taught in the same classroom at the same time can be so scheduled.* Certain courses have different course codes but they are essentially the same course. These courses are required to be scheduled in the same classroom at the same time.

0.10 *Courses that should overlap in teaching time must be so scheduled.* Some courses may be required to be scheduled in such a way that their teaching times overlap (not necessary completely) with each other to enforce a prerequisite or for some other reason.

0.11 *Courses that should have exactly the same teaching time but different classrooms can be so scheduled.* A group of courses are sometimes required to have the same teaching time but classrooms may not be the same. Many second language courses at the University of Ottawa are of this type. Normally this situation happens when students have to take one course but they do not know which section (sometimes denotes the level) of the class they will attend until some time after the class starts.

0.12 *If a course is held more than once a week, there will be always at least one day between any two lectures of the course.*

3.3.3.2. Second Order Constraints of CTS Model

Secondary, or second order, constraints are generally those constraints which should be followed but sometimes are impossible or impractical. For example, we would like to have all the recommended courses of student programs be scheduled into different time blocks so that students of this program can choose any recommended courses they want to take. But this is frequently impossible due to the limited amount of room and time resources.

Some of the secondary constraints are imposed for improving the two efficiency indices, time utilization and seating usage of classrooms.

1.01 *Courses should be scheduled into the time zone specified by the department or professor. (timezone rule)* The time blocks in day time are further divided into 3 time zones, morning, noon and afternoon. While night courses have to be scheduled into night time blocks, a day course should be scheduled into the proper zone as specified. As the specification of time zone is optional, a course will be assigned to any time zone if no specific requirement is indicated.

1.02 *Courses should be scheduled into classrooms in the building as close to the host department or the building specified by the department or professor as possible. (geographic rule)* For every course, the host department (the department offering courses is called these courses' host department) or the professor can indicate a preferred building. The system will try to schedule the course into one of the classrooms of the building if possible. If it fails to do so, the system will find a classroom as close to this building as possible while satisfying all first order constraints and all those secondary constraints that have higher priority.

1.03 *Courses should be scheduled into rooms of the type specified by the department or professor. (room type rule)* The *type* property of a room is defined by three arguments, class type, furniture type and seating type. Any number of the three arguments can be specified. The system will first try to find rooms satisfying all the three arguments. If it cannot be found, rooms which satisfy room type and furniture type will be searched. If this kind of room still cannot found, rooms only satisfying the room type is searched. Finally if all the above tries failed, the system will abandon this restriction. Indeed, this constraint contains three subrules, class type rule, furniture type rule, and seating type rule.

1.04 *Courses should be scheduled into classrooms in such a way that a certain level of seating usage is kept. (seating usage rule)* This constraint requires the system to assign courses into classrooms of as small a size as possible as long as they are large enough to contain the number of students registered in these courses.

1.05 *Recommended courses of a student program should be scheduled in such a way that there are the fewest time conflicts among them. (recommended course rule)*

1.06 *Friday afternoon shall be used only after other scheduling possibilities are exhausted. (Friday afternoon rule)*

3.3.3.3. The Precedence of Secondary Constraints in CTS Model

Most times, it is impossible to satisfy all the secondary constraints, especially in large institutions where the number of constraints is huge. When this situation happens, some of them have to be relaxed. In our project, each secondary constraint has been given a precedence level denoting its priority of being kept in case of relaxation. The higher the priority, the later it is going to be relaxed, or disabled.

The process of assigning a suitable teaching time and classroom to a course is called one *scheduling cycle*. It is obvious that relaxing one constraint at one time is not an efficient way of programming as the running time could be greatly prolonged if more than one constraint is going to be relaxed for a large number of scheduling cycles. So secondary constraints in our system are relaxed in groups which are grouped according to the constraint's priority levels. Constraints of the same priority are disabled at the same time.

Level 1

1.02 The geographic rule;

Level 2

1.03 The seating type rule (Room type rule);
1.03 The class type rule (Room type rule);

Level 3

1.03 The furniture type rule (Room type rule);

Level 4

1.04 The seating usage rule;
1.05 The recommended course rule;

Level 5

1.01 The time zone rule;
1.06 The Friday afternoon rule;

3.3.4. The Expressions of Constraints in an Extended Horn Clause Form

3.3.4.1. Extension to Horn Clause Form

We will transform all the first order and secondary constraints into expressions of an extended Horn Clause form. We could have just used Horn Clauses and they do have all the expressive power needed in our case. But in order to avoid explicit loops and recursions, we have chosen an extended form instead. In addition to all the features of Horn Clauses, universal quantifiérs (**for all** and **for some**) and conditions of implications are allowed.

For example, the definition of a clause (relationship) of no_common_member(L_1, L_2) which expresses that lists L_1 and L_2 do not have common members will be:

no_common_member(nil, L_2).
no_common_member(list(a, L), L_2) if
    not(is_a_member(a, L_2)) and
    no_common_member(L, L_2).

in the Horn Clause form. (*nil* stands for an empty list) If written in our new form, the same predicate will looks as:

no_common_member(L_1, L_2) **if**
    for all A
        (not(is_a_member(A, L_2)) if
            is_a_member(A, L_1)).

Note that the first expression involves an explicit recursion while the second one does not. Furthermore, in the second expression, we used an universal quantifier "**for all A**" and an implication as the condition.

3.3.4.2. Assumptions and Predicates

For all the expressions in the following sections, we have assumed:

- There is a set of n courses, $C=c_{1,...,}\ c_n$, a set of q professors, P, a set of l classrooms, R, a set of m student programs, S, of which the ith member has the form $sp_i(s_1,s_2)$ where s_1 and s_2 represent respectively the list of required courses and the list of recommended courses of the ith program.

- For each course c_i, there is a set of n_i^t time periods, T_i, representing its teaching time.
- For each professor P_i, there is a set of n_i^p courses, Q_i, of which P_i is the teacher.
- For every classroom R_i, there is a set of course-time pairs of which each member has the form $r(c_j, t_k)$, where c_j is the course taught in R_i at time t_k.

We may just start directly to convert our constraints into expressions of the new form now, but with the help of a few properly named predicates the expressions can be more readable and closer to plain English. For example, we can define the no-conflict constraint of required courses as:

for all i, j_1, j_2, k_1, k_2
($t'_{k_1} \neq t''_{k_2}$ **if**
($sp_i(s_1,s_2) \in S$ **and**
$c_{j_1} \in s_1$ **and**
$c_{j_2} \in s_1$ **and**
$j_1 \neq j_2$ **and**
$t'_{k_1} \in T_{j_1}$ **and**
$t''_{k_2} \in T_{j_2}$))

where

$i = 1, 2, ..., m$; $j_1 = 1, 2, ..., n$; $j_2 = 1, 1, ..., n$;
$k_1 = 1, 2, ..., n_{j_1}$; $k_2 = 1, 2, ..., n_{j_2}$

We can see this is a first order logic expression. By using the two predicates we have defined earlier as examples, however, the expression could be written in a much more declarative style:

constraint001 **if**
for all i, j_1, j_2
(no_common_member(T_{j_1}, T_{j_2}) **if**
(is_a_member($sp_i(s_1,s_2)$, S) **and**
is_a_member(c_{j_1}, s_1) **and**
is_a_member(c_{j_2}, s_1) **and**
$c_{j_1} \neq c_{j_2}$))

where

$i = 1, 2, ..., m$; $j_1 = 1, 2, ..., n$; $j_2 = 1, 2, ..., n$

Besides the two predicates, is_a_member and no_common_member, whose definitions have been given in previous sections, we will use a few more simple predicates in the expressions in the following two subsections. They are described as follows:

same_set(L_1, L_2): it evaluates the relationship of equal sets, i.e. the members of list L_1 and list L_2 are identical.

same_set(L_1, L_2) **if**
for all A and B
((is_a_member(A, L_1) **if**
is_a_member(A, L_2)) **and**
(is_a_member(B, L_2) **if**
is_a_member(B, L_1)))

room_size(R, S): this predicate gets the size of room R and puts the value into S;

room_type(R, T): T is the type value of room R. T is in the form of t(RT, FT, ST) where RT, FT, and ST denote room type, furniture type, and seating type respectively;

enrollment(C, E): it gets the enrollment of C and puts into E;

teaching_time(C, T): it collects all the teaching time blocks of course C into the set T;

teaching_room(C, R): it puts all the classrooms used by the C into the set R;

teaching_roomtype(C, T): T is the room type required by C;

teaching_time_room(C, TR): C is a course and TR contains all the teaching time_room pairs of this course. Each member of TR has the format of $tr_i(T,R)$ where i is the sequential number of the element in set TR, T is one teaching time of C and R is the classroom.

teacher_courses(P, C): C is the set of courses taught by professor P;

unavailable_time(P, T): T is the set of all time blocks when the professor P could not teach; thus courses taught by P cannot be scheduled in any time period in T;

reserved_time(R, T): T is the set of all reserved time blocks of room R.

3.3.4.3. First Order constraints

0.01 The no-conflict constraint for required courses

constraint001 **if**
for all i, j_1, j_2
(no_common_member(T_{j_1}, T_{j_2}) **if**
(is_a_member($sp_i(s_1,s_2)$, S) **and**
is_a-member(c_{j_1}, s_1) **and**
is_a_member(c_{j_2}, s_1) **and**
$c_{j_1} \neq c_{j_2}$ **and**
teaching_time(c_{j_1}, T_{j_1}) **and**
teaching_time(c_{j_2}, T_{j_2})))

where

$i = 1, 2, ..., m;\ j_1 = 1, 2, ..., n;\ j_2 = 1, 2, ..., n$

0.02 The no-conflict constraint for recommended courses

constraint002 **if**

for all i, j_1, j_2

(no_common_member(T_{j_1}, T_{j_2}) **if**

(is_a_member($sp_i(s_1,s_2)$, S) **and**

is_a-member(c_{j_1}, s_1) **and**

is_a_member(c_{j_2}, s_2) **and**

teaching_time(c_{j_1}, T_{j_1}) **and**

teaching_time(c_{j_2}, T_{j_2})))

where

$i = 1, 2, ..., m;\ j_1 = 1, 2, ..., n;\ j_2 = 1, 2, ..., n;\ j_1 \neq j_2$

0.03 The no conflict constraint for professor

constraint003 **if**

for all i, j_1, j_2

(no_common_member(T_{j_1}, T_{j_2}) **if**

(is_a_member(p_i, P) **and**

teacher_courses(p_i, Q_i) **and**

is_a_member(c_{j_1}, Q_i) **and**

is_a_member(c_{j_2}, Q_i) **and**

$c_{j_1} \neq c_{j_2}$

teaching_time(c_{j_1}, T_{j_1}) **and**

teaching_time(c_{j_2}, T_{j_2})))

where

$i = 1, 2, ..., q;\ j_1 = 1, 2, ..., n;\ j_2 = 1, 2, ..., n;$

0.04 The day-evening separation constraint

constraint004 **if**

for all j, k

((is_a_member(t_k, T^d) **if**

(is_a_member(c_j, C^d) **and**

teaching_time(c_j, T_j) **and**

is_a_member(t_k, T_j)) **or**

(is_a_member(t_k, T^e) **if**

(is_a_member(c_j, C^e) **and**

teaching_time(c_j, T_j) **and**

is_a_member(t_k, T_j)))

where C^d and C^e are subsets of C and contain respectively all of the n^d daytime courses and all of the n^e night courses; T^d and T^e are subsets of T and contain all of the day p^d time blocks and all of the p^e night time blocks respectively; and

$j = 1, 2, ..., n^d$; $k = 1, 2, ..., p^d$;

or

$j = 1, 2, ..., n^e$; $k = 1, 2, ..., p^e$

0.05 The classroom size constraint

constraint005 **if**

for all i, j

(*Rsize* ≥ *Size* **if**

(is_a_member(c_j, C) **and**

enrollment(c_j, *Size*) **and**

teaching_room(c_j, R^j) **and**

is_a_member(r_i, R^j) **and**

room_size(r_i, *Rsize*))

where

$i = 1, 2, ..., m$; $j = 1, 2, ..., n$

0.06 The classroom reservation constraint

constraint006 **if**

for all i, j

(not(is_a_member(t, T_i^j)) **if**

(is_a_member(c_j, C) **and**

teaching_time_room(c_j, TR_j) **and**

is_a_member(tr_i(t , r), TR_j) **and**

reserved_time(r, T_i^j)))

where

$i = 1, 2, ..., m$; $j = 1, 2, ..., n$

0.07 The professor time constraint

constraint007 **if**

for all i, j

(no_common_member(T_j, T_i^p) **if**

(is_a_member(p_i, P) **and**

unavailable_time(p_i, T_i^p) **and**

teacher_courses(p_i, Q_i) **and**

is_a_member(c_j, Q_i) **and**

teaching_time(c_j, T_j)))

where

$i = 1, 2, ..., m$; $j = 1, 2, ..., n$

0.08 The preassignment constraint

constraint008 **if**

for all k, j

(is_a_member(tr_k(t , r), TR_j) **if**

(is_a_member(c_j, Z) **and**

reserved_value(c_j, Z_j) **and**

teaching_time_room(c_j, TR_j) **and**

is_a_member(tr_k(t , r), Z_j)))

where Z is the set of all courses that have preassigned values, Z_j is the set of n_j^z preassigned values of course c_j, and

$j = 1, 2, ..., n$; $k = 1, 2, ..., n_j^z$

0.09 The identical courses constraint

constraint009 **if**

for all k

(same_set(TR_{j_1}, TR_{j_2}) **if**

(is_a_member($pair_k$(c_1 , c_2), IP) **and**

teaching_time_room(c_1, TR_{j_1}) **and**

teaching_time-room(c_2, TR_{j_2})))

where IP is the set of all n^{IP} pairs of courses that should be given the same teaching time and classrooms, and

$k = 1, 2, ..., n^{IP}$; $j_1, j_2 = 1, 2, ..., m$

0.10 The time overlap constraint

constraint010 **if**

for all k, some i

(is_a_member(t_i, T_{j_2}) **if**

(is_a_member($pair_k$(c_1, c_2), PT) **and**

teaching_time(c_1, T_{j_1}) **and**

is_a_member(t_i, T_{j_1}) **and**

teaching_time(c_2, T_{j_2})))

where PT contains all n^{PT} pairs of courses that should overlap with each other in time and $k = 1, 2, ..., n^{PT}$; $j_1, j_2 = 1, 2, ..., m$; $i = 1, 2, ..., 36$

0.11 The same room constraint

constraint011 **if**

for all k, k_1, k_2

($r_1 \neq r_2$ **if**

(is_a_member($pair_k$(c_1, c_2), PTR) **and**

teaching_time_room(c_1, TR_{j_1}) **and**

teaching_time_room(c_2, TR_{j_2}) **and**

is_a_member($tr_{k_1}(t, r_1)$, TR_{j_1}) **and**

is_a_member($tr_{k_2}(t, r_2)$, TR_{j_2})))

where PTR is the set of n^{PTR} pairs of courses whose teaching time should be same but classrooms should be different, and

$k = 1, 2, ..., n^{PTR}$; $j_1, j_2 = 1, 2, ..., m$; $k_1, k_2 = 1, 2, ..., n_j^{TR}$

0.12 The one-day intermission constraint

constraint012 **if**

for all j

(($t_1 - t_2 > 1$ **or** $t_2 - t_1 > 1$) **if**

(is_a_member(c_j, C) **and**

teaching_time(c_j, T_j) **and**

is_a_member(t_1, T_j) **and**

is_a_member(t_2, T_j) **and**

$t_1 \neq t_2$))

where

$j = 1, 2, ..., m$; $t_1 = 1, 2, ..., 36$; $t_2 = 1, 2, ..., 36$

3.3.4.4. Secondary constraints

1.01 The timezone rule

constraint101 **if**

for all i, j

(is_a_member(t_i, *Tzone*) **and**

(is_a_member(c_j, *C*) **and**

course_timezone(c_j, *zone*) **and**

zone_time(*zone*, *Tzone*) **and**

teaching_time(c_j, T_j) **and**

is_a_member(t_i, T_j)))

where

$i = 1, 2, ..., 36;\ j = 1, 2, ..., n$

1.02 The geographic rule

constraint102 **if**

for all i, j

(is_a_member(r_i, R'') **if**

(is_a_member(c_j, *C*) **and**

teaching_room(c_j, R') **and**

prefer_building(c_j, *b*) **and**

building_rooms(*b*, R'') **and**

is_a_member(r_i, R')))

where

$j = 1, 2, ..., n;\ i = 1, 2, ..., l$

1.03 The room type rule

constraint103 **if**

for all i, j

($type_1$ = $type_2$ **if**

(is_a_member(c_j, *C*) **and**

teaching_roomtype(c_j, $type_1$) **and**

teaching_room(c_j, *R*) **and**

is_a_member(r_i, *R*) **and**

roomtype(r_i, $type_2$)))

where

$j = 1, 2, ..., n$; $i = 1, 2, ..., l$

1.04 The seating usage rule

cosntraint104 **if**

for all i, j

(*per* $\geq$ *Rate* **if**

(is_a_member(c_j, C) **and**

enrollment(c_j, *Size*) **and**

teaching_room(c_j, R^i) **and**

is_a_member(r_i, R^i) **and**

room_size(r_i, *Rsize*) **and**

percentage(*Size*, *Rsize*, *per*)))

where *Rate* represents the least seating usage rate admissible and

$j = 1, 2, ..., n$; $i = 1, 2, ..., l$

1.05 The recommended course rule

constraint105 **if**

for all i, j_1, j_2

(no_common_member(T_{j_1}, T_{j_2}) **if**

(is_a_member($sp_i(s_1,s_2)$, S) **and**

is_a-member(c_{j_1}, s_2) **and**

is_a_member(c_{j_2}, s_2) **and**

$c_{j_1} \neq c_{j_2}$ **and**

teaching_time(c_{j_1}, T_{j_1}) **and**

teaching_time(c_{j_2}, T_{j_2})))

where

$i = 1, 2, ..., m$; $j_1 = 1, 2, ..., n$; $j_2 = 1, 2, ..., n$

1.06 The Friday afternoon rule

constraint106 **if**

for all j, k

($t_k \leq 54$ **and** $t_k > 59$ **if**

(is_a_member(c_j, C) **and**

teaching_time(c_j, T_j) **and**

is_a_member(t_k, T_j)))

where

$j = 1, 2, ..., m$; $k = 1, 2, ...,36$

[See § 4.2.1 for the time slot system]

Chapter 4 THE IMPLEMENTATION OF CTS

In this chapter, we will discuss the implementation issues of the Computerized Timetabling System at the University of Ottawa.

4.1. TECHNICAL ENVIRONMENT OF CTS

4.1.1. The Computer System

CTS was developed on the University's mainframe computer Amdahl VM/370 which is running under the IBM VM/CMS operating system. The mainframe system is the major computing resource available to the university's academic community. The fact that all administrative software being used by the university including that used by the Scheduling Office is running on this system leads naturally to our selection.

It is obvious that the amount of memory required to solve the timetabling problem at a university level is very large. Also we need to choose a system which runs fast enough to give results in a reasonable length of time. This has eliminated the possibility of using contemporary micro-computer systems. The mainframe system which is the only large computer system within easy reach of our system's designated users became our only choice.

CTS, at current stage, is basically a batch-processing system, and each run will take no more than a few hours of CPU time on the mainframe system. Though the University may need several runs for producing one timetable, these runs can be done during weekends or at other times when

usage of the computer is not heavy, thus the speed and efficiency of normal academic computing usage on the mainframe system will not be affected by the integration of CTS into the current timetabling process.

4.1.2. The Existing Software

There are several programs used in aid of timetabling at the University of Ottawa. Their functions include maintaining the main database, facilitating time scheduling and room scheduling, producing the final timetables as well as doing statistical analysis of timetables.

On the academic research aspect, the core of the CTS system is completely independent from this existing software so that it may be easily transformed onto other computer systems. However for the practical purpose of the effective implementation at the University of Ottawa, we have written several supporting programs which make the input and output data compatible with the other software.

The main database file of timetabling as well as other data files are maintained by using the standard CMS editor XEDIT. The main database file, *HORAS FILE*, serves both as a file containing the most current information on the university timetable, and as a starting file for timetabling at the beginning of new academic terms (The new timetable is created by making changes to that of last year). All programs working for the purpose of timetabling are linked together around this database file.

The integration of CTS into the current timetabling practice will not affect the running of the existing software because CTS uses *HORAS FILE* both as the source of input data and as the destination of its output. The supporting utility programs of CTS link CTS's input file to *HORAS FILE* for the most up-to-date course and room reservation information. They also record the output data from CTS's scheduling program into the same *HORAS FILE*. So the existing software for timetabling will still do what it does now.

Currently the most frequently used system is *TTAB*. *TTAB* is a classroom reservation system which was designed to provide scheduling information to the faculties and departments and allow them to find both available and suitable classrooms for their professors, by giving them controlled (and read-only) access to the Scheduling Office's files (the main database). *[TT 78]*

TTAB is a menu-driven system. It provides the following functions:

- Reservation requests and Changing requests; Users at faculty or department level can produce classroom reservation requests or make changes to the requests produced before. Requests and changes will be electronically transmitted to the scheduling office for verification and approval;
- Add courses to timetable; This function allows users to add new courses or course sections to the timetable;
- Free rooms for timeslot; This option is used to find all classrooms on campus that are free for the specified session and time-slot and display their characteristics on the screen;
- Display classroom specs/Display classroom schedule; Users can use these functions to display physical information and time schedules about any classroom on campus;
- Find free classrooms; This function can be used to find all available classrooms in a specified building that are free for the indicated term, start time and duration;
- Scanfile search; Users may use this function to do different searches among the main database file.

TTAB is mainly used for room scheduling at the University while at times it serves as a medium of communication between the scheduling office and the faculties and departments.

There is another program that is being frequently used by the scheduling office at the University. It is mainly used for timetable production and statistics collection. *[TT 77]*

The functions of the systems can be grouped into following classes:

- Timetable production; This is the function used by the scheduling office to produce different versions of university timetables. Scheduling data in the main database is transformed into special forms which can be interpreted and printed on the IMAGEN laser-printer at the University;
- Timetable changes listing; The function extracts changed assignments from the main database file, converts timeslot codes into real day-times, and produces a listing file which contains all changes since the printing of the most recent edition of the timetable;
- Statistics reports; There are several functions available in the system to produce statistics reports on the current timetable. The statistics include classroom seating usage, classroom time utilization, course section distribution, course subject distribution, etc.

4.1.3. Waterloo PROLOG

The main programs of CTS are implemented using Waterloo PROLOG (WPROLOG). There are, however, several utility programs written in C, mainly for efficiency.

WPROLOG is an interpreter for the programming language PROLOG developed at the University of Waterloo, Ontario, Canada. It is available on the CMS system at the University of Ottawa which has licenced the product since 1986. We decided to use WPROLOG not only because it was, and still is, the only PROLOG implementation available on the mainframe system at the University of Ottawa at the time we started the CTS project, but also due to many pleasing characteristics of this implementation.

WPROLOG is a simple but complete version of PROLOG. It "features a program development environment that allows rapid program prototyping as well as the structured, modular development of large systems. Programs are entered by using the standard CMS editor XEDIT, which is integrated with WPROLOG. Programs are ready to run as soon as they are loaded; no other processing steps are necessary. The Workspace of WPROLOG is composed of any number of modules which can be individually changed and re-loaded. Modules are supported in a way that preserves the fundamental syntax and nature of the language. These features, along with the extensive error checking, the built-in trace facility and the comprehensive library of routines, make WPROLOG easy to learn and convenient to use". *[LP 20]* From our experience, anyone who knows PROLOG can easily become familiar with WPROLOG within a few days.

With a few exceptions, we are satisfied with all aspects of WPROLOG, especially the simplicity in organizing modules. One of the weak points is its inability to interface with programs written in other languages (C in particular). The speed of WPROLOG programs for the purpose of text format tansformation is also disappointing, being much slower than of those written in C.

4.2. SYSTEM SPECIFICATION

4.2.1. Timeslot system

The University of Ottawa currently has a multi-length slot system in which different lengths of time slots are permitted. Included in this system are two major slot-systems, one for $2\times1\frac{1}{2}$ hour courses (i.e. twice weekly for $1\frac{1}{2}$ hours) and the other for 3×1 hour courses (i.e. three times weekly for 1 hour). [See Appendix B] Besides the two course formats, $2\times1\frac{1}{2}h$ and $3\times1h$, many other formats exist. This system has several disadvantages:

- Any course which does not follow any of the two slot systems will leave one or more slots empty but unusable by other courses;
- If two courses following two slot systems respectively are assigned to the same classroom, some of the room's scheduling capacity will be lost as there are holes between these two systems;
- A large proportion of the courses offered by the University does not follow the slot systems, approximately 50 percent in the Fall term of 1985;
- Having two slot systems as well as allowing other formats increases the difficulty of scheduling for scheduling officers both at the University level and the faculty and department level. *[IT 42]*

The University has been striving towards one standard slot system in which lengths or durations of lectures and labs are limited to be one or more times $1\frac{1}{2}$ hours, a time block. In recent years, the University is gradually eliminating the 3×1 hour format courses. The Registrar's Office has also decided to regulate other odd formats which are not one or more times $1\frac{1}{2}$ hour time blocks. These courses either have to be preassigned or transferred into formats compatible with the standard slot system.

CTS has adapted the new standard-slot system [Figure 1]. In the new system, the smallest time unit is one and a half hours. The duration of every lecture is also limited to times of the $1\frac{1}{2}h$ block. There are four course formats which are considered as basic formats, $1\times1\frac{1}{2}h$, $1\times3h$, $2\times1\frac{1}{2}h$, and $1\times2h$. Courses in the format of $1\times2h$ are permitted on the basis that they will always start at 8:00am, so as not to disrupt the block sequence. So these courses could be scheduled as $1\times1\frac{1}{2}h$ courses with the

restriction that they can only occupy the first time block of each day.

The New Standard Slot System								
			Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Day Time	Morning	08:30-10:00	11	21	31	41	51	61
		10:00-11:30	12	22	32	42	52	..
	Noon	11:30-13:00	13	23	33	43	53	..
		13:00-14:30	14	24	34	44	54	..
	Afternoon	14:30-16:00	15	25	35	45	*55*	..
		16:00-17:30	16	26	36	46	*56*	..
		17:30-19:00	17	27	37	47	*57*	..
Night Time	Evening	19:00-20:30	18	28	38	48	*58*	..
		20:30-22:00	19	29	39	49	*59*	..

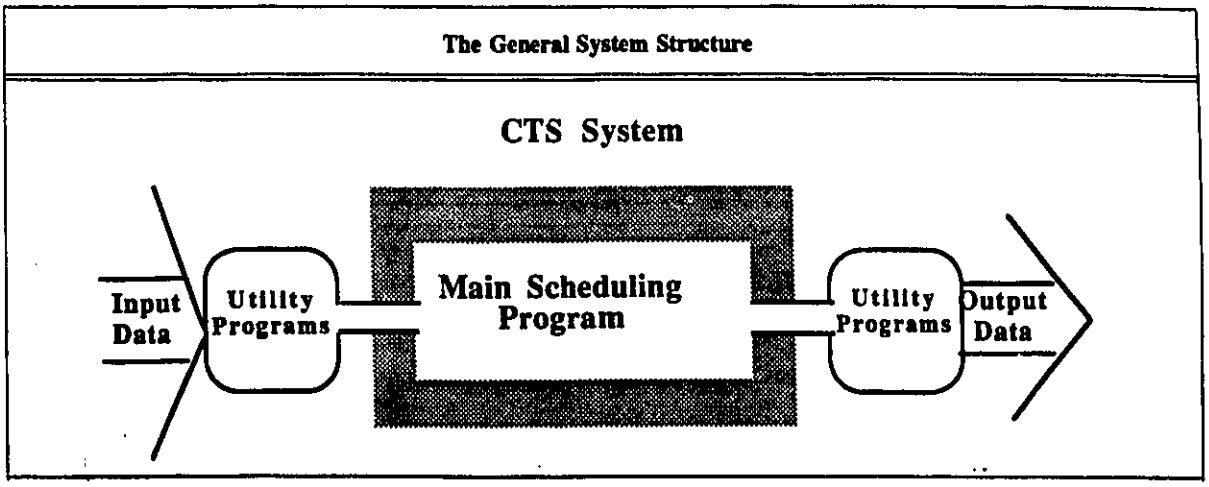
[Figure 1]

Time blocks in Friday afternoon and evening (codes in italics) are not used until other possibilities are exhausted. Courses can be scheduled on Saturday only when they are so specified.

Courses which do not have one of the basic formats are not scheduled by the system. They will have to be preassigned. Due to the fact that the proportion of this type of courses is not large (approximately 5 percent at the University of Ottawa) and most of these courses are very specialized such as special laboratories of Arts departments requiring specific classrooms, preassigning them will not pose a serious problem and, in fact, is the current practice.

4.2.2. General System Structure

A group of supporting utility programs, the main scheduling program, and input and output data constitute the CTS system. [Figure 2] (See Appendix A for the detailed structure of the system)



[Figure 2]

The utility programs' duties include

- accept and process course information directly from various sources outside the system; transform the information into a set of data files usable by the main program; and keep the new data files updated;
- transform the output data produced by the main program into forms readable by other programs.

The main program takes its input data from data files produced by the pre-processing programs along with a set of *constant* data files (Room inventory etc.), schedules all the cou.. into suitable classrooms and time blocks, and produces an output file containing these assignments.

4.2.3. The Scheduling Program

4.2.3.1. Input Data

There are two types of input information required by the CTS scheduling program. One type of data which is relatively constant over different terms, such as the room inventory data and time-slot data, is called *constant data*. The other type of data changes along with the time. The data about courses that are going to be scheduled is a representative example. Data of this type are called *variable data*.

There are three files containing the constant data information needed by the scheduling algorithm:

- **Time-Slot File** contains all the legal time blocks in the standard slot system. Each time block is identified by a unique two-digit code. There are totally 46 blocks, of which 45 are for weekdays and one for Saturday.

- **Room-Inventory File** maintains information on all the classrooms available for teaching purposes at the University. Classrooms at the University of Ottawa are divided into two types: rooms controlled by the Scheduling Office and those controlled by different faculties or departments. Each room is described by *size, room type, furniture type, seating type, accessibility to handicapped people*.

- **Geographic File** defines the building preference of each faculty or department. For each course, the host department can identify a building whose rooms are most wanted for this course. If the scheduling program tries all the possibilities and cannot find a suitable, free room for the course, it needs to know what order it should follow to try other buildings, i.e. the department's preference of buildings. In the Geographic File, each department is associated with a list of buildings which are ordered according to its preference. Buildings listed foremost have higher priority than those listed later.

The set of variable data files needed prior to scheduling has four members:

- **Course-data File** is the largest input data file. It contains information about all courses to be scheduled. Information about a course described in this file includes: *course code, course section, lab code, semester, course format, teaching professor(s), room requirement, enrollment, time zone, best-building,* and *pre-assignment values* (if any).

- **Professor-unavailability File** specifies the time blocks when a professor is occupied by other duties. Professors can identify one or more time blocks to be marked off when the system schedules their lectures. The faculty/department may also specify some time for staff meetings, thus during this time all professors of this department should be free from teaching. This file is linked to the Course-data File by the key of professor names.

- **Room-reservation File** contains all the room reservation data. A room can either be taken off from the scheduling processing completely or some of its scheduling time blocks are marked off.

- **Course-conflict File** includes all information about course conflicts. Conflicts may come from the requirement of the first order constraints of the system as well as some occasions specified by the Scheduling Office.

4.2.3.2. Scheduling Algorithm

The final timetable is the integration of all the information in the seven input files. So only when the data in these files are complete and accurate, can the scheduling process start. The function of keeping input files complete and accurate is handled by the supporting utility programs, in particular, the preprocessing programs.

The scheduling process of CTS is a sequential one in which courses are processed one by one, but not necessarily in the order of the sequence they are input. One course is scheduled when it goes through one *scheduling cycle*. During the cycle, assignments to some other courses may be deleted and these course are rescheduled in order to accommodate the courses currently being scheduled.

4.2.3.2.1. Courses with preassigned values

Courses with preassigned values are processed first. They are also handled differently from those without preassigned values.

There are three different cases of preassignments. Some courses are completely pre-scheduled, i.e. their teaching time and classrooms have been decided manually prior to the start of the scheduling programming. Some courses are only given teaching time while their classrooms are waiting to be found (time preassigned). Other courses can only go to certain classrooms but the teaching time is not set (room preassigned).

For completely prescheduled courses, the program will record the assignment and make sure the room-time pairs will not be used again by other courses.

Steps to process time preassigned courses are:

- find a room which is free for the preassigned time blocks and satisfies the room requirement of this course;
- check the time-room pairs to see whether any conflicts will be caused; if yes, backtrack to the last step to find another room.

Similarly the scheduling program will process room preassigned courses in the following steps:

- find a time block which is in the right time zone and when the preassigned room is free;
- check the time-room pair whether any conflicts will be caused; if yes, backtrack to the last step to find another time block;
- if the format of the course needs more than one time block, such as $2\times1\frac{1}{2}$ courses, repeat steps 1 and 2.

4.2.3.2.2. Courses without preassigned values

The general algorithm (GA) of scheduling a course without any pre-assigned values is

- find a time block which is in the proper timezone;
- find a classroom which satisfies the room type requirement;
- check this time-room pair whether it causes any conflicts, i.e. breaking any first order and secondary constraints, if yes, backtrack to the above steps.

The program may find no answer after the above three steps have explored all the combinations of time blocks and rooms. In this case, we realize that the constraints allowed for this course have been too restrictive and that they should be relaxed somehow. According to our specification, only secondary constraints can be disabled; so we will delete those secondary constraints of Level 1 (See § 3.2.1) and go through the three steps (GA) again.

If the program still cannot find an assignment, the secondary constraints of level 2 will be relaxed and the program tries again with the GA steps.

In this way, secondary constraints of level 3, level 4, or even level 5 will be disabled until the program finds an acceptable assignment.

There is a possibility that the above tries still cannot schedule the course, i.e. the course is unschedulable even after all the secondary constraints have been disabled. There are certain possible causes of this situation. Of course, the most obvious reason is that no timetable that satisfies the input

data exists. This case rarely happens because from past manual scheduling experience we know that in most institutions the size of their physical teaching resources is usually larger than required and that usually more solutions do exist as manual scheduling has always been able to find at least one. A more plausible reason is that some courses which are scheduled before have taken the only possible assignments of the current course while there are other solutions for these courses. Changing the processing sequence of courses may solve the unschedulable problem. But which courses are most probably occupying the only possible "seat" of a given course? The logical answer is *courses with similar input constraints*. Because these courses ask for the same kind of assignments as the given course, they are the obvious and most possible suspects of "robbing" away the scheduling possibility of the course. Based on this assumption, we have built an *equivalent reversing* procedure to handle the unschedulable situations.

4.2.3.2.3. Equivalent reversing

The equivalent reversing(ER) is actually another way of backtracking in CTS. When a course A becomes unschedulable, the ER first chooses an equivalent course B which has already been successfully processed, i.e. has been given proper assignments, based on a group of criteria. It then deletes B's assignments, and tries to reschedule B after successfully scheduling A. If A cannot be scheduled after deleting B's assignment, ER will resume B's assignment and go on to choose another course according to the criteria and repeat the deleting-schedule-reschedule process.

For the purpose of equivalent-reversing in CTS, courses that satisfy all the following criteria are considered equivalent:

- their enrollments fall in the same size range;
- they are in the same timezone;
- they require the same type of rooms; (room type)
- they request the same building.

In order to avoid infinite looping and also for the sake of the system's efficiency, we have disallowed nested equivalent reversing, i.e. calling up an equivalent reversing procedure from inside one equivalent reversing. The number of *equivalent* courses that could be selected in one reversing process has also been limited to 4. Furthermore, assignments of completely pre-scheduled courses cannot be deleted.

If the scheduling program cannot find an assignment for one course after even the equivalent reversing procedure, it will put the course into the list of unschedulable courses which will be manually scheduled later.

4.2.3.2.4. Size limitator

A concept of *size limitator* has been introduced in CTS to guard the room seating usage in the resulting timetable. We have divided classrooms into groups according to their sizes. Each group is identified by a *range number*:

Size Range Chart	
Size	Range Number
301 -	19
201 - 300	17
151 - 200	15
101 - 150	13
71 - 100	11
51 - 70	9
31 - 50	7
19 - 30	5
11 - 18	3
1 - 10	1

[Figure 3]

Courses are also given range numbers by their enrollments according to the same schedule.

When scheduling a course, CTS will first try classrooms that have the same range number, say 5, as the course does. When all the classrooms of this range have been tried and no suitable assignments found, CTS will go up one range, search from the rooms having a range number higher than the current range, 7. Constraint 1.05 requires that the seating usage of classrooms be kept not lower than a pre-defined level. In implementing CTS, we have used *size limitator* to achieve this goal. Each size range is associated with a size limitator and the size limitator of a size range indicates the largest classrooms into which courses whose enrollments fall into the size range may be assigned. For example, if the size limitator of Size range 5 is 2, any courses whose enrollments are range 5 can only be scheduled classrooms in range 5, range 7 and range 9, i.e. a course of range R whose size

limitator is **L** can only be scheduled into classrooms no larger than range **R+2*L**.

The size limitators can be adjusted based on the result of previous runs. If there are a large number of courses of a certain range left unscheduled after one run, the reason might be that the number of classrooms of the same range or one range above is too small. Thus increasing the size limitator of the range may help in the next run.

4.2.3.3. Output Data

Due to the existence of the equivalent reversing procedure, assignments to any courses which are not completely pre-scheduled are not definite until the assignment of the last course is decided. So the assignments of courses are written into output files only when an execution terminates.

Two files are produced, one contains all the assignments made during the scheduling process and the other includes data of all the courses that could not be scheduled. The first file will then be processed by the post-processing programs to update the *HORAS FILE* and to produce several schedules for testing and practical use. These schedules are described when we discuss the post-processing programs in § 4.2.4.3.

Unschedulable courses listed in the second file will be manually scheduled by the Scheduling Office and results will also go into the *HORAS FILE*.

4.2.4. Supporting Utility Programs

Supporting utility programs are designed for the integration of CTS with the existing scheduling software at the University. They collect and process the input data for the scheduling program as well as updating the main database *HORAS FILE* with the output data from CTS. The detailed relationship of supporting untility programs with the scheduling program is illustrated in *Appendix A -The System Structure*.

4.2.4.1. *Hello* - the data collecting system

Hello is an online data entry system designed for scheduling officers of the faculties and departments at the University of Ottawa. It is the interface system to CTS.

Hello reads course data from the most up-to-date course database *HORAS FILE* and accepts other data about the courses which are required by the CTS's scheduling program. Through *Hello*, the course data in *HORAS FILE* will also be verified by the faculties/departments.

It is a friendly screen-driven system consisting of three main screen displays: Introduction Screen, Department Code Screen, and Course Information Screen. Users can edit, delete and insert the course records of their respective departments by following the instructions on the screen. These changes are subject to approval from the Scheduling Office.

After the approval from the Scheduling Office, files produced by *Hello* are transformed into the major input data file for CTS, Course-data File.

4.2.4.2. The pre-processing programs

CTS has a group of pre-processing programs processing and organizing input from different source files. These programs transform information in the source files into forms that can be easily and efficiently processed by the main scheduling program.

Besides *HORAS FILE*, pre-processing programs have many other input sources:

- files produced by different faculties/departments through using *Hello*. They are organized into the main input file, Course-data File;
- Nonoverlapping and overlapping files which contain the information about student programs. They are formed by faculties/departments with the supervision of the Scheduling Office. The information in them consists of the contents of the Course-conflict File;
- files containing information about classroom reservations. Their contents are transferred into Room-reservation File;
- files about professors' unavailable time blocks. They are used to produce the Professor-unavailability File;
- files about building preference of faculties/departments. These files are provided by faculties and departments and are used for the creation of Geographic File. As the information

contained in this file is rarely changed, Faculties/departments need only report changes to the Scheduling Office once the file has been created.

4.2.4.3. The post-processing programs

The post-processing programs take the output file from the scheduling program as input and produce a series of schedules for testing and practical purposes. They also update *HORAS FILE* with the new scheduling assignments.

For testing and verification purposes, the following schedules or reports are produced from the output data of the scheduling program:

- a list of all assignments made by the program in the order of course codes;
- a list of all assignments grouped by professors;
- a list of all assignments grouped by classrooms; the seating usage of each room and the average of all rooms are included in the list.

4.3. USING THE CTS SYSTEM

4.3.1. Using CTS to Produce Timetables

Using CTS to produce timetables involves three major steps: preparing input data, running the scheduling program, and producing and analyzing test reports. If the result is not satisfactory, adjust the input data and repeat the last three steps.

The first step of using CTS to produce timetables is to collect and process input data for the scheduling program. Every faculty/department is asked to prepare the course data file for their faculty/ department using the *Hello* system, the nonoverlapping file, the overlapping file, the classroom reservation file, the professor availability file, and the optional geographic file. Along with some special reservations from the Scheduling Office, these files are processed by the pre-processing programs to produce the seven input files for the scheduling program.

When the input files are complete and accurate, the scheduling process starts. It takes all the input data and produces assignments and a list of courses that could not be scheduled by the scheduling algorithm.

The third step involves producing reports for testing with the output data from the scheduling program, analyzing the results, and making decisions if the timetable produced is acceptable or the scheduling program should be re-run. The Scheduling Office evaluates the timetable generally and may reject it for many reasons. One typical reason is that there are too many courses left unscheduled. Then the input data will need to be adjusted and the scheduling program re-run. Ways of adjusting input data include deleting unnecessary constraints, changing course sequences, and even adding new rooms.

Even if the timetable is satisfactory, the scheduling program may be rerun with adjusted input files to see if better timetables can be produced.

4.3.2. Using CTS for Other Purposes

CTS can be used for other purposes, besides its main application area of producing timetables. As each run of the main scheduling program takes no more than a few hours of CPU time (in our test done so far, only less than 30 minutes required), the University may test the results of removing certain physical resources (classrooms) or adjusting the size distribution pattern of classrooms. CTS may also be used to test different slot-systems. Its character of modularity enables the easy modification of the system. Without too much work of reprogramming, we could build a scheduling system of other slot-systems such as multi-length slot system. With the same set of input data, we would be able to compare the timetables created by two different systems.

Chapter 5 THE TESTING OF CTS

5.1. INTRODUCTION

The complete testing of the CTS system prior to its practical implementation consists of three stages while so far we have gone through the first two. During the first test stage, we have tested the system with what consists of a subset of the complete input data, data of the Faculty of Science at the University of Ottawa. In the second stage, due to the fact that the development of the data collecting tool, *Hello* - the interface system, had not been finished, we have used the previous year's data of the entire university. Test stage three, in which the CTS will be running in parallel with the current timetabling process in a "simulated" real world, will be conducted from December of 1989 to April of 1990. It is expected that with the completion of this test stage the CTS will be formally put into practical use for the academic year 1991-1992.

The testing process is also part of the evolution of CTS. The frequent contact with the Scheduling Office, the destined user of CTS, has revealed many problems and constraints ignored by our initial specification and implementation. It seems that new problems appeared every time we sent in results of testing runs. CTS matures in this process of testing, modification, retesting and re-modification.

In this chapter, we will report the results of the Test stage one and two (§ 5.2 and § 5.3) as well as the data collection process for the Test Stage Three which is currently being carried on with the cooperation of the Scheduling Office at the University (§ 5.4).

5.2. TEST STAGE ONE

5.2.1. Input Data

The first stage of CTS testing is done with a subset of the data of the whole university. We have used the data of the Fall term, 1989 of the Faculty of Science along with a subset of the classroom resources. As science students take courses of other faculties and students from other departments may also select science courses, the test results may well not reflect the picture of the whole university. But by running the system with a reasonable size subset of the whole database, we feel that potential problems are more detectable and debugging is much easier than running with the complete data set.

The Faculty of Science is composed of seven departments offering a total of about 250 courses for each of the Fall and Winter terms. For the fall term of 1989, the whole Science Faculty has 249 courses taught by 120 professors. Each course may have lectures and labs so it may need more than one format to describe. Each format is expressed by one entry.

For example: A course of computer science, CSI1500B, has two 1 ½-hour lectures and two 1 ½-hour labs. So we need two entries, each standing for a 2x1½ h format request. (Even if we had a format of 4x1½-hour still this course would be better treated as two entries of 2x1½-hour format as the enrollment of the lectures is different from that of the labs.)

The 249 courses of Fall term in 1989 are represented in 426 entries. The range distribution of these entries is as following:

Course Range Distribution (Test 1.1)										
Range #	1	3	5	7	9	11	13	15	17	19
# of Entries	141	63	72	41	33	25	20	15	13	3

[Figure 5]

There are about ten different course formats used by science courses. Besides testing the CTS system which adapts the one standard slot system allowing only four basic formats, [Refer § 4.2.1] we have created another version of the scheduling program to accommodate the current multi-length slot systems. [See Appendix B.2] We will call this program *M-CTS* in contrast to *CTS*.

In testing the CTS system, we have transferred formats not admissible in the standard-slot system into the basic formats using the following formula: [Figure 6]

Format Transformation Chart	
Old Format	New Format
1 x 1 h	1 x 1½ h
1 x 3½ h	1 x 3 h
3 x 1 h	2 x 1½ h

[Figure 6]

Courses of other irregular formats are considered as preassigned courses and they took the assignments in the current university timetable which is done manually. Of the 426 entries of the Faculty of Science, there are 28 requiring to be preassigned.

The student program information is extracted from the official *Science Calendar, University of Ottawa, 1989-1990*. According to the calendar, we have grouped these courses into 50 programs which produced 605 pairs of time-conflict courses. Each pair represents two courses that must not be conflicting in teaching time.

On the campus at the University of Ottawa, all faculties/departments share a classroom pool of 168 controlled by the Registrar's Office. This means that each classroom may have courses from different departments. Each faculty or department may have its own classrooms or special laboratories. CTS is not responsible for the scheduling of these department-controlled rooms. Any courses that need to use these rooms have to be pre-scheduled manually and their assignments will be entered into CTS input files as preassignments. For the purpose of Test stage one, we have used 23 classrooms which are popular among science faculty courses. (The reason we call them popular is that historically they have been used mainly for accommodating science courses in the current timetabling practice.) Following is the range distribution of these rooms:

Room Range Distribution (1)										
Range #	1	3	5	7	9	11	13	15	17	19
# of Rooms	0	0	4	6	6	2	2	1	1	1

[Figure 7]

In collecting the information about professors' unavailability, we have used the office hours posted by professors as their unavailable time blocks. There are 226 professor-timeblocks. Each professor-timeblock represents one timeblock when a professor is not free. This type of data is stored in Professor-unavailability File. [See § 4.2.3.1]

5.2.2. Result Analysis

Observing the large difference between the range distribution charts of course entries and classrooms, [Figure 5 & Figure 7], we expected that the setting of size limitators would play an important role in the scheduling of the set of entries.

During the first run, we have set all size limitators to 0, i.e. all courses can only use classrooms of their proper size ranges. For example, a course having an enrollment of 100, which falls into range 11, may only be scheduled into rooms of range 11. The result of CTS program is:

CPU time (CPU): 167228 msc or 167 seconds
Preassigned entries (PE): 28
Entries scheduled (ES): 261 (including preassigned entries)
Unschedulable entries (UE): 185
The average seating usage (ASU): 85.4%
The average room utilization (ARU): day 51.2%, evening 49.7%

and the result of M-CTS is:

CPU time (CPU): 236754 msc or 237 seconds
Preassigned entries (PE): 28
Entries scheduled (ES): 211 (including preassinged entries)
Unschedulable entries (UE): 215
The average seating usage (ASU): 85.7%
The average room utilization (ARU): day 48.8%, evening 42.1%

The distribution of those unschedulable entries is as following:

Distribution of Unschedulable Entries (Test 1.1)											
	Range #	1	3	5	7	9	11	13	15	17	19
# of UE	CTS	126	52	2	0	0	2	0	3	0	0
	M-CTS	126	52	17	7	0	4	5	2	2	0

[Figure 8]

Even though the seating usage is high, there are a lot of courses left unscheduled, almost half of all the entries if the preassigned entries are not counted. It is obvious that all entries of range 1 and 3 could not be scheduled because there were no classrooms of range 1 and 3. Thus size limitators of range 1 and 3 should be adjusted or some classrooms of range 1 and 3 should be added to the classroom pool. In the second run, we have used the following size limitators:

Size Limitators for Test 1.2										
Range #	1	3	5	7	9	11	13	15	17	19
Size Limitator	4	2	1	0	0	0	0	0	0	0

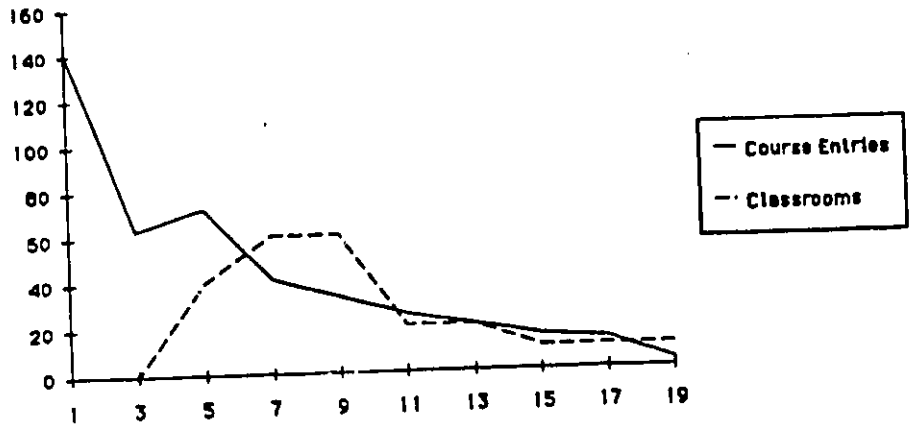
[Figure 9]

In order for easy comparison we have put results of the test run two together with the run one:

Results of Test Run in Stage One							
Run #	CPU	PE	ES	UE	ASU	ARU-day	ARU-eve
CTS-1	167	28	261	185	85.4%	51.2%	49.7%
CTS-2	152	28	421	5	74.6%	73.9%	51.1%
M-CTS-1	237	28	211	215	85.7%	48.8%	42.1%
M-CTS-2	210	28	388	38	74.8%	67.8%	50.3%

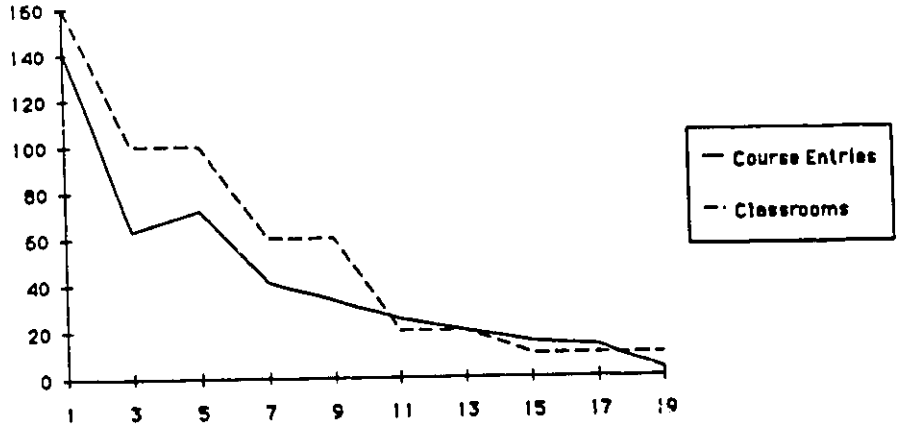
[Figure 10]

By varying the size limitators we have increased the number of courses being scheduled by respectively about 60 and 82 percent for CTS and M-CTS. Ideally, the range distribution chart of courses should be the same as that of classrooms. But it is rarely the case that the ideal situation happens. We should adjust the size limitators to make the range distribution of classrooms available closer to that of courses. Fig.11 shows the line graph of the room and course distribution. We have increased the figures of room ten times to make the two graphs compatible.



[Figure 11]

After adjusting the size limitators, the graph for the room range distribution changes to the one in Figure 12. In Figure 12, as in Figure 11, the vertical values of the classroom distribution graph have been multiplied by a factor of 10.



[Figure 12]

It should be noted that these graphs can only roughly (not accurately) display the request and demand relationship between courses and classrooms and there are many other important factors affecting the availability of classrooms for course entries. The geographic restraint may reduce the number of rooms suitable for course entries, as well as other constraints. Even the dayzone factor has an impact on the demands for classrooms. For example, if a set of course entries of range A are largely evening classes, the number of classrooms needed to schedule them may be much larger than that for a set of course entries of range B which has more members but the members are evenly spread among all the teaching times, mornings, noon, afternoons, and evenings. Thus two entries of the same size range will not necessarily have same number of schedulable classrooms. But those distribution graphs can provide a general picture. Especially in this case, there are a large number of course entries involved which, generally speaking, are random data. Another way of adjusting size limitators is to use the result of previous runs.

Using the one-standard slot system has proven superior to using the multi-length slot systems. First of all, the computing time of M-CTS is obviously longer than that of CTS. The speed of M-CTS is slowed down by the extra operations needed by the system in scheduling each entry to check potential conflicts between two different length slot systems. The higher the number of course entries involved, the larger effect those extra operations will have on the speed. The results of test stage two will prove this point.

Two slot systems may also leave empty ½-hour holes in the timetable that could not be utilized. These unusable holes reduce the real scheduling space. This phenomenon is illustrated in Figure 10 where M-CTS schedules fewer course entries and results in lower average room utilization rate than CTS does.

5.3. TEST STAGE TWO

5.3.1. Input Data

For test stage two, we have input all the courses being offered on the main campus of the University for the fall term of 1989 as well as all the room resources controlled by the scheduling office. Not included in the input file are the courses offered by departments of the Faculty of Health Sciences which is located on a separate campus.

There are 1657 courses being transferred into 2268 course entries in the Course-data File. Among these entries there are 96 being treated as prescheduled entries for the reason that they do not follow any of the four basic formats. Another 59 entries are input as prescheduled for other reasons.

In total there are 167 classrooms whose sizes range from 4 to 474. The distributions of these rooms as well as course entries are listed in Figure 13. In calculating the distribution of course entries, we excluded those entries(121 in total) that require department-controlled rooms.

Course Entry and Room Range Distributions (2)										
Range #	1	3	5	7	9	11	13	15	17	19
# of Entries	156	160	871	387	267	189	63	30	21	3
# of Rooms	9	17	44	31	35	18	4	5	2	2

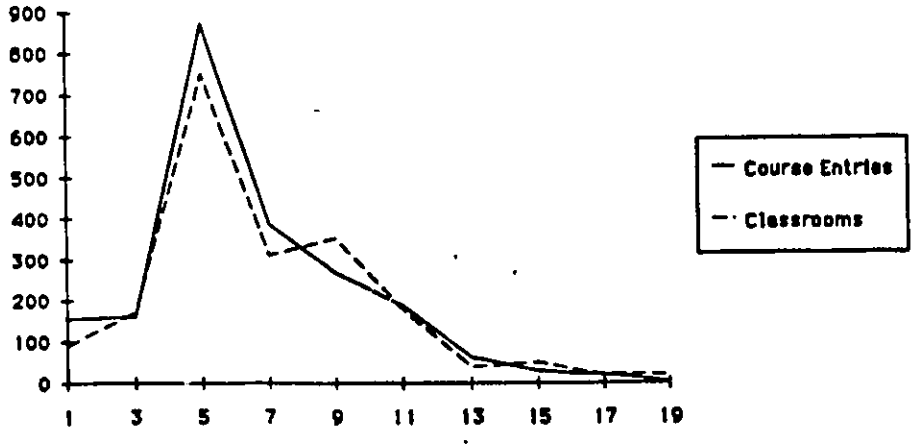
[Figure 13]

Based on the analysis of the distributions of course entries and classrooms, we have used the following size limitators in our first run of stage two. [Figure 14]

Size Limitators for Test 2.1										
Range #	1	3	5	7	9	11	13	15	17	19
Size limitator	0	0	1	0	0	0	0	0	0	0

[Figure 14]

The line graphs of the course entry and classroom distributions after applying the size limitators are shown in Figure 15.



[Fig. 15]

We included 363 student programs which resulted in 14,329 pairs of time-conflicting courses. As each course may be represented by more than one entry, the number of time-conflicting course entry pairs implied by these data is even bigger.

The information about the professor unavailability had to be collected from different departments as there were no existing files containing such information. After we finished compiling the *Professor-unavailability File*, we were surprised to notice that the number of requests were much fewer than we expected. We have only 256 professor-timeblocks, compared to 226 in the test stage one.

5.3.2. Result Analysis

Following is the result of Test 2.1, i.e. the first run of test stage two:

Results of Test 2.1							
System	CPU(secs)	PE	ES	UE	ASU	ARU-day	ARU-eve
CTS	1427	155	1981	132	85.5%	55.4%	59.0%
M-CTS	11021	155	1818	295	85.2%	50.9%	57.7%

[Figure 16]

The running time of M-CTS is nearly 8 times as much as that of the CTS scheduling program. This demonstrates again that the extra operations needed to process the compatibility of two different slot-systems will hurt the performance of the system. The extraordinary inconvenience of testing M-CTS coming from the lengthy running(11021 seconds of CPU time usually means about 10 to 12 hours waiting after we started the program) has prohibited us from testing it as many times as we did to CTS. On the other hand, the performance of CTS is very satisfactory, both on the speed aspect and the number of entries being scheduled.

Distribution of Unscheduled Entries (Test 2.1)											
	Range #	1	3	5	7	9	11	13	15	17	19
# of UE	CTS	7	10	48	10	33	8	8	4	4	0
	M-CTS	30	19	79	21	80	9	35	15	9	0

[Figure 17]

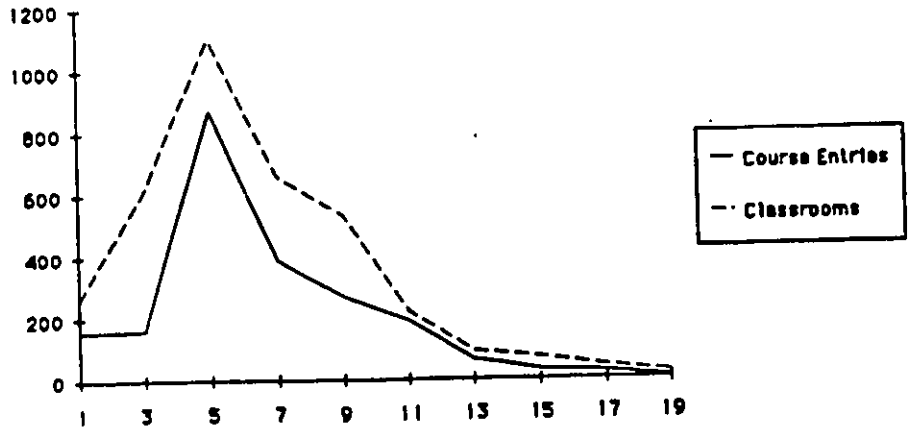
The output has also proved that the availability of classrooms is not determined by the single factor of how many rooms are available for each range. There are other factors having impact. So adjusting size limitators simply based on the distributions of entries and classrooms will not completely solve the problem of unschedulable entries.

For the second run of the this stage, we have used following size limitators:

Size Limitators for Test 2.2										
Range #	1	3	5	7	9	11	13	15	17	19
Size limitator	1	1	2	1	2	1	1	1	1	0

[Figure 18]

and the line graphs of the room and course distribution looked like: [Fig. 19]



[Figure 19]

The result of Test run 2 is:

Results of Test 2.2							
System	CPU(secs)	PE	ES	UE	ASU	ARU-day	ARU-eve
CTS	1402	155	2113	0	80.2%	58.5%	60.4%
M-CTS	10733	155	2066	47	80.8%	58.3%	55.9%

[Figure 20]

The results of all the test runs indicate that our initial objectives have been mostly achieved.

-In compiling the student programs, we have considered the factor of recommended courses which have never been taken into account in the current manual approach. Thus the timetable produced by the CTS system will provide more choices of selecting courses for students;

-Current rates of seating usage and time utilization are around 70% and 50% respectively. While the improvement of time utilization is not obvious as it is largely affected by the total number of courses offered, the figures in Figure 20 have shown some increase in the average seating usage.

-CTS scheduling program takes into account the geographic situations of host departments while making assignments. It will try to satisfy the geographic requirement defined by the host departments as much as possible.

-Using CTS will largely reduce the amount of manual work required by the timetabling process. Communication between the Scheduling Office and the scheduling officers at the faculty/department level will be improved by the using of *Hello*, and consequently the work needed for preparing the data base *HORAS FILE* also becomes easier.

5.4. THE DATA COLLECTION FOR TEST STAGE THREE

5.4.1. The Test Plan for Stage Three

With the approval from the University administration, Test Stage Three, also the final test stage before CTS is put into practical use, has been designed to test the CTS system in a "simulated" real world, in which all test data are the same as those used by the current manual scheduling process in constructing timetables for the next academic year, 1990-1991. The Scheduling Office at the University will be taking charge of the work of data collection and output evaluation.

The complete procedure of the testing will be conducted in three steps: data collection, test run, and result evaluation.

Data collection: Planned to be conducted from October of 1989 to January of 1990. This process will collect all the most up-to-date information about all courses being offered on the main campus at the University of Ottawa. Traditionally at the University a new timetable is built upon modification to an old one, usually the one used for the same term of the previous year, and departments send all the changes to the Scheduling Office on forms. For the purpose of testing (also implementing in future) CTS, a data collecting system, *Hello*, (See § 4.2.4.1) has been developed to collect data about courses that are unavailable in the current course data base but required by the CTS system. Other information necessary for CTS's running is prepared by departments according to a standard format specified by the Scheduling Office.

Test Run: After the completion of the data collecting process, the stage of test run will last from February to March of 1990. Tasks included in this step include organizing all information into input files for the scheduling program of CTS, and running the system with these input data to produce complete timetables.

Result Evaluation: The examination and evaluation of the result of the test run will start in April of 1990. The resulting timetable will be compared to the timetable constructed by the normal manual process. Lecture schedules will be produced for individual professors for evaluation and comments.

5.4.2. Data Collecting Sources

Currently, we have started the process of data collection. To run the scheduling program of CTS, we need 7 input files that are listed and described in § 4.2.3.1. The data inside these files are collected from various sources in different ways.

The completion of the design and implementation of *Hello* has facilitated the process of collecting course data from various faculties/departments. Using Hello we get all the data for **Course-data File**.

Time-Slot File and **Room-Inventory File** are prepared by the Scheduling Office. Once they are created, they will stay unchanged until the classroom pool is changed or the slot-system is modified.

Geographic File contains information that is relatively constant. Each department reports its building preference list to the scheduling office which compiles lists from all the departments/faculties into one file. **Professor-unavailability File**, **Course-conflict File**, and **Room-reservation File** are produced in a similar way as the *Geographic File*, though information included in them is very volatile. Information is first collected from all departments and then incorporated into different files respectively by the scheduling office.

5.4.3. Unexpected Problems

Before and during the development of CTS, deans of all faculties/departments have been consulted about their wants and don't wants about CTS. We have made every effort toward a system that satisfies all the users' needs and thought we had solved all the problems until we started the data collecting process. Some unexpected problems were brought up during this step.

Dynamic student programs: Unlike academic programs in science departments, programs for social science and art students are more flexible and thus hard to define by lists of required courses and recommended courses. A typical description of such a program normally consists of a list of core courses and a group of very general guidelines about what the other part of the program should be composed of. For such programs, there are many combinations of courses that fulfill the requirements. Sometimes it becomes practically impossible to explicitly describe every one of these combinations. For example, in the official university calendar for 1989-90, the program of *B.A. with Honours in Theory and History of Art* is defined as:

B.A. with Honours in Theory and History of Art	20 courses
Two of Eng 1120, 1121, 1320, 1321 PHI 1201 Second language test	
ART 1204 Visual Workshop (6 cr.) ART 2010 Experimental media Workshop (6 cr.) ART 2224 General Art History (6 cr.) ART 2234 Premises of Modernity (6 cr.) ART 2242 Arts of the 20 th Century (6 cr.) ART 3222 Art and Criticism (6 cr.)	
Six courses (36 cr.) chosen from ART 2332, 3020, 3105, 3120, 3122, 3123, 3124, 3215, 3222, 3259, 3301, 3320, 3332, 3333, 3334, 4110, 4115, 4116, 4126, 4206	
One course (6 cr.) in three different departments other than the Department of the Honours program (18 cr.)	
Three electives (18 cr.)	
The Department will determine the order in which some courses have to be taken.	

We ignore the fourth to sixth parts which are either too general or irrelevant to our purpose. While the information of the first two parts can be easily described in the form of lists of courses that could not have time conflicts, describing the third part is not as easy. One solution is to put all the 20 courses into one list with courses of the first and second parts of the table. This will guarantee that any two courses will not be scheduled into same timeblock. But by doing so, we have introduced a large amount of unnecessary restraints on the scheduling system and reduce the general schedulability of the courses involved. It is obvious to see that if every program is processed this way, the restraint put on the scheduling program will become so strict that no timetable will exist that could accommodate all the courses and their constraints.

Another solution would be to enumerate all the combinations, that is, test all the possibilities of completing the program. This is obviously impractical as just for the third part of

this program there are 37920 different combinations!

We call these programs *dynamic*, referring to their indefinite characteristics. Our current solution to the problem of dynamic programs is identifying several of the most popular patterns among previous students. This work is done by scheduling officers at respective departments.

Problems inside HORAS FILE: The old scheduling system at the University of Ottawa is mainly a room reservation or room scheduling system. Thus the main data base *HORAS FILE* is also specifically designed for a room reservation system. While converting course information from the file, we have encountered some problems originating from the file's room reservation system nature.

Course enrollments in HORAS FILE are the figures of last year, not the most up-to-date ones which are available from another administration system. The major part of the course scheduling job within the current timetabling system is done at the department level and each department reports its own schedule by sending in classroom requests. These requests are evaluated and, if approved, are entered into the HORAS FILE as new assignments. But the enrollments of courses are rarely modified as after the room requests for some courses are approved, the enrollments of these courses do not really matter any more. What really matter are those assignments. Normally, the enrollments in HORAS FILE are only up-to-date once with the last year's figure at the beginning of a timetabling cycle, i.e. starting point for constructing the next year's schedule.

Resistance to CTS: Although some parts of the university academic community welcome the idea of an automated scheduling system, there is some resistance when the testing stage of CTS is started. One source of resistance to CTS comes from the group of people who fear that they may lose their privileges within the current system. Timetabling involves many political activities among a large number of players, departments, the university administration, individual professors, etc. The assignment of time and room resources many times can be reached only after some complex bargaining process. As a result, some players get the best deals while others don't. A good "deal" may mean teaching at a good time (non early morning, not late afternoon) in a good classroom (close to his/her office, good lighting, good ventilation, good equipment) for a professor, or getting good assignments (good time and good room) for all or most of its courses for a department. "Players" who are getting good deals are afraid that CTS will not treat them "fairly" by treating everyone as equal.

More resistance comes from people who do not believe CTS is able to consider their special requirements. Realizing the complexity of the timetabling problem and the variety of special requests, they feel that it is impossible for a computer system to take every option into account, so they oppose CTS for the thought that their special requirements for the scheduling of certain courses will not be fulfilled.

Resistance also comes from some people who are afraid of computers. These people do not want go near a computer terminal. Implementing CTS means that they have to learn to use the editor of the CMS system.

Chapter 6 CONCLUSIONS AND FUTURE DEVELOPMENT

6.1. CONCLUSIONS

We have described in detail a timetabling algorithm based upon a logic programming model. The algorithm is implemented in the computerized timetabling system at the University of Ottawa (CTS), which is currently under the final test stage before it is put into practical operation.

We started by reviewing the existing models, algorithms and practical applications of the timetabling problem. The complexity of the timetabling problem has been described during this reviewing. Next, we introduced the basics of logic programming and its application in the timetabling problem. In our project, the timetabling is modeled as a constraint satisfaction problem and it is solved using logic programming techniques. We have shown that constraints contained in a timetabling program can be easily transformed into logic expressions in an extended Horn clause form, consisting of the basic elements of the logic programming language, PROLOG. The ease of this transformation has facilitated the specification and development of our system which intends to solve the timetable construction problem at the University of Ottawa.

After explaining the logic model of the system, we have presented some important implementation issues of CTS such as the technical environment and the system specification. We believe that by using a standard timeslot system, we have increased the schedulability of courses in general and the efficiency of resource usage. The concepts of equivalent reversing and size limitator have also been introduced into the system implementation to improve its efficiency.

The two test stages completed so far demonstrate the efficiency of our algorithm and the superority of the one-standard slot system. The introduction of the size limitator provides a way of adjusting the number of available classrooms for groups of course entries in general. It has been shown in the tests that adjusting size limitators according to analysis to the distribution graphs of classrooms and course entries could largely reduce the number of unschedulable entries.

Our algorithm is a heuristic method as it does not guarantee that it will find a solution if there is one. But the situation in most large institutions is that there are more resources available than necessary and usually there is more than one solution available to their scheduling problems. This can be proved by the fact that even the manual process, which searches through many fewer combinations than a computerized system, is able to find a solution in most cases. In addition, the enormous size of the combinations makes an exhaustive, or even an semi-exhaustive search impractical. We can assume that by properly adjusting the available room distributions, our system will be able to find an acceptable solution under general circumstances.

From our experience in building the computerized timetabling system at the University of Ottawa, we see a lot of potential in the research topic of applying logic programming to the problem of timetabling in large educational educations. The size of the data for scheduling at the University of Ottawa is fairly comparable with that of an average large institution and the case we considered is also more complicated than situation in most universities. Our system, at least from the results of the test stages, has shown its ability to solve the problems of timetabling with satisfactory efficiency. Besides, the speed of the scheduling algorithm of the CTS system could be further improved by the new research results in the field of logic programming, such as the techniques of improving the backtracking, the introduction of the domain concept in PROLOG, forward chaining, and many others.

6.2. FUTURE DEVELOPMENT

As we have just mentioned, our system could be further improved by using the new research results in logic programming. These techniques could largely reduce the computing time of the main scheduling algorithm. Currently each run of the main scheduling program of CTS needs about 24 minutes CPU time on the mainframe system at the University of Ottawa. For a large institution like the University of Ottawa, this time is quite satisfactory. But reducing the length of running is still an attractive development. Once the CTS system is fully tested and put into practical use, we will start working on this area.

Besides improving the system's speed, there are some other projects being considered and maybe put into our plan for future development:

- An interactive system.
- Study on factors affecting the number of rooms available.
- Incorporation of student pre-registration data.
- Study of the classroom overbooking problem.

Chapter 7 BIBLIOGRAPHY

7.1. INTRODUCTION

References are grouped into two parts, **timetabling** and **logic programming**. The first class includes all the references on the problem of timetabling while the second one contains bibliography about logic programming. Publications on applying logic programming techniques in timetabling and papers on constraint satisfaction problems (CSP) are included the first part.

When being referenced in the thesis, items in the two classes will be denoted as [TT #] and [LP #] respectively, where the # is the sequential number assigned to the item. For example, the full citation of [LP 3] is listed as the third entry in the part of logic programming, i.e. § 7.3.

7.2. REFERENCES ON TIMETABLING

[1] **Akkoyunlu, E.A.** "A Linear Algorithm for Computing the Optimum University Timetable", *Computer Journal*, vol.16, no.4, pp.347-350, 1973.

[2] **Almond, M.** "A University Faculty Timetable", *Computer Journal*, vol.12, pp.215-217, 1969.

[3] **Aubin, J.** and **Ferland, J.A.** *A Large Scale Timetabling Problem* , Publication No. 568, Departement d'Informatique et de Recherche Operationnelle, Universite de Montreal, 1986.

[4] **Aust, R.J.** "An Improvement Algorithm for School Timetabling", *Computer Journal*, vol.19, pp.339-342, 1976.

[5] **Barraclough, E.D.** "The Application of a Digital Computer to the Construction of Timetables", *The Computer Journal*, vol.8, pp.136-146, 1965.

[6] **Berghuis, J., Heiden, A.J.,** and **Bakker, R.** "The Preparation of School Time Tables by Electronic Computer", *BIT*, vol.4, pp.106-114, 1964.

[7] **Bloomfield, S.D.** and **McSharry, M.M.** "Preferential Course Scheduling", *Interfaces*, vol.9, pp.24-31, 1979.

[8] **Brelaz, D.** "New Methods to Color the Vertices of a Graph", *Communications of the ACM*, vol.22, no.4, pp.251-156, 1979.

[9] **Brittan, J.N.G.** and **Farley, F.J.M.** "College Timetable Construction by Computer", *Computer Journal*, vol.14, no.4, pp.361-365, 1971.

[10] **Bybee, R.A.** *Academic Class Scheduling Summary of Systems and Procedures*, Report presented to the 34th Annual College and Administrative Computer Users Conference, Boston, Mass., April, 1989.

[11] **Bybee, R.A.** *Predicting the Impact of Curriculum Changes on Course Offering and Room Scheduling*, Report presented to the 34th Annual College and Administrative Computer Users Conference, Boston, Mass., April, 1989.

[12] **Carmel, C.** and **Itzovitz, M.** "A Comprehensive University Planner Implemented in a 5th Generation Language", *Proceedings of 33rd Annual College and Administrative Computer Users Conference*, Los Angeles, Calif., pp.418-428, 1988.

[13] **Carter, M.W.** *A Decomposition Algorithm for Practical Timetabling Problems*, Working Paper #83-06, Department of Industrial Engineering, University of Toronto, 1983.

[14] **Carter, M.W.** "A Lagrangian Relaxation Approach to the Classroom Assignment Problem", *INFOR*, vol.27, no.2, pp.230-245, 1989.

[15] **Carter, M.W.** "A Survey of Practical Applications on Examination Timetabling", *Operations Research*, vol.34, no.2, pp.193-202, 1986.

[16] **Carter, M.W.** and **Tovey, C.A.** *When Is The Classroom Assignment Problem Hard?* Working Paper #89-03, Department of Industrial Engineering, University of Toronto, 1989.

[17] **Clementson, A.** and **Elphick, C.H.** "A Result for an Extended Version of the Simple Timetabling Problem", *INFOR*, vol.20, no.2, pp.168-172, 1982.

[18] **Colijn, A.W.** "A Sectioning Algorithm", *INFOR*, vol.11, no.3, pp.210-225, 1973.

[19] **Colijn, A.W.** *Interactive Construction of University Timetables*, Conference of the Computer Science Association, preprint, Fredriction, New Brunswick, 1977.

[20] **Colijn, A.W.** *Reduction of Second-Order Conflicts in Examination Timetables*, Working paper, University of Waterloo.

[21] **Csima, J.** and **Gotlieb, C.C.** "Tests on a Computer Method for Constructing School Timetables", *Communications of the ACM*, vol.7, no.3, pp.160-163, 1964.

[22] **Dechter, R.** "Learning while Searching in Constraint-Satisfaction-Problems", *Proceedings of Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, pp.178-183, August, 1986.

[23] **Dechter, R.** and **Pearl, J.** "The Anatomy of Easy Problems: A Constraint-satisfaction Formulation", *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, pp.1066-1072, August, 1985.

[24] **Dempster, M.A.H.** "On the Gotlieb-Csima Time-Tabling Algorithm", *Canadian Journal of Mathematics*, vol.20, pp.103-119, 1969.

[25] **Desroches, S., Laporte, G.** and **Rousseau, J.M.** "HOREX: A Computer Program For The Construction Of Examination Timetables", *INFOR*, vol 16, No.3, pp.294-298, 1978.

[26] **Devine, M.D.** and **Kumin, H.J.** "A Time-Sharing Interactive Program for Class Scheduling", *Computers and Education*, vol.1, pp.33-38, 1976.

[27] **Dowsland, W.B.** and **Lim, S.** "Computer Aided School Timetabling: the history of computerized timetabling", *Computers and Education*, no.42, pp.22-23, 1982.

[28] **Dunstan, F.D.J.** "Sequential Colorings of Graphs", *Proceedings of the 5th British Combined Conference*, Aberdeen, England, pp.151-158, 1975.

[29] **ECA, Excellere Associates, Inc.** *Solutions In Software*, Brocheurs distributed at the 34th Annual College and Administrative Computer Users Conference, Boston, Mass., April, 1989.

[30] **ECS: Environmental Consulting Services, Toronto** *Master Plan Study: Draft Report - Faculty of Science and Engineering, University of Ottawa*, University of Ottawa, April, 1986.

[31] **Even, S., Itai, A.** and **Shamir, A.** "On the Complexity of Timetable and Multicommodity Flow Problems", *SIAM Journal of Computer*, vol.5, no.4, pp.691-703, 1976.

[32] **Feldman, R.** and **Golumbic, M.C.** "Constraint Satisfiability Algorithms for Interactive Student Scheduling", *Proceedings of Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, pp.1010-1015, August, 1989.

[33] **Ferland, J.A.** "The Timetabling Problem", *OR Models on Microcomputers*, (J.D. Coelho and L.V. Tavares, ed.) Amsterdam: North-Holland Publishing Co., pp.97-103, 1986.

[34] **Ferland, J.A., Babin, G.** and **Aubin, J.** *A Course Scheduling System*, Publication No.590, Departement d'Informatique et de Recherche Operationnelle, Universite de Montreal, 1986.

[35] **Ferland, J.A.** and **Roy, S.** "Timetabling Problem for University as Assignment of Activities to Resources", *Computers & Operations Research*, vol.12, No.2, pp.207-218, 1985.

[36] **Foxley, E.** and **Lockyer, K.** "The Construction of Examination Timetables by Computer", *Computer Journal*, vol.11, pp.264-268, 1968.

[37] **de Gans, O.B.** "A Computer Timetabling System of Secondary Schools in the Netherlands", *European Journal of Operational Research*, vol.7, pp.175-182, 1981.

[38] **Glassery, C.R.** and **Mizrach, M.** "A Decision Support System for Assigning Classes to Rooms", *Interfaces*, vol.16, No.5, pp.92-100, 1986.

[39] **Gosselin, K.** and **Truchon, M.** "Allocation of Classrooms by Linear Programming", *Journal of Operational Research Society*, vol.37, No.6, pp.561-569, 1986.

[40] **Gotlieb, C.C.** "The Construction of Class-Teacher Time-Table", *Information Processing 1962*, (C.M. Popplewell, ed.) Amsterdam: North-Holland Publishing Co., pp.73-77, 1963.

[41] **Hilton, A.J.W.**, "School Timetables", *Studies on Graphs and Discrete programming*, (P.Hansen, ed.) Amsterdam: North-Holland Publishing Co., pp.177-188, 1981.

[42] **Hodges, B.** *Report on Classroom Allocation*, Registrar's Office, University of Ottawa, August, 1986.

[43] **Hoppe, L.A.** and **Johnson, J.F.** "Decentralized Master Schedule of Classes", *Proceedings of 34th Annual College and Administrative Computer Users Conference*, Boston, Mass., pp.345-354, 1989.

[44] **Jefferis, D.S.** *Computerized Room Scheduling and Facilities Management*, Report distributed on the 34th Annual College and Administrative Computer Users Conference, Boston, Mass., April, 1989.

[45] **Jones, K.** "New Software Unravels Colleges' Class-Scheduling Snarls", *MIS Week - the newspaper for information management*, August 25, 1982.

[46] **Karp, R.M.** "Reducibility among Combinatorial Problems", *Complexity of Computer Computations*, New York: Plenum Press, pp.85-104, 1972.

[47] **Kassicieh, S.** and **Burleson, D.** *A Decision Support System for Academic Scheduling*, Working Paper, Albuquerque, New Mexico, 1983.

[48] **Keng, N.** and **Yun, D.Y.** "A Planning/Scheduling Methodology for the Constrained Resource Problem", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, pp.998-1003, August, 1989.

[49] **Laporte, G.** and **Desroches, S.** *Examination Scheduling by Computer*, Working Report G-82-03, Ecole des Hautes Etudes Commerciales de Montreal, Montreal, 1982.

[50] **Laporte, G.** and **Desroches, S.** *The Problem of Assigning Students to Course Sections in a Large Engineering School*, Working Report G-84-15, Ecole des Hautes Etudes Commerciales de Montreal, Montreal, 1984.

[51] **Lawrie, N.L.** "An Integer Linear Programming Model of a School Timetabling Problem", *Computer Journal*, vol.12, pp.307-316, 1969.

[52] **Lions, J.** "The Ontario School Scheduling Program", *Computer Journal*, vol.10, pp.14-21, 1967.

[53] **Matula, D.W., Marble, G.,** and **Isaacson, I.D.** "Graph Coloring Algorithms", *Graph Theory and Computing*, (R.C. Read, ed) New York: Academic Press, 1972.

[54] **McNeish, M.F.** "A Microcomputer Based Course/Event Scheduling System Based on Down-loaded mainframe Data: Room Ledger Books Eliminated", *Proceedings of 33rd Annual College and Administrative Computer Users Conference*, Los Angeles, Calif., pp.162-176, 1988.

[55] **Metha, N.K.** "The Application Of A Graph Colouring Method To An Examination Scheduling Problem", *Interfaces*, vol.11, no.5, pp.57-64, 1981.

[56] **Mittal, S.** and **Frayman, F.** "Making Partial Choices in Constraint Reasoning Problems", *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, pp.631-636, July, 1987.

[57] **Mulvey, J.M.** "A Classroom/Time Assignment Model", *European Journal of Operational Research*, vol.9, pp.64-70, 1982.

[58] **Murphy, J.** *School Scheduling by Computer: The Story of GASP*, Educational Facilities Laboratory, 477 Madison Avenue, New York 22, N.Y., 1964.

[59] **Neufeld, G.A.** and **Tartar, J.** "Graph Coloring Conditions for the Existence of Solutions to the Timetable Problem", *Communication A.C.M.*, vol.17, no.8, pp.450-452, 1974.

[60] **Oakford, R.V., Allen, D.W.,** and **Chatterton, L.A.** "School scheduling- practice and theory", *Journal of Educational Data Processing*, vol.4, no.1, pp.16-50, 1966-67.

[61] **Osterman, R.** and **De Werra, D.** "Some Experiments with a Timetabling System", *OR Spektrum*, vol.3, pp.199-204, 1982.

[62] **Papoulias, D.B.** "The Assignment-to-days Problem in a School Timetable, A Heuristic Approach", *European Journal of Operational Research*, vol.4, pp.31-41, 1980.

[63] **Peck, J.E.L.** and **Williams, M.R.** "Algorithm 286 - examination scheduling", *Communications of the ACM*, vol.9, p.433, 1966.

[64] **Peemoller, J.** "A Correction to Brelaz's Modification of Brown's Coloring Algorithm", *Communications of the ACM*, vol.26, pp.595-597, 1983.

[65] **Prosser, P.** "A Reactive Scheduling Agent", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, pp.1004-1009, August, 1989.

[66] **Randall Brown, J.** "Chromatic Scheduling and the Chromatic Number Problem", *Management Science*, vol.19, pp.456-463, 1972.

[67] **Romero, B.P.** "Examination Scheduling in a Large Engineering School: A computer-assisted participative process", *Interfaces*, vol.12, pp.17-23, 1982.

[68] **Rousseau, J.M.** "The Column Generation Principle and the Airline Crew Scheduling Problem", *INFOR*, vol.25, no.2, pp.136-151, 1987.

[69] **Salazar, A.** and **Oakford, R.V.** "A Graph Formulation of a School Scheduling Algorithm", *Communications of the ACM*, vol.17, no.12, pp.696-698, 1974.

[70] **Schmidt, G.** and **Strohlein, T.** "Timetable Construction-an annotated bibliography", *Computer Journal*, vol.23, no.4, pp.307-316, 1980.

[71] **Selim, S.M.** "An Algorithm for Constructing a University Faculty Timetable", *Computers and Education*, vol.6, pp.323-332, 1982.

[72] **Selim, S.M.** "An Algorithm for Producing Course and Lecture Timetables", *Computers and Education*, vol.7, no.2, pp.101-108, 1983.

[73] **Selim, S.M.** "Split Vertices in Vertex Colouring and Their Application in Developing a Solution to the Faculty Timetabling Problem", *Computer Journal*, vol.31, no.1, 1988.

[74] **Smith, G.** "On Maintenance of the Opportunity List for Class-Teacher Timetable Problems", *Communications of the ACM*, vol.18, no.4, pp.203-208, 1975.

[75] **Smith, G.** and **Sefton, I.M.** "On Lion's Counter Example for Gotlieb's Method for the Construction of School Timetables", *Communications of the ACM*, vol.17, no.4, pp.196-197, 1974.

[76] **Smith, L.D.** "The Application of an Interactive Algorithm to Develop Cyclical Rotational Schedules for Nursing Personnel", *INFOR*, vol.14, no.1, pp.53-70, 1976.

[77] **Systems Development Services** *System documentation: Statistical Account MT51A* University of Ottawa, Ottawa, Canada, February, 1989.

[78] **Systems Development Services** *System documentation: TTAB* University of Ottawa, Ottawa, Canada, September, 1988.

[79] **Timmreck, E.M.** *Scheduler - A Program that Uses Instructor and Student Preferences to Form Timetables*, Computer Science Technical Report #3, University of Wisconsin, Wisconsin, 1967.

[80] **Tripathy, A.** "A Lagrangean Relaxation Approach to Course Timetabling", *Journal of Research of National Bureau of Standards*, No.84, pp.489-506, 1979.

[81] **Welsh, D.J.A.** and **Powell, M.B.** "An Upper Bound for the Chromatic Number of a Graph and Its Application to Timetabling Problems", *Computer Journal*, vol.10, pp.85-86, 1967-68.

[82] **de Werra, D.** "An Introduction to Timetabling", *European Journal of Operational Research*, vol 19, pp.151-162, 1985.

[83] **de Werra, D.** "On a Particular Conference Scheduling Problem", *INFOR*, vol.13, no.3, pp.308-315, 1975.

[84] **de Werra, D.** "Some Comments on a Note about Timetabling", *INFOR*, vol.16, no.1, pp.90-92, 1978.

[85] **White, D.J.** "A Note Of Faculty Timetabling", *Operations Research Quarterly*, vol 26, pp.875-878, 1975.

[86] **White, G.M.** and **Chan, P.W.** "Towards the Construction of Optimal Examination Schedules", *INFOR*, vol.17, no.3, pp.219-229, 1979.

[87] **White, G.M.** and **Haddad, M.** "An Heuristic Method for Optimizing Examination Schedules which have Day and Night Courses", *Computers and Education*, vol.7, no.4, pp.235-238, 1983.

[88] **White, G.M.** and **Wong, S.K.S.** "Interactive Timetabling in Universities", *Computers and Education*, vol.12, no.4, pp.521-529, 1988.

[89] **Williams, M.R.** "Heuristic Procedures-(If they work-leave them alone)", *Software Practice and Experience*, vol.4, pp.237-240, 1974.

[90] **Winters, W.K.** "A Scheduling Algorithm for a Computer Assisted Registration System", *Communication of the ACM*, vol.17, pp.166-171, 1971.

[91] **Wood, D.C.** "A System for Computing University Examination Timetables", *Computer Journal*, vol.11, pp.41-47, 1968.

[92] **Wood, D.C.** "A Technique for Colouring a Graph Applicable to Large Scale Timetabling Problems", *Computer Journal*, vol.12, pp.317-319, 1969.

[93] **Ulph, A.** "An Extension of a Result in School Timetabling", *INFOR*, vol.15, no.2, pp.255-257, 1977.

[94] **Universal Algorithms Incorporated** *SCHEDULE25 Product Overview* and *SCHEDULE25 Theoretical Background*, Brochures distributed by Universal Algorithms Incorporated, 519 SW Park Ave, Portland, OR 97205, 1989.

[95] **Wolfston, J.H.** *Testing Claims of Room Scheduling Optimization Against the Schedule25 Standard*, Report distributed at the 34th Annual College and Administrative Computer Users Conference, Boston, Mass., April, 1989.

[96] **Yule, A.P.** "Extensions to the Heuristic Algorithm for University Timetables", *Computer Journal*, vol.10, pp.360-364, 1968.

7.3. REFERENCES ON LOGIC PROGRAMMING

[1] **Amble, T.** *Logic Programming and Knowledge Engineering* Reading, Massachusetts: Addison-Wesley Publishing Co., pp.4-8, 12-27, 268-269, 1987.

[2] **Boyer, R.S.** and **Moore, J.** "The Sharing of Structure in Theorem Proving Programs", *Machine Intelligence 2*, (B. Meltzer and D. Michie, ed.), Scotland: Edinburgh University Press, pp.101-116, 1972.

[3] **van Caneghem, M.** and **Warren, H.D.** ed. *Logic Programming and Its Applications* Norwood, New Jersey: Ablex Publishing Corp., pp.vii-x, 1986.

[4] **Clark, K.L.** and **Tarnlund, S.A.** ed. *Logic Programming* London: Academic Press, pp.xiii-xvii, 1982.

[5] **Copi, I.M.** *Symbolic Logic, 5th ed.* New York: Macmillan Publishing Co., Inc., pp.213-258, 290-332, 1979.

[6] **Dincbas, M., Simonis, H.** and **van Hentenryck, P.** *Extending Equation Solving and Constraint Handling in Logic Programming* Technical Report, E.C.R.C. (European Computer-Industry Research Center), 1987.

[7] **Dincbas, M., Simonis, H.** and **van Hentenryck, P.** *Solving Large Combinatorial Problems in Logic Programming*, Technical Report TR-LP-21, E.C.R.C. (European Computer-Industry Research Center), June, 1987.

[8] **Fuchi, K.** "Aiming for knowledge information processing systems", *Logic Programming and Its Applications*, ed. by M.van Caneghem and H.D. Warren, pp.279-305, 1986.

[9] **Green, C.C.** "Theorem Proving by Resolution as a Basis for Question-answering Systems", *Machine Intelligence 4*, (B. Meltzer and D. Michie, ed.), American Elsevier, pp.183-205, 1969.

[10] **van Hentenryck, P.** and **Dincbas, M.** *Forward Checking in Logic Programming*, Technical Report TR-LP-16, E.C.R.C. (European Computer-Industry Research Center), November, 1986.

[11] **Kowalski, R.** "Logic as a computer language", *Logic Programming*, ed. by K.L. Clark and S.A. Tarnlund, pp.3-16, 1982.

[12] **Kowalski, R.** "Predicate Logic as Programming Language", *Information Processing 74*, pp.569-574, 1974.

[13] **Loveland, D.W.** "Mechanical Theorem Proving by Model Elimination", *Journal of ACM*, vol.15, no.2, pp.236-251, 1968.

[14] **Maier, D.M.** and **Warren, D.S.** *Computing with Logic - logic programming with PROLOG* Menlo Park, California: The Benjamin/Cummings Publishing Co., Inc., pp.xi-xiv, 78-105, 269-341, 347-376, 1988.

[15] **Narain, S.** "MYCIN: The expert system and its implementation in Loglisp", *Logic Programming and Its Applications*, ed. by M.van Caneghem and H.D. Warren, pp.161-173, 1986.

[16] **Robinson, J.A.** "A Machine-oriented Logic Based on the Resolution Principle", *Journal of ACM*, vol.12, no.1, pp.23-41, 1965.

[17] **Simonis, H.** and **Dincbas, M.** *Using an Extended Prolog for Digital Circuit Design*, Technical Report TR-LP-22, E.C.R.C. (European Computer-Industry Research Center), August, 1987.

[18] **Sterling, L.** and **Shapiro, E.** *The Art of Prolog*, Cambridge, MA: MIT Press, 1986.

[19] **Sussman, G.J.** and **Steele, G.L.** "CONSTRAINTS-A Language for Expressing Almost-Hierarchical Descriptions", *Artificial Intelligence*, vol.14, no.1, pp.1-30, 1980.

[20] **Veitch, R.G.** and **Wilson, J.C.** *An Introduction to PROLOG Programming with WPROLOG - University of Waterloo PROLOG Interpreter for VM/SP CMS*, Waterloo, Canada: WATCOM Products Inc., 1986.

[21] **Werren, D.H.D., Pereira, L.M.,** and **Pereira, F.** "PROLOG -- The language and its implementation compared with LISP", *Proceedings of ACM Symposium on Artificial Intelligence and Programming Languages*, issued as *SIGPLAN Notices*, vol.12, no.8, pp.109-115, 1977.

[22] **Dincbas, M.** and **van Hentenryck, P.** "Extended Unification Algorithms for the Integration of Functional Programming into Logic Programming", *Journal of Logic Programming*, vol.4, pp.199-227, September, 1987.

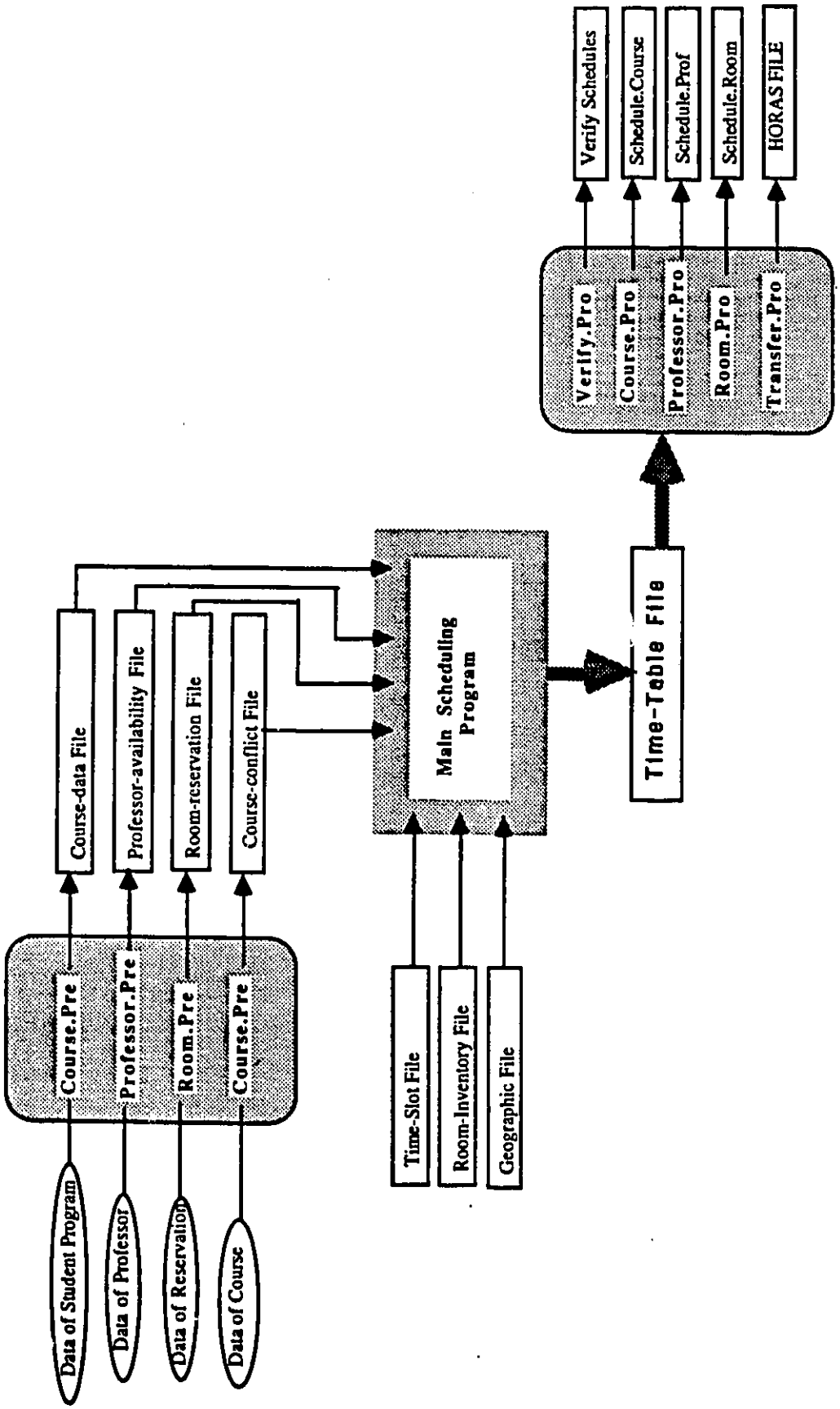
[23] **van Hentenryck, P.** "A Constraint Approach to mastermind in Logic Programming", *ACM Sigart*, vol.103, January 1988.

[24] **van Hentenryck, P.** *Constraint Satisfaction in Logic Programming*, Cambridge, Mass.: The MIT Press, 1989.

[25] **Lassez, K, McAloon, K.,** and **Yap, R.** "Constraint Logic Programming and Option Trading", *IEEE Expert*, vol.2, no.3, pp.42-50, Fall, 1987.

APPENDICES

APPENDIX A. SYSTEM STRUCTURE OF CTS



APPENDIX B. TIME-SLOT SYSTEM USED AT UNIVERSITY OF OTTAWA

HORAIRE / TIMETABLE

COURS DU JOUR / DAY COURSES (Périodes 3 x 1h Periods)

		LUN/ MON.	MAR/ TUES.	MER/ WED.	JEU/ THUR.	VEN/ FRI.
	8:30	1	4	2	5	3
	9:30	2	5	3	1	4
A	10:30	3	1	4	2	5
_._._	11:30	11	14	12	15	13
	12:30	12	15	13	11	14
B	13:30	13	11	14	12	15
_._._	14:30	21	24	22	25	23
	15:30	22	25	23	21	24
C	16:30	23	21	24	22	25
_._._	17:30					

COURS DU JOUR / DAY COURSES (Périodes 2 x 1½h Periods)

		LUN/ MON.	MAR/ TUES.	MER/ WED.	JEU/ THUR.	VEN/ FRI.
	8:30	101	104	102	105	103
A	10:00	102	105	103	101	104
_._._._	11:30	111	114	112	115	113
B	13:00	112	115	113	111	114
_._._._	14:30	121	123	122	124	125
C	16:00	122	124	121	123	125
_._._._	17:30	131	133	131	133	/////
	19:00					

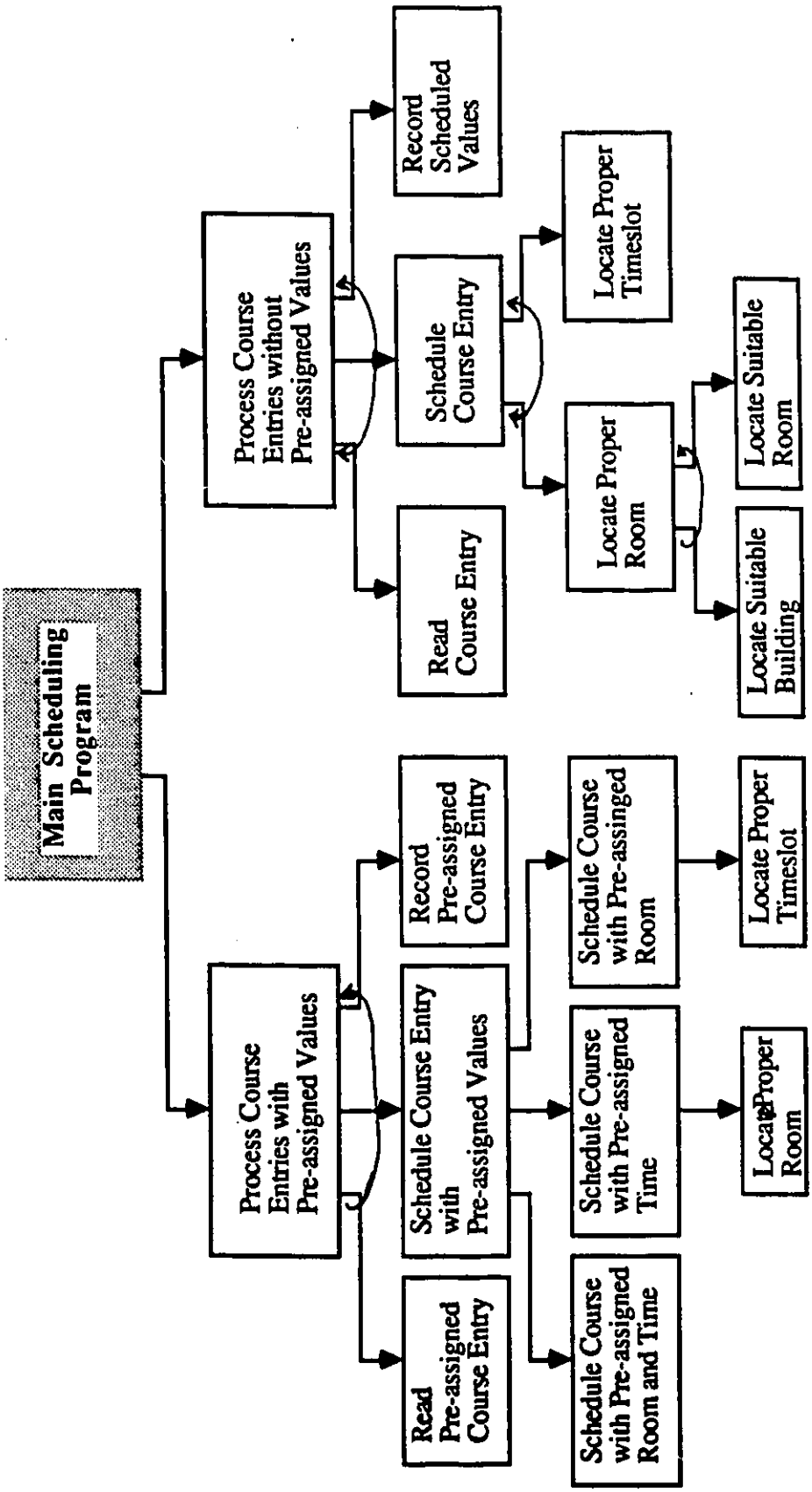
COURS DU SOIR / NIGHT COURSES (Périodes 1 x 3h Periods)

_._._._	19:00	331	334	332	335	333
D						
_._._._	22:00					
	23:00					

APPENDIX B.2. NEW TIME-SLOT SYSTEM USED BY CTS

The New Standard Slot System								
			Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Day Time	Morning	08:30-10:00	11	21	31	41	51	61
		10:00-11:30	12	22	32	42	52	..
	Noon	11:30-13:00	13	23	33	43	53	..
		13:00-14:30	14	24	34	44	54	..
	Afternoon	14:30-16:00	15	25	35	45	55	..
		16:00-17:30	16	26	36	46	56	..
		17:30-19:00	17	27	37	47	57	..
Night Time	Evening	19:00-20:30	18	28	38	48	58	..
		20:30-22:00	19	29	39	49	59	..

APPENDIX C. FLOWCHART OF THE MAIN SCHEDULING ALGORITHM



APPENDIX D. INPUT FILE STRUCTURE

D.1 Room Inventory File

This file contains the information about all the classrooms.

a) Room Code;
b) Room Type;
c) Furniture Type;
d) Seating Type;
e) Number of Seats;
f) Reserved Timeblocks.

D.2 Professor File

This file contains information about professor availability.

a) Professor Name;
b) Department;
c) Unavailable Timeblocks.

D.3 Student Program

This file contains information of student programs.

a) List of Required Courses;
b) List of Recommended Courses.

D.4 Course File

This file contains all the information of courses waiting to be scheduled. If a course has labs, different course codes should be given to these labs.

a) Course Code;
b) Faculty;
c) Department;
d) Semester of the course;
e) Course Format;
f) limitation to the Teaching Time;
g) Perferred Building;
h) Requirement to the Classroom;
i) Enrollment;
j) Preassigned Values;
k) Teacher(s).