INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa • University of Ottawa

Master's thesis

An interactive decision support system for course assignment

By Paul Mercier

Supervised by
Professor Jean-Michel Thizy

Systems Science Program
University of Ottawa

April 1999

Abstract

This thesis proposes to assist administrators and instructors in their task of assigning instructors to courses while respecting resources, goals and preferences of the members of the academic unit and its students.

The analysis of the different methods and models used to assign courses to instructors leads to the development of a new mathematical model that retains most of the advantages of previous ones while being simple and efficient enough to be implemented in a real administrative context. The new model is used to design a flexible, interactive and customizable decision support software to allocate courses to faculty members and maintain the derived selections. This decision support software relies on a database system with modelling and optimization capabilities.

The methodology is applied to the course assignment problem faced each year by the Faculty of Administration of the University of Ottawa and reports actual course assignment exercises.

Author: Paul Mercier
Supervisor: Jean-Michel Thizy,
Associate Professor,
Faculty of Administration

Acknowledgements

To my thesis supervisor, Professor Thizy, I owe my deepest appreciation and I am grateful for his support, understanding and assistance throughout the completion of my thesis. His excellent critical judgement and constructive suggestions were a constant source of motivation. Thank you!

I would also like to thank Michael Carter at the University of Toronto for sending me an impressive bibliography, many entries of which appear in the references.

To my wife Gail, I wish to express my gratitude for her extraordinary patience and fine support while I was away, working on my thesis. I owe you all my love.

Finally, to my parents, instructors and colleagues, thank you for giving me the relentless desire to learn and share my knowledge with others.

Paul Mercier

Table of contents

List of Figures

List of Tables

Introduction

The administration of an academic unit has become increasingly complex and cumbersome, especially in recent years, due in large part to increased public accountability in a context of human and financial resource rationalization. Under these conditions, the problem of the assignment of instructors to courses has become a crucial component of the annual planning exercise in which the abilities and preferences of faculty members must be balanced against the objectives of the institution, while taking into consideration administrative policies, procedures and resource constraints. Not only does it entail both technical and individual characteristics, but numerous factors can also affect the size and complexity of a practical assignment problem. For example, changes in student enrolment, faculty retirements or resignations, turnover in part-time teaching staff and new appointments of instructors are all factors that increase the difficulty of assigning courses to instructors.

The allocation of instructors to courses is usually done manually by the dean, chairman or supervisor of an academic unit in consultation with all the instructors and, in large institutions, this problem may be quite difficult, even for an experienced planner. Faced with changes of market conditions and budgetary restrictions, academic units have come to rely on increasingly sophisticated information systems. Therefore, a flexible decision support system can prove to be very useful for assigning courses to the teaching personnel interactively and attaining a compromise between the preferences of instructors and the goals and policies of the academic unit itself.

Objectives of the thesis

The objective of the thesis is to design a Decision Support System which will assist administrators and instructors in their task of assigning instructors to courses while respecting resources, goals and preferences of the members of the academic unit and its students.

The objective is attained in a threefold logical sequence. First, Chapter 1 reviews the different methods and models used to assign courses to instructors. These models trade computational complexity for added planning flexibility, in particular one, based on teaching plans, would entail so many variables that its direct implementation is out of reach. Chapter 2 is therefore devoted to the development of a new mathematical model that will retain most of the advantages of former ones while being simple and efficient enough to be implemented in a real administrative context. Finally, in Chapter 3, the new model will be used to design a flexible, interactive and customizable decision support software to allocate courses to faculty members and maintain the derived selections. This decision support software will rely on a relational database system with modelling and optimization capabilities.

The decision model of the thesis will be restricted to the specific operations related to the allocation of courses to instructors, delegating the assignment of sections and rooms to posterior phases such as the repetitive and labour-intensive task of assigning individual students to courses as well as the distribution of courses to rooms and time slots. The outcome of the thesis is the design of an allocation procedure supported by a two-tiered decision system:

- a flexible spreadsheet suite used by individuals or groups of instructors to evaluate and adjust each instructor's teaching schedule in consultation with the administration;
- a formal model encompassing the entire academic unit to manage common interests and promote longer-term, master planning.

The methodology is applied to the course assignment problem faced each year by the Faculty of

Administration. In this context, such a decision support system will also serve as a policy integrator and a conflict resolution mechanism.

Nomenclature

The thesis adopts an unusual format to number equations which should facilitate its reading while emphasizing the common nature of the models reviewed. Next to a traditional reference based on sequential numbering, each equation is also referenced by a letter recalling its role. A brief list is given below:

Symbol	Constraints related to
a	Teaching load (a_i)
b	Number of sections of course (b_j)
p	Selection of one teaching plan (y_{ik})
x	variables linking instructor to course sections (x_{ij})
y	variables linking instructor to teaching plans (y_{ik})

The keep general terminology, the terms *instructor*, or occasionally *teacher*, will be used to designate anyone eligible to teach a course. *Professor* will designate a full-time member of the academic unit being considered (which may comprise staff members not having a rank of professor).

Chapter 1 Background and review of the literature

Academic planning problems have been studied and documented by a large number of operational researchers since the early 60s and several relevant articles have appeared in the literature. The different types of problems include the allocation of faculty members to courses and administrative tasks, the assignment of courses and sections to rooms and time slots and the enrolment of individual students to their course sections. In the last three decades, the solution of these complex planning problems have been enhanced by techniques stemming from mathematical programming, graph theory, computer-based graphics or numerical analysis programs using exact algorithms and heuristic procedures. Although there exists a rich variety of approaches, techniques and algorithms developed for the scheduling and timetabling problem, there are no universally accepted solution methods. Conversely, successful implementations need not be supported by one general, all-inclusive academic scheduling model. The problems can be of such sizes and the number of computations so large that no known method can guarantee optimality of practical solutions or even the existence of one solution within a predictable and reasonable time limit.

Previous research has generally focussed on several partial aspects of the academic planning problem, such as class and teacher timetabling, student scheduling and classroom/time assignment [Kang and White, 1992]. Because of its importance, the scheduling of courses to rooms and time slots in order to minimize students' conflicts or optimize their preferences has received most attention and, in fact, several commercial scheduling packages have been developed for that purpose. The methods commonly used for this type of planning traditionally include linear and non-linear programming, goal programming, logic programming and aggregation and disaggregation techniques.

Lesser attention has been paid to the problem of assigning teachers to courses while taking into consideration the instructors' abilities, preferences and departmental or institutional goals. Nonetheless, the use of mathematical programming models and techniques for faculty/course assignment has been proposed by several authors. The majority of these presents mathematical models consisting of different aspects of problem solving for course assignment such as the maximization of preferences, taking into account the course assignment and teaching load constraints. The results generally suggest that there is considerable merit in following a quantitative optimization approach to such a problem, although some models in the literature may be difficult to implement. Some of these traditional models are presented below.

1.1. Traditional linear programming models for course assignment

The basic structure of the traditional model for faculty/course assignment is that of a transportation problem [Hitchcock, 1941]. This conventional type of problem consists of I origins that contain various amount of a commodity that must be shipped to J destinations to meet the demand requirement. There is a unit shipping cost associated with the shipping of the commodity from the origin to its destination. The problem is to find the shipping pattern between origins and destinations that satisfies all the requirements and minimizes the total shipping cost.

Specifically applied to the faculty/course assignment problem, Origin i (each instructor) contains an amount a_i (the teaching load) and Destination j (each courses) has a requirement of amount b_j (the number of courses). The shipping cost of the traditional transportation problem corresponds to the preference rating of each instructor to teach a given course. Of course, in this case, the problem is to maximize the total aggregate preference while ensuring that all the constraints are met.

Among the first researchers to apply this simple model to the faculty/course assignment

problem, Andrew and Collins [1971] described the first model to be successfully implemented in a real academic context. The purpose of their model was to optimize the assignment of faculty members to courses by maximizing the weighted sum of preference and effectiveness ratings. If we assume that J courses will be offered in the next academic year, when I full-time instructors will be available and if we let:

- q_{ij} be the preference rating of Instructor i for Course j,
- e_{ij} be the effectiveness of Instructor i to teach Course j as determined by the department chairman, the dean or an evaluation committee of students and/or other faculty members,
- a_i be the number of courses to be taught by Instructor i in the coming year (also called "teaching load"),
- b_j be the number of sections of Course j to be offered this coming year,
- x_{ij} be the number of sections of Course j assigned to Instructor i this coming year,
- w be a weighting factor chosen between 0 and 1

then the course allocation problem can be formulated as a linear programming problem as follows:

$$\text{Max} \sum_{i=1}^{I} \sum_{j=1}^{J} x_{ij} \, [w q_{ij} + (1-w) e_{ij}]$$

s.t.

$$(1a) \quad \sum_{j=1}^{J} x_{ij} \le a_i \qquad \text{for all } i = 1, \ldots, I$$

$$(2b) \quad \sum_{i=1}^{I} x_{ij} = b_j \qquad \text{for all } j = 1, \ldots, J$$

$$(3x) \quad x_{ij} \ge 0 \qquad \text{for all } i = 1, \ldots, I \text{ and } j = 1, \ldots, J$$

This single-period model seeks to maximize the aggregate preference (q_{ij}) and effectiveness (e_{ij}) of instructors while satisfying two basic sets of constraints. The first set of constraints ensures that each instructor *i* teaches no more than his or her course load a_i. The second set of constraints ensures that the planned and required number of sections (b_j) of each course *j* are taught. The structural form of this model and its data characteristics guarantee that the extreme points of the feasible region will provide an integer solution (i.e. instructors are not assigned fractional number of sections). Andrew and Collins developed a computer program which was tested and implemented by the Electrical Engineering department at the University of Minnesota for several years.

In 1976, Breslaw described a model similar to what Andrew and Collins proposed a few years earlier. In fact, from Breslaw's model, we obtain the same formulation if we redefine x_{ij} as the percentage of course *j* assigned to Instructor *i* and assign w=1 and b_j=1 (each course is taught only once). Both of these models are in the form of a standard transportation problem for which several efficient and fast algorithms are available and which, by the nature of the model, provide integer solutions. Breslaw's model has also been tested in a real academic setting, but not implemented in a real case scenario.

1.2. Integer programming formulations

Based on Andrew and Collins' work, Tillett [1975] also proposed a variant of the linear programming model for the assignment of teachers to courses. To allow more flexibility, his model introduced the possibility that the preference of a teacher for a particular course may be influenced by the number of sections of that course she/he is assigned. Moreover, his model included a set of constraints to control the acceptable maximum number of preparations (different courses) assigned to each instructor. He defined the additional variables:

- x_{ijk} as a 0/1 decision variable having a value 1 if Instructor *i* is assigned to teach *k* sections of Course *j* this coming year, a value of 0 otherwise,
- K_{ij} as equal to a_i or b_j, whichever is less,
- u_i as the maximum number of preparations (different sections) to be assigned to Instructor *i* in the coming year,

His model is formulated below:

$$\max \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{k_{ij}} x_{ijk} \, [kwq_{ij} + k(1-w)e_{ij}]$$

s.t.

(4a) $$\sum_{j=1}^{J} \sum_{k=1}^{k_{ij}} kx_{ijk} = a_i \qquad \text{for all } i = 1, ..., I$$

(5b) $$\sum_{i=1}^{I} \sum_{k=1}^{k_{ij}} kx_{ijk} = b_j \qquad \text{for all } j = 1, ..., J$$

(6) $$\sum_{k=1}^{k_{ij}} x_{ijk} \le 1 \qquad \text{for all } i = 1, ..., I \text{ and } j = 1, ..., J$$

(7) $$\sum_{j=1}^{J} \sum_{k=1}^{k_{ij}} x_{ijk} \le u_i \qquad \text{for all } i = 1, ..., I$$

(8x) $x_{ijk} \ge 0$ integer for all $i = 1, ..., I$, $j = 1, ..., J$ and $k = 1, ..., K_{ij}$

Constraints (4a) and (5b) have basically the same interpretation as in Andrews and Collins' model. Constraints (6) ensure that all sections of all courses are allocated, while Constraints (7) stipulate that no instructor should be assigned more that his specified number of preparations. Contrary to previous ones, this model features Constraints (8x) to require that some variables assume only values of zero or one in order to prevent the splitting of the number

of sections assigned. Although Tillett provided evidence of empirical testing of his model, he did not report any actual implementation.

The approach taken by Shih and Sullivan [1977] was somewhat more extensive. They proposed a multi-period scheduling model formulated as a zero-one programming problem which is formed by a two-stage scheduling process; first, an integer programming model assigns courses to faculty, then a second model assigns the course-instructor combinations to time slots. In the first stage, the objective is to assign courses to instructors in such a fashion as to maximize the total faculty course preference over a period of several terms. This first stage is very similar to the models presented above. Once courses have been assigned to each instructor, the second stage of the scheduling process assigns the combination course-instructor to time slots so that the total faculty preference with respect to time slots is also maximized. Their model can also be expanded to meet special requirements, such as specific requests from faculty members or administrators, or the sequential arrangement of specific courses over several periods. One interesting aspect of their formulation is the inclusion of constraints to ensure the "quality" of undergraduate and graduate education. These constraints specify that a certain minimum percentage of graduate courses must be taught by instructors with a doctorate and that those instructors must also teach a certain number of undergraduate courses. The small example they provide, which has only nine courses to assign to five different instructors, requires a total of ninety variables and over forty constraints. The model has apparently not been tested on a more realistic academic environment with a larger number of courses and instructors.

1.3. Goal programming

Goal programming is a method that seeks to find an optimal solution for mathematical models that are composed solely of goals [Charnes and Cooper, 1977]. Compared to traditional models, the main advantage of using a goal programming approach for the allocation of courses to instructors lies in the fact that, by definition, it incorporates multiple goals and objectives that

can be underachieved because of conflicts between them; most traditional models contain strict requirements that must be fully satisfied before reaching a feasible solution. The other reason for the interest in goal programming is that conventional mathematical programming methods such as those previously described do not consistently provide an acceptable solution nor always reflect practical tradeoffs offered by the actual problem.

To eliminate, or at least alleviate some of the limitations of previous programming methods, some authors used goal programming techniques in the assignment of faculty members to courses. Harwood and Lawless [1975] were apparently first to use goal programming to ensure least possible deviation from the inherent conflicting goals of the allocation of courses. Their multiple-criteria model first determines the assignment of faculty members to courses, and then courses to time slots in a two-stage optimization procedure, somewhat like Shih and Sullivan [1977]. In the formulation of their model, individual faculty preferences are only taken into consideration indirectly by using various organizational and personal preference goals; the ones considered important by faculty respondents included:

- the observance of expected teaching loads,
- the granting of individualized requests,
- the assignment of the least possible number of preparations,
- a shortest possible teaching day,
- a maximum assignment of two sections of the same course,
- the least possible number of teaching days in a week and, finally,
- the assignment of the least number of night classes.

Although their model encompasses many different dynamic aspects of course assignment often ignored and difficult to capture, it also requires a very large amount of information and processing. Furthermore, the small application they described, where only seven faculty members were assigned twelve different courses, required more than 1,000 constraints and almost 200 variables. The approach chosen, the modelling characteristics and the solution

method hinder practical solutions. Consequently, the model was not implemented in a real academic setting.

Several years later, Schniederjans and Kim [1987] also proposed to use a goal programming formulation to assign faculty members to courses. Their model was simpler than Harwood and Lawless', incorporating faculty course teaching preferences. If we define the following additional variables and constants:

- z_{ij} equals one (1) if Course *j* is assigned to Instructor *i*, zero (0) otherwise;
- d_j^{b-} as the negative deviation from the requirements of Course *j* (assigning fewer sections than desired)
- d_j^{b+} as the positive deviation from the requirements of Course *j* (assigning more sections than desired)
- d_i^{a-} as the negative deviation from the teaching load of Instructor *i* (assigning fewer sections than the desired teaching load)
- d_i^{a+} as the positive deviation from the teaching load of Instructor *i* (assigning more sections than the desired teaching load)
- d_h^{r-} as the negative deviation from the number of course section offerings at the same faculty assigned h^{th} preference rank
- d_h^{r+} as the positive deviation from the number of course section offerings at the same faculty assigned h^{th} preference rank
- H as the total number of ranks used by faculty to define their course preferences
- r_h as the number of course sections permitted within the same h^{th} ranking
- w_h as the ranked weight given by faculty on the preference to be assigned to teach a specific course

then the model described by Schniederjans and Kim is as follows:

$$\text{Min} \quad P_1 \sum_{i=1}^{I} [d_i^{a-} + d_i^{a+}] + P_2 \sum_{j=1}^{J} [d_j^{b-} + d_j^{b+}] + P_3 \sum_{h=1}^{H} w_h [d_h^{r-} + d_h^{r+}]$$

s.t.

$$\text{(9a)} \quad \sum_{j=1}^{J} z_{ij} + d_i^{a-} - d_i^{a+} = a_i \qquad \text{for all } i = 1, ..., I$$

$$\text{(10b)} \quad \sum_{i=1}^{I} z_{ij} + d_j^{b-} - d_j^{b+} = b_j \qquad \text{for all } j = 1, ..., J$$

$$\text{(11)} \quad \sum_{i=1}^{I} \sum_{j=1}^{J} z_{ij} + d_h^{r-} - d_h^{r+} = r_h \qquad \text{for all } h = 1, ..., H$$

Equations (9a) take into consideration the teaching load in order to constrain the assignment of courses to each Instructor *i*. Similarly, Equations (10b) seek to assign the exact number of sections of Course *j* that should be taught in the next planning period. The last set of goal constraints build the faculty course teaching preferences into the model. By structuring the goal constraints contained in Equations (11) in order of the ranked preference and by using a weighting approach in the objective function, the model attempts to confine the assignment of a given course to the faculty member having the highest preference. Finally, the objective function is formulated to ensure the least possible deviation from the goals, ie. to seek the assignment of the exact teaching load a_i to each instructor (Priority P1) and the exact number of courses b_j (Priority P2). The third priority (P3) represents the faculty course teaching assignment preference. For the assignment of 20 different courses to 12 instructors, the formulation of Schniederjans and Kim consisted of 87 decision variables and 36 goal constraints, which is relatively small compared to previous models. It was easily solved using the modified Simplex algorithm and results show deviations only for the last priority (P3). Given its relative simplicity, it was successfully implemented at the University of Nebraska.

Following Shih and Sullivan, Badri [1996] proposed a two-stage zero-one programming

model, but used a goal programming approach to first assign faculty to courses and then assign faculty-course combinations to time blocks. The first stage used a model adapted from Schniederjans and Kim's [1987]. The output of the first stage was then fed through the second stage where a similar procedure maximized the assignment of course-faculty to time slots with respect to instructor's preference.

1.4. Network formulations

Network flow problems represent another class of mathematical programming methods used for the allocation of courses to instructors. In this model, each course is represented as a node and so is each instructor. The *possible* assignment of courses to each instructor is represented by arcs between each nodes. The traditional transportation problem previously described can be represented by this simple network formulation known as a pure network. More precisely, the classical transportation problem can be generalized by flows of maximal values in a network. However, a network structure can incorporate different features that extend the classical transportation problem to a more elaborate problem, for example by including arc flow restrictions to achieve various objectives.

Dyer and Mulvey [1976, 1977] employed a network formulation approach to the problem. With regards to the definition of the decision variables and the types of constraints being considered, the model structure resembles that of Andrew and Collins [1971]. Although Dyer and Mulvey's model is somewhat more complex, all their constraints can be represented as a pure network model.

Figure 1, excerpted from [Dyer and Mulvey, 1977], presents their network formulation. The authors describe it as "the best compromise among computational considerations, information requirements and user needs."

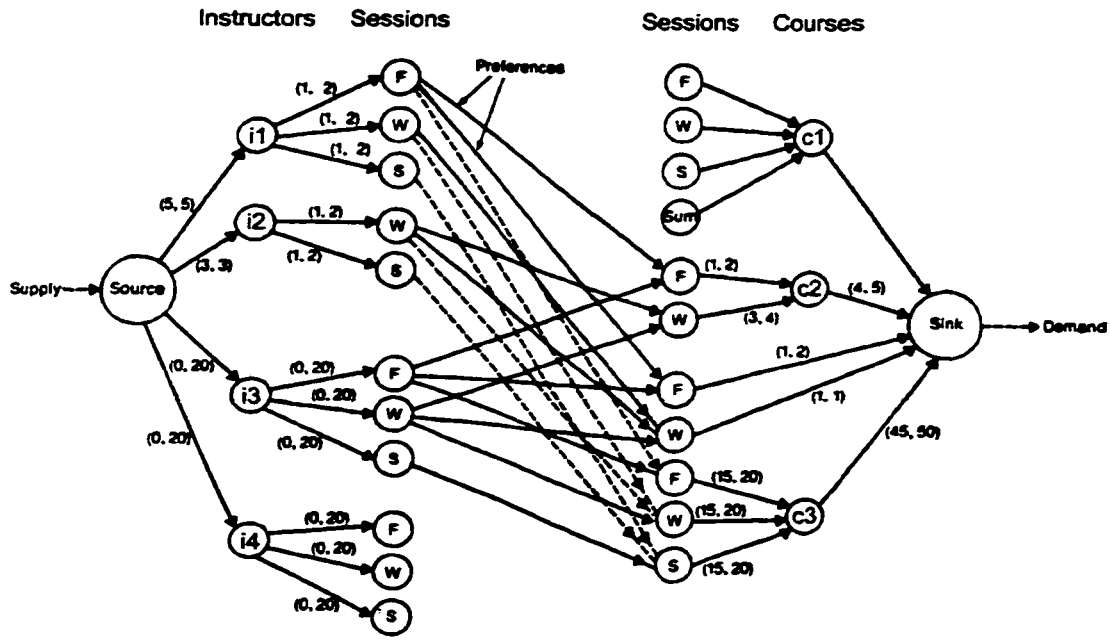


Figure 1 - Network representation of the faculty assignment problem

On the left side of this figure, there is one circle for each faculty member (in this example there are only four instructors labelled i1 to i4). The circle labelled F, W, S stand for Fall, Winter and Spring/Summer sessions respectively. The arrows represent the availability of each teacher for each session and their respective workload. On the right side of Figure 1, there are circles corresponding to courses (labelled C1 to C3) and arrows representing the possible offering of each course for each session. Several network modifications are possible, to allow, for example, the modelling of restrictions on the number of courses or to force certain assignments. The teaching and staffing goals of Schniederjans and Kim's model [1987] are achieved through the abstraction of *Superprof* and *Supercourse*. A superprof represents a fictitious individual who has the ability to teach any course offered, but at a very high cost (or low preference) to the objective function. Similarly, all the faculty members can teach the supercourse, but at a very high cost. Originally introduced to ease computational considerations, these two features are also very useful to immediately detect any difficulty in solving the problem. It can also be used to diagnose any imbalance in the demand or supply of instructors and to plan the hiring of

additional teaching resources or offering of additional sections. The analysis of the courses assigned to the superprof can quickly determine what teaching abilities are required from additional lecturers. For a small application like this, the modelling characteristics of a network formulation and its pictorial representation can be easily understood by nontechnical users. This aspect of a network design constitute a real advantage over other formulations [Dyer and Mulvey, 1977]. This model has been successfully implemented and used for several semesters and has helped both short and longer range planning at UCLA.

More recently, Fontaine and Truchon [1989] also used a transportation network to model the assignment of courses to faculty. They simplified the formulation of Dyer and Mulvey by defining a node for each couple instructor-semester, which can be considered as the sources. Similarly, a node is also defined for each course-semester combinations. An arc joins each instructor in a particular semester to a course if the instructor can teach this course in this semester. Just as Dyer and Mulvey's, the problem is then to find the maximal values satisfying the demands while respecting the availability at the source. In order to simplify the whole process, Fontaine and Truchon developed a spreadsheet program to set up and modify the input data. Then, a BASIC program reads the data, formulates the network and finds an optimal solution.

In spite of all its advantages, a network formulation cannot explicitly include other considerations such as those reviewed by some authors; for example, a requirement that Course A be offered only if Course B is not, that a faculty member teach at least a given number of undergraduate courses, or that an instructor teaches no more than a total of two courses if she/he teaches Course A. Moreover, adding instructors and courses to the problem dramatically increases the complexity of the network formulation and its solution process.

1.5. The use of plans as decision units

Unlike previously proposed models, McClure and Wells [1984] suggested that course allocation should be based on instructors' schedules, i.e., individual teaching plans for the planning period. In this way, interaction effects between course utilities could be taken into account within the model, which had never been considered before. For example, one instructor may have a high preference for teaching a certain graduate course and a high preference for giving a particular seminar, but a low preference for giving both at the same time because of the amount of preparation involved or for other reasons. Denote:

- π_{ik} as the preference of Instructor i for his or her k^{th} plan,
- β_{ijk} as the number of sections of Course j that Instructor i proposes to teach in the k^{th} plan that she or he submits;
- y_{ik} as a decision variable valued at 1 if the k^{th} plan of Instructor i is retained, 0 otherwise,
- K_i as the number of plans submitted by Instructor i,

then the model proposed by McClure and Wells is as follows:

$$\text{Max} \quad \sum_{i=1}^{I} \sum_{k=1}^{K_i} \pi_{ik} y_{ik}$$

s.t.

$$\text{(12b)} \quad \sum_{i=1}^{I} \sum_{k=1}^{K_i} \beta_{ijk} y_{ik} = b_j \qquad \text{for all } j = 1, ..., J$$

$$\text{(13p)} \quad \sum_{k=1}^{K_i} y_{ik} = 1 \qquad \text{for all } i = 1, ..., I$$

$$\text{(14y)} \quad y_{ik} = 0 \text{ or } 1 \qquad \text{for all } i = 1, ..., I \text{ and } k = 1, ..., K_i$$

Again, this model seeks to maximize the overall preference of the instructors for their chosen schedules (or plans) while satisfying two sets of constraints. Constraints (12b) ensure that all the plans assigned include the required number of sections of each course, while Constraints (13p) require that each instructor be assigned exactly one teaching plan. In this model, the decision variables of Constraints (14y) represent the assignment of a complete schedule rather than individual courses as proposed in previous models. Furthermore, teaching load and course preparation constraints need not be explicitly expressed in the model since they are implied in the use of the generated plans. In their paper, McClure and Wells tried to prove the superiority of their model by comparing the computational and system performance with previous models from Andrew and Collins [1971] and Tillett [1975]. However, their model has not been implemented in a real academic setting.

The following table collects the features of the different assignment models proposed in the literature (in chronological order).

Criteria	Models proposed									
	AC	T	HL	BR	DM	SS	MW	SK	FT	BA
0/1 programming		✔	✔			✔		✔		✔
Integer programming		✔	✔	✔	✔		✔		✔	
Goal programming			✔					✔		✔
Network formulation					✔				✔	
Plans (schedules of courses)							✔			
Includes course timetable						✔				✔
Explicit preference treatment	✔	✔		✔	✔	✔	✔	✔	✔	✔
Number of preparations		✔	✔				✔			
Course prof interaction effect							✔			
Multi period model					✔	✔			✔	
Interactive data input interface				✔					✔	
Implemented	✔				✔			✔		

Table 1 - Models proposed in literature

Legend:

AC: Andrew and Collins [1971]

T: Tillett [1975]

HL: Harwood and Lawless [1975]

BR: Breslaw [1976]

DM: Dyer and Mulvey [1976 and 1977]

SS: Shih and Sullivan [1977]

MW: McClure and Wells [1984]

SK: Schniederjans and Kim [1987]

FT: Fontaine and Truchon [1989]

BA: Badri [1996]

1.6. Mathematical Programs vs. Expert Systems

The previous review focussed on mathematical programs with emphasis on their computational tractability, since excessive execution time remains a major barriers to immediate and practical application [Tillet, 1975, Harwood and Lawless, 1975]. Research by McClure and Wells [1984] also demonstrated how prohibitive the number of constraints and variables were for some of the models proposed (i.e. Tillett [1975], Breslaw [1976] and Shih and Sullivan [1977]). From this perspective, the low number of constraints and variables in McClure and Wells' formulation represents a major advantage. In particular, only small and well structured mathematical formulations requiring integer variables can be solved in reasonable computing time [Bardi, 1996]. However, the rapid growth of computing capacity in recent years has greatly facilitated the solution of problems that were once practically impossible to solve repetitively.

Owing partly to this progress, mathematical models are now compared using other criteria, as noted in recent literature [Schniederjans and Kim, 1987 and Bardi, 1996]. For example, the time necessary to develop and maintain a given model could affect its usefulness and initial purpose when updated incrementally. Important information requirements can also pose a serious limitation to the effective and practical use of a model. This is one of the major drawbacks of the model of Harwood and Lawless [1975]; it requires so much information and processing that it may be very difficult to implement and use effectively. According to Schniederjans and Kim [1987], the model proposed by Harwood and Lawless might require more time and effort in data collection and processing than the benefits gained by using the model. Implementation is greatly facilitated if no new additional information on course offering, teaching load requirements and preference ratings need to be collected. This information is often collected on a regular basis by administrators for planning purposes.

Past experience has shown that the application of mathematical models may not satisfy the administration of the academic unit and its faculty members [Breslaw, 1976, Stimson and Thompson, 1974]. Many models proposed in the literature do not present enough modelling

flexibility or interactivity to receive widespread acceptance in most academic units. For example, few models consider the need to include the maximum acceptable number of preparations in their formulation [Tillet, 1975, Harwood and Lawless, 1975 and McClure and Wells, 1984]. Generally, as can be expected, the preference decreases as the number of sections proposed increases. Their research indicates that this secondary aspect is considered crucial to the acceptability of the overall assignment of courses to instructors.

Such flexibility and attention to contingent factors is at the centre of a comparison between mathematical programs and expert systems [Dhar and Ranganathan, 1990], which has become a benchmark for decision support systems devoted to course assignment. Dhar and Ranganathan compare an integer programming (IP) approach with an expert system hinging on an assumption-based truth maintenance system (TMS). They analyse their performance, reliability, ability to express complex composite preferences and the ease with which they allow revisions of academic plans or underlying assumptions.

The expert system incorporates a heuristic rule-based problem solver and a truth maintenance system. In contrast with mathematical programs, this system does not utilize an objective function to guide the solution toward an optimum. Rather, the problem solver iterates through several local decisions most preferred at any one point while the TMS ensures that the solution is contradiction-free and well-founded.

The integer program comprises some 700 binary variables and 300 constraints. Using existing data on faculty preferences and course requirements, the authors note the following differences between the two approaches:

- The integer program does not always yield a feasible solution. Even if feasible solution exists, the computing time required to verify its optimality is highly unpredictable and very sensitive to variations of constraint coefficients. On the other hand, the expert system provides at least a partial solution, in a time varying usually from 1 to 2 hours.

- Since the IP formulation contains a single objective, i.e., to assign courses to instructors based on their preferences/desires, it cannot address other goals that the expert usually takes into consideration. Unfortunately, using a multi-objective function or expressing the objective function as a constraint may not be manageable. In contrast, the rules of the expert system can include knowledge about multiple objectives, such as the maximum number of preparations or large class sizes.
- The coefficients of the IP formulation aggregate compiled knowledge; in particular, the objective function summarizes the information about the course-instructor assignment preferences. The aggregation often makes it difficult to express complex rules or preferences. The IP also lacks the flexibility to add or modify often contradictory objectives or to face unusual situations. These characteristics can be handled by the expert system, enabling it to take into account several types of preferences, such as those delineated in the previous sections of this chapter.
- The aggregation of compiled knowledge performed by the IP formulation hinders attempts to justify the solution. On the other hand, the Expert System can trace the choices made by the TMS and therefore help the decision-makers to support their decisions or to adjust the solution to changes in their assumptions.
- Whereas the IP formulation either provides a solution fast or none at all (even after several hours of execution time), the expert system generates a solution invariably. Even if the latter contains holes in the schedule, it can be easily filled to form a practical plan. Furthermore, the expert system provides a history of the choices made, a convenient source of information to explain why a complete solution was not found.
- The IP formulation lacks support for making revision to the initial plans. For example, if a change suddenly occurs and an instructor goes on leave due to sickness, the IP solution does not contain information about how to reassign the course, contrarily to what the expert system can suggest.

1.7. The thesis

The thesis proposes to combine the models of Andrew and Collins with that of McClure and Wells. These two models feature contrasted formulation complexities and computational speed. In the latter, representing all feasible plans would entail so many variables that a direct implementation would be out of reach. Since McClure and Wells' model can be viewed as a column generation of Andrew & Collins', the latter will actually be used as a device generating additional solutions on demand. On the other hand, the former model can also be viewed as a summary knowledge base, ushering advantages found in expert systems and thereby endowing mathematical programming with additional applicability. Thus, the formulation proposed reflects a general aspiration for effective decision-making found in virtually all current decision support systems. The expected benefit of this approach encompass the following features:

1. User orientation: An effective model for course assignment must reflect instructors' preferences accurately. Otherwise, instructors will by-pass the academic planners and reach mutually agreeable schedules in subgroups, which may conflict with the central planners' objectives (*de-facto course allocation*). One step in this direction was made by taking course interaction into account [McClure and Wells, 1984], assigning complete schedules instead of individual courses. More generally, the preference for a given teaching plan may not be expressed easily, or it may change during the planning period, calling for an interactive planning system.

2. Interactivity: Often, the optimal solution of the model is not satisfactory for a variety of reasons: it is contentious, socially unacceptable or simply inadequate. To be successful, all the persons involved by the course-to-faculty assignment must participate in the design, development, implementation and use of a decision modelling tool. In the past, some authors attempted to alleviate this problem by actively involving the administration during the entire planning phase [Breslaw, 1975, Dyer and Mulvey ,1977]. However, the models developed still lack important aspects of course assignment and don't have the flexibility and interactivity

necessary to receive widespread acceptance.

3. *Ease-of-use:* Few of the models described in the literature have an easy conversational user interface that a person with little knowledge of mathematical models or computers could effectively use. Administrative staff does not always have the time and the necessary expertise to learn and develop mathematical models containing several hundreds of variables and constraints. Their goal is to assign courses to instructors and create a suitable schedule for everyone involved. Not only can an interactive user interface provide convenient ways of exploring alternative assignment, adapt the data or browse the solutions provided, but it also enables decision-makers to guide the solution procedure by practically preforming some of the exploratory work with insight not possessed by the decision support system.

Chapter 2 Problem Definition and Methodology

As most managerial exercises, allocating instructors to courses entails two concerns. First are immediate operational requirements: are all course sections staffed, does every instructor have a proper teaching load that satisfies administrative and union rules? Only then arise long-term issues: are the courses relevant, are they properly taught, do they form a coherent set, do instructor have time for professional advancement? One can roughly say that the operational concerns are met by constraints of the programs, whereas longer-term targets are reached via an objective function. The mathematical programs proposed in this chapter answer operational requirements squarely. Models of increasing complexity and realism are presented in this chapter, starting with a basic program that represents the utility under a very general form.

2.1. A non-linear integer transportation model

Recall the notation:

- J courses will be offered in the next academic year;
- I full-time instructors will be available to teach;
- α_j is the number of credits assigned to each section of Course j;
- a_i is the number of credits to be taught by Instructor i in the coming year (teaching load),
- b_j is the number of sections of Course j to be offered this coming year,
- x_{ij} is the number of sections of Course j assigned to Instructor i this coming year (the decision variables).

The course allocation problem can be formulated as the following linear programming problem:

P: Max $f(\{x_{ij}\}_{ij})$

s.t.

(15a) $\sum_{j=1}^{J} \alpha_j x_{ij} = a_i$ for all $i = 1, ..., I$

(16b) $\sum_{i=1}^{I} x_{ij} = b_j$ for all $j = 1, ..., J$

(17x) $x_{ij} \geq 0$ integer for all $i = 1, ..., I$ and $j = 1, ..., J$

Constraints (15a)-(17x), similar to those of Andrew and Collins [1971], address situations where the number of credits may vary between courses and further require that the number of sections of a given course be integer, even if the instructors' loads a_i or courses requirements b_j represent fractions of such courses. These constraints may for example be satisfied by assigning two full-year courses plus one semester-long courses to an instructor, or one full-year course plus three semester-long courses, or five semester-long courses, but an assignment of four semester-long courses, plus one half of a full-year course violates the same constraints.

The function f denotes the academic unit's preference for a given assignment. Its precise characterization is elusive and depends on many parameters. Yet, some tangible characteristics of the utility function f can be determined. Obvious components of this utility function include the qualification of the instructor for delivering such a course, the quality of his/her teaching, as measured in past deliveries of this same course and the instructor's preference for teaching the course. The function is typically not separable, i.e. is not made up by utilities related to the assignment of one course to one instructor. There are many reasons for that, as noted by McClure and Wells [1984]. On the administrative side, the quality of a program is not simply made up by the sum of satisfactions for enrolling in each individual course. This is particularly true of curricula integrating team work, practicums or internships. On the professorial side, an

instructor's utility for a particular course is heavily influenced by other courses in her/his own schedule. Hence, neither the academic unit nor individual instructors may easily assign a preference for a course without knowing the other courses in her or his schedule. Whereas the overall utility derived by all beneficiaries of the allocation cannot be expressed analytically, there are practical ways of assessing the desirability of a complete course roster. This suggests a pointwise exploration of the utility space, which calls for techniques of optimization where the objective function must be assessed by separate calculations at each point through a combination of calculations and human evaluation.

It is therefore proposed to resort to an interactive exploration of this utility function. Rather than devising an algorithmic approach to evaluating this utility, the thesis advocates reliance on the current procedure adopted to design the course roster of an upcoming academic year. Although this procedure is a highly interactive and collegial process supported by the academic unit's subsections and program directors, it can also be analysed as an approximation of the utility function. This search derives robustness from the academic practice of requesting that each full-time instructor submits a plan of teaching (i.e., a schedule) for the following academic year to the administration.

2.2. Alternative model based on plans

The actual solution of the preceding program will be achieved by considering an alternative formulation focussing on instructors' plans [McClure and Wells, 1984]. After recalling some notation, the alternative formulation is presented. Recall the definitions:

- β_{ijk} as the number of sections of Course j that Instructor i proposes to teach in the k^{th} plan that she or he submits;
- y_{ik} as a decision variable valued at 1 if the k^{th} plan of Instructor i is retained, 0 otherwise,

- K_i as the number of plans submitted by Instructor *i*.

then the model proposed is:

$$\Pi: \quad \text{Max} \ \phi\left(\{y_{ijk}\}_{ijk}\right)$$

s.t.

$$(18b) \quad \sum_{\substack{j=1 \\ k=1}}^{J,K_i} \beta_{ijk}\, y_{ik} = b_j \qquad \text{for all } j = 1, ..., J$$

$$(19p) \quad \sum_{k=1}^{K_i} y_{ik} = 1 \qquad \text{for all } i = 1, ..., I$$

$$(20y) \quad y_{ik} = 0 \text{ or } 1 \qquad \text{for all } i = 1, ..., I \text{ and } k = 1, ..., K_i,$$

Constraints (18b) ensure that all the plans assigned include the required number of sections of each course *j*, while Constraints (19p) and (20y) require that each instructor *i* be assigned exactly one teaching plan.

From a modelling viewpoint, the use of decision variables that represent teaching plans is realistic and manageable. Although it entails a very large number of variables, the model has a degree of flexibility and efficiency not found in the integer transportation model (P). The impact of numerous incidental features and global characteristics can be taken into consideration, such as the interaction or interdependence effect between courses or the maximum acceptable number of preparations for any instructor. This model can therefore avoid additional constraints which would have increased the complexity and size of the formulation. Moreover, the formation of each plan relies on the instructor's assessment of the relative importance of such factor. The use of plans has also beneficial information processing advantages, for example:

- the academic unit may suggest alternative plans to instructors by retrieving plans from

previous years, stored in a database;

- the submission of multiple plans encourages formative planning previously hindered by the complexity of a manual treatment.

In the model proposed, the utility function ϕ denotes the academic unit's preference for an overall course offering. Its precise characterization is as elusive as that of model (P) and depends on many parameters such as the preferences and needs of the students, the administration of the academic unit and the instructors themselves. Its measure is quite complex as it draws on historical, organizational and practical elements, each interacting with others in the final decision. Yet, the formation of plans will be encouraged by assuming that each plan k proposed by Instructor i possesses an intrinsic utility π_{ik}. Therefore, in accordance with an academic spirit of collegiality, it will be assumed that the overall utility of a course roster can be approximately evaluated as the sum of utilities of each instructor's plan retained, with the ensuing optimal function:

$$\text{Max} \quad \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \pi_{ik}\, y_{ik}$$

Although it may be difficult to assess the utility resulting from a given assignment, there are several methods for providing a measure of the coefficient π_{ik} associated with the k^{th} plan submitted by Instructor i. First, the coefficient could be evaluated by administrators or section coordinators of the academic unit. However, the establishment of such plans has contractual value and cannot be performed without the instructor's consent and input. According to McClure and Wells [1984], faculty members generally have little difficulty assigning utilities to the various schedules. In practice, the evaluation will be performed by some form of mutual recognition of administrative and professorial imperatives. The solution retained in the implementation is therefore for the instructor to initiate the assessment of the plans proposed, with safeguards against wide discrepancies between the instructor and the community's aspirations. Several sources of discrepancies are listed below:

- Plans may have a number of courses different from the instructor's assigned teaching load. This happens frequently when an instructor is assigned administrative duties at fairly short notice, such as with the outcome of an election. Note that voluntary teaching of courses above the nominal teaching load are treated separately.
- Plans may contain courses not scheduled for delivery in the following academic year.

The following model will show how such plans can be constructed and evaluated progressively as an overall course schedule is established for the academic unit.

2.3. A column generation model

Given that plans incorporate implicit planning factors, their generation is highly subjective and time consuming. Hence, time, storage space and planning rigour prevent the inclusion of all feasible teaching plans in the model (II). A practical course scheduling will need to be based on a small subset of such plans. To support it, a composite model will include both explicit plans and the integer transportation model (P) that contains every such plan implicitly in which the utility of the individual teaching assignments will be also simplified as:

$$\sum_{\substack{i=1 \\ j=1}}^{I,J} p_{ij}\, x_{ij}.$$

A discussion of the computer implementation will show that the coefficients p_{ij} are small compared to the coefficients π_{ik} . Hence, the column generation model including professorial plans is:

$$\textbf{P}\Pi\text{: max} \sum_{\substack{i=1 \\ j=1}}^{I,J} p_{ij}\, x_{ij} + \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \pi_{ik}\, y_{ik}$$

s.t.

$$\text{(21a)} \sum_{j=1}^{J} \alpha_j x_{ij} + \sum_{\substack{j=1 \\ k=1}}^{J,K_i} \alpha_j \beta_{ijk}\, y_{ik} = a_i \qquad \text{for all } i = 1, \ldots, I$$

$$\text{(22b)} \sum_{i=1}^{I} x_{ij} + \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \beta_{ijk}\, y_{ik} = b_j \quad \text{for all } j = 1, \ldots, J$$

$$\text{(23p)} \sum_{k=1}^{K_i} y_{ik} \leq 1 \qquad \text{for all } i = 1, \ldots, I$$

$$\text{(24x)}\ x_{ij} \geq 0 \text{ and integer} \qquad \text{for all } i = 1, \ldots, I \text{ and } j = 1, \ldots, J,$$

$$\text{(25y)}\ y_{ik} = 0 \text{ or } 1 \qquad \text{for all } i = 1, \ldots, I \text{ and } k = 1, \ldots, K_i,$$

Progressive column generation provides another advantage; constructing plans based on incumbent solutions may help instructors refine the content of their plans and their utilities, thus proposing plans better responding to the current overall objectives. In defence of the accuracy of this approximation method, note that when instructors propose plans, they tend to give it a utility, not solely based on their absolute preferences, but on the preferences for the schedule given what they perceive the final course assignment of the academic unit will be.

Constraints (23p) are now inequalities, allowing for some variables x_{ij} to form plans directly; attempting to enforce equality will be addressed in Section 2.8. The linear relaxation of this model could be solved by decomposition techniques presented at the end of the chapter. The next subsection reviews adaptations of the model to the current conditions faced by the Faculty of Administration.

2.4. Adapting the model to the Faculty of Administration

With over 70 regular instructors, the Faculty of Administration offers about 480 course sections per year. Annually, in consultation with the Registrar of the University, the Faculty selects its course offering, including the number and schedule of sections of each course. In practice, a given year's program tends to resemble the previous year's, both in the number and type of courses and times of the sections.

In some academic units and in the Faculty of Administration in particular, several courses are traditionally given by the same instructor every year for different reasons: particular faculty skills or knowledge and contractual agreements are good examples. The administration needs to integrate such cases seamlessly (for example, by assigning its own utility sparingly). Even after this predetermined and historic assignments, the residual problem can be very large and a manual treatment can prove to be time-consuming and sub-optimal.

In the last few years, external conditions and budgetary restrictions propelled a reorganization of the Faculty, then a suppression of courses. These changes were capped by an increased teaching load for full-time instructors. More demand is therefore placed on the academic information system and how course allocation should proceed and be assisted by a decision support system. Yet, the overall reduction of courses enables the Faculty to avoid the implementation of a decision support system that would include the allocation of courses or sections to time slots. Faculty policies and room restrictions limit the maximum number of students registered in any course section.

About a third of the courses offered by the Faculty are taught by part-time instructors on contract. Because the Canadian Capital region offers a large pool of qualified and talented instructors, chronic shortages due to short-term conflicts of schedules do not usually arise. Courses that cannot be filled by regular full-time faculty members are simply given to part-time instructors. Hence, Constraint (22b) can be relaxed to the one below.

$$(26b)\ \sum_{i=1}^{I} x_{ij} + \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \beta_{ijk}\, y_{ik} \le b_j \text{ for all } j = 1, \ldots, J$$

The relaxation can also be understood by viewing the slack variable (denoted x_{oj}) of Constraints (26b) as the potential assignment of the relevant course to a generic part-time instructor indexed by 0 , who is also the *superprof* of Dyer and Mulvey [1977].

To promote collegiality, the Faculty promotes team teaching of some courses. Team teaching generally involves pairs of instructors, although some courses have been taught by up to 5 regular instructors of the Faculty. Team-teaching poses an important challenge to academic planners, partly due to the fact that some courses are taught by teams regularly, others very occasionally. For the purposes of course assignment, the former may be regarded as two half-courses and the coefficient α_j is halved accordingly (team-teaching is particularly common in the graduate programmes that offer courses of 1.5 credits and thus half of such a course entails 0.75 credits!). However, an individual course assignment that is numerically accurate may prove infeasible if all members of the team are not assigned to the intended course section. In practice, many teams comprise part-time instructors to whom the particular co-assignment is customized. The lack of consistency in the definition of team assignments compels the model to put little emphasis on Constraints (26b), reporting every occurrence of its violation for manual treatment.

Given the availability of part-time instructors, professors of the Faculty tend to specialize their teaching. Therefore, a given course is usually taught by few regular professors. In the planning of course assignment, contention for a given course exists, but is fairly limited. This particularity will greatly enhance the column generation procedure.

2.5. Incidental requirements

A practical course allocation model typically considers other specific features of the course planning exercise that are not directly taken into account by the proposed model, but can be addressed interactively.

First, several sections of many courses are taught in different semesters or times of the day. Thus, an instructor may express preference for a specific section. However, the Faculty often specifies teaching schedule limitations and it routinely selects the setting of the course sections that each instructor can teach. For example, the Faculty discourages its professors from teaching more than 3 sections per semester and it often requires that full-time instructors teach at least one large section of a course per year. The Faculty also discourages instructors from teaching only in the evening and seeks to staff day-time sections with full-time instructors, thereby limiting the choice of sections available to each instructor. Finally, the Faculty may choose to control the number of compulsory (core) courses taught by each full-time instructor, or to balance instructors' loads in each semester. Some of these constraints were initially implemented, then removed as the variety of courses and modes of teaching increased so much as to require a case by case plan evaluation. In final analysis, it was felt unnecessary that the model should distinguish sections with a proviso to differentiate between semesters if the need arose.

At a more general level, the model proposed, as are most of the approaches previously discussed, is restricted to a single period exercise. In order to capture the different dynamic aspects of teaching assignment, some authors pinpoint the advantages of a multi-period model [Shih and Sullivan, 1977]. Research shows that faculty members are generally sensitive to the dependence between schedules in consecutive semesters [McClure and Wells, 1984]. Although some multi-period models have been proposed for the teaching assignment problem, none consider the preferential dependence between courses in a schedule over multiple terms or sessions; hence, course utilities are determined without considering the teaching schedules of adjacent terms. The applicability of such a multi-period model to the given context can be

questioned; it would require faculty members to assign utilities to courses over multiple time periods, but ignore course preferential dependence among periods. If the proposed model was to consider multiple periods, faculty members would assign utilities to schedules for each term and incorporate preferential dependence in their utility values. The proposed formulation could also be easily expanded into a multi-period format by indexing each variable for every semester in the planning period. However, a model which takes into account all the elements of interaction between periods is generally larger and more complex than the sum of all the individual models for the same planning periods. In fact, many of the dynamics of a multi-period model can be included in the analysis of an incremental addition of teaching plans.

2.6. Prioritizing the constraints

The presentation of model (PII) promoted the role of constraints, while emphasizing subjective aspects of the objective function. In particular, favouring individual instructors' preferences does not take into account collective policies and preferences. In reality, the implementation will hinge on a fine interplay between constraints and preferences. Relying on a hierarchical goal programming approach will provide insight into the relative priority of the constraints and give a theoretical basis for broadly adjusting individual preferences. For example, a tenet of an academic institution is that instructors should in no case be required to teach a course that they are not qualified to offer. Therefore, the set of courses that Instructor *i* can teach will be denoted as J_i, as an explicit recognition that the penalties incurred for violating this rule override any other combination of penalties. Note that, under normal assumptions, $J_i \supseteq \cup_i K_i$.

Teaching plans

An intended benefit of the support system is to favor instructors' plans. To do so, deviation variables d_i^-, similar to those introduced by Schniederjans and Kim [1987], modify Equation (23p) as:

$$(27p)\ \sum_{k=1}^{K_i} y_{ik} + d_i^- = 1 \qquad \text{for all } i = 1, \ldots, I$$

for a modified objective function:

$$\max \sum_{\substack{i=1 \\ j=1}}^{I,J_i} p_{ij}\, x_{ij} + \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \pi_{ik}\, y_{ik} - \sum_{i=1}^{I} P_i\, d_i^-$$

where P_i is the penalty of not meeting the i^{th} equality exactly. Reordering the terms of Equations (27p) yields:

$$d_i^- = 1 - \sum_{k=1}^{K_i} y_{ik}$$

which allows for the following substitution in the objective function:

$$\max \sum_{\substack{i=1 \\ j=1}}^{I,J_i} p_{ij}\, x_{ij} + \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \pi_{ik}\, y_{ik} - \sum_{i=1}^{I} P_i \left(1 - \sum_{k=1}^{K_i} y_{ik} \right)$$

or, through simple algebraic manipulations:

$$\max \sum_{\substack{i=1 \\ j=1}}^{I,J_i} p_{ij}\, x_{ij} + y_{ik} \sum_{i=1}^{I} \left(P_i + \sum_{k=1}^{K_i} \pi_{ik} \right) - \sum_{i=1}^{I} P_i$$

the last term of which is constant and can be omitted from optimization. Hence, Plan *k* has a very large objective coefficient $P_i + \pi_{ik}$. Our implementation does not distinguish between the penalties of instructors considered to be a single large value P.

A standard method to reduce the number of criteria of a linear program is to aggregate them via a weighted linear combination, where large weights are found to enforce the hierarchy of original objectives. Rewriting slightly the current objective as:

$$\max \sum_{\substack{i=1,I \\ j=1,J_i}} p_{ij}\, x_{ij} \;+ P \sum_{\substack{i=1,I \\ k=1,K_i}} (1 + \pi_{ik}/P)\; y_{ik}\,,$$

shows that P acts as a large weight. Accordingly, the implementation will strive to accommodate instructors' plans or, this failing, to utilize the variables x_{ij} for surrogate. As a recourse, a second level evaluates individual course assignments mostly as a deterrent against restrictive plans, which incites instructors to propose broadly acceptable options when submitting their plans. The coefficient $P + \pi_{ik}$, simply denoted π_{ik} in the sequel, will have a large value. This modification of the coefficients π_{ik} has an additional advantage. Since a very coarse and additive estimate $\sum_{\substack{i=1,I \\ j=1,J_i}} p_{ij}\, x_{ij}$ of the utility is adopted, it is important that the parameters p_{ij} be given small values compared to $\sum_{\substack{i=1,I \\ k=1,K_i}} \pi_{ik}\, y_{ik}$. Hence, the approximation by $\sum_{\substack{i=1,I \\ j=1,J_i}} p_{ij}\, x_{ij}$ can be considered to remain a lower estimate, as occur in successive approximation methods.

Teaching loads

A collective view of the fairness of the course assignment requires that all instructors be allocated their pre-assigned teaching load precisely. In spite of the Faculty of Administration's efforts to maintain an equitable allocation, it is sometimes impossible to assign enough course sections in order to meet an instructor's specified teaching load. Deviation variables d_i^- and d_i^+ modify Equation (21a) as:

$$(28a)\quad \sum_{j=1}^{J_i} \alpha_j x_{ij} + \sum_{\substack{j=1 \\ k=1}}^{J_i,K_i} \alpha_j \beta_{ijk}\, y_{ik} + d_i^- - d_i^+ = a_i \qquad \text{for all } i = 1, \ldots, I$$

for a modified objective function:

$$\max \sum_{\substack{i=1 \\ j=1}}^{I,J_i} p_{ij}\, x_{ij} \;+\; \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \pi_{ik}\, y_{ik} \;-\; \sum_{i=1}^{I} P_i \left(d_i^- + d_i^+ \right)$$

where P_i is the penalty of not meeting the k^{th} equality (28a) exactly. Given the collective wish for full load allocation, we can assume that the penalty P_i associated with the variable d_i^- is higher than the expressed preference of any individual instructor. Courses taught above the specified load are treated as separate contracts and not managed by the proposed system; therefore d_i^- are currently not allowed. A simple reordering of the terms of Equations (28a) yields:

$$d_i^- = a_i \;-\; \sum_{j=1}^{J_i} \alpha_j\, x_{ij} \;-\; \sum_{\substack{j=1 \\ k=1}}^{J_i,K_i} \alpha_j\, \beta_{ijk}\, y_{ik}$$

which allows for the following substitution in the objective function:

$$\max \sum_{\substack{i=1 \\ j=1}}^{I,J_i} p_{ij}\, x_{ij} \;+\; \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \pi_{ik}\, y_{ik} \;-\; \sum_{i=1}^{I} P_i \left(a_i \;-\; \alpha_j\, x_{ij} \;-\; \sum_{\substack{j=1 \\ k=1}}^{J_i,K_i} \alpha_j\, \beta_{ijk}\, y_{ik} \right)$$

or, through simple algebraic manipulations:

$$\max \sum_{\substack{i=1 \\ j=1}}^{I,J_i} \left(P_i\, \alpha_j \;+\; p_{ij} \right) x_{ij} \;+\; \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \left(\pi_{ik} \;+\; P_i \sum_{j=1}^{J_i} \alpha_j\, \beta_{ijk} \right) y_{ik} \;-\; \sum_{i=1}^{I} P_i\, a_i$$

the last term of which is constant and can be omitted from optimization. Uniform penalties $P_i = P$ are also applied. However, the value of P is now much smaller than that of the penalty associated with the satisfaction of Constraints (23p) as equalities.

Further coefficient manipulation may be necessary: in the implementation, the coefficients p_{ij} are generated from historical data and not updated by individual instructors each year. Not only are the coefficients evaluated very coarsely, but the additive estimate $\sum_{\substack{i=1,I \\ j=1,J_i}} p_{ij}\, x_{ij}$ introduces an additional skewedness. Under normal circumstances, the coefficients π_{ik} are sufficiently large to overshadow this effect. However, the influence of the terms p_{ij} may have a second-order effect upon two plans that have very close coefficients π_{ik}. To avoid interference, the value of $P_i\alpha_j + p_{ij}$ could be decreased so as to remain a lower approximation of the original function $f(p_{ij})$.

Integrality Constraints

The reliance of the objective function on user-defined parameters, the large number of variables and the need for interactive solution weigh against the customized implementation of an integer program. A solution will be sought by linear relaxation, i.e., relaxing Constraints (24x) and (25y) as:

(29x) $x_{ij} \geq 0$ for all $i = 1, ..., I$ and $j = 1, ..., J$,

(30y) $y_{ik} \geq 0$ for all $i = 1, ..., I$ and $k = 1, ..., K_i$,

Note that Equations (23p) and (30y) imply that $y_{ik} \leq 1$ for all $i = 1, ..., I$ and $k = 1, ..., K_i$. We will denote by $(\overline{PII})$ the relaxation of (PII). Simple data manipulations are proposed to obtain a closer approximation of Constraints (24x) and (25y).

Data Property 1

For any indices i, j and k such that $\alpha_j > a_i$, all feasible solutions of ($\overline{PII}$) satisfy $x_{ij} < 1$.

For any indices i, j and k such that $\beta_{ijk} > b_j$, or $\alpha_j\beta_{ijk} > a_i$, all feasible solutions of ($\overline{PII}$) satisfy $y_{ik} < 1$. □

A proof by contradiction is quite simple, but the property indicates that, in order to yield positive decision variables x_{ij} and $y_{ik} = 1$, the mathematical program should include only plans such that $\beta_{ijk} \leq b_j$ and $\alpha_j\beta_{ijk} \leq a_i$ and courses such that $\alpha_j \leq a_i$.

The generation of fractional solutions, i.e., violating Constraints (25y), generally hampers the validity of the solution. Occasionally, these factional variables may yield solutions that satisfy Constraints (21a)-(23p) of the program. For example, Plan *k* of Instructor *i* may contain two sections of Course 1 and 2 ($\beta_{ijk} = 2$ for j = 1 and 2). Selecting half of the plan ($y_{ik} = 0.5$) lets Instructor *i* teach one section of each course ($\beta_{ijk}\, y_{ik} = 1$ for j = 1 and 2). Hence, Constraints (25y) delineate only sufficient conditions for the establishment of an effective teaching assignment. Given the current facilities that the Faculty offers to team teaching, it is not possible to know in advance whether teaching part of a course section is feasible or not. In practice, most variable y_{ik} will assume values of 0 or 1.

The integrality requirements (24x) of the variables x_{ij} is also relaxed. This relaxation may hinder or help the relaxation of (25y). Continuing on the preceding example, if ($y_{ik} = 0.4$), the variables x_{ij} may complement the staffing ($\beta_{ijk}\, y_{ik} + x_{ij} = 1$ for j = 1 and 2). However, their objective value may be inaccurate and even though the resulting preference $p_{i1}\, x_{i1} + p_{i2}\, x_{i2} + .4\, \beta_{ijk}$ may not be realistic, the small magnitude of the coefficient π_{ik} ensures that the full plan *k* would have been chosen ($y_{ik} = 1$) if a conflict of sections had not arisen. The presence of variables x_{ij} may also introduce undesirable results: suppose that Plan 1 has (exactly or approximately) the same preference as Plan 2, but contains one fewer section than Plan 2 which itself matches Instructor *i*'s assigned teaching load. The addition of one variable x_{ij} will

promote Plan 1 unduly over Plan 2, in effect creating a new Plan 3 with appropriate load and higher preference. Even though such correction may be viewed as a warranted reaction to shortsighted proposal of Plan 1, the model attempts to prevent it by decreasing the preference of all plans that contain a number different from instructors' assigned teaching loads.

Thus far, the assignment model has not included any restriction on the amount of credits that Instructor *i* may include in Plan *k* for Course *j*. This reflects the large flexibility that has been offered at the Faculty of Administration. Recall that in Section 2.4, examples were offered where one course was co-taught by 5 professors. Theoretically, each of these could have included $\beta_{ijk} = .2$ in their respective plans. However, this situation is still exceptional. If a course can be taught by *q* instructors, it is possible to rescale α_j as $\alpha_j' = \alpha_j / q$ and correspondingly rescale β_{ijk} as $\beta_{ijk'} = q\ \beta_{ijk}$ and b_j as $b_j' = q\ b_j$. Note that in this transformation, the transportation variable x_{ij} is rescaled as $x_{ij}' = q\ x_{ij}$. A general simplification is therefore to allow only for integer values of β_{ijk}. With this assumption, the data of the integer program can be further simplified in the hope of avoiding fractional teaching assignments

Data Property 2

Suppose all values of β_{ijk} are integer. Then, for any feasible solution (x_{ij} , y_{ik}) of (PII), (x_{ij} , y_{ik}) of (PII) is also a feasible solution of a problem (PII') with right-hand sides $\lfloor b_j \rfloor$. □

The proof by contradiction is also quite simple. Of course, the property does not hold for the linear relaxation of the problem (PII) which may then yield unnecessarily fractional teaching assignments. The property points to the care with which the course requirements $\lfloor b_j \rfloor$ should be defined and the need for special treatment of fractional course assignments.

Even with the simple data properties proposed, a fractional solution may not immediately yield a feasible teaching. Suppose for example that the Faculty intends to offer two sections of Course 1. Instructor *1* puts very a high priority on a plan including one section of Course 1 while Instructor *2* assigns a lower priority on a plan including two sections of Course 1. It is likely that

the plan of Instructor 1 and half of the plan of Instructor *2* will be retained without any immediate remedy. Any method to obtain a feasible assignment, using fractional values of the variables x_{ij} and y_{ik}, needs to rely on global properties on the fractional solution.

2.7. A small example

To illustrate the benefit of the column generation model, a small example is constructed in which two professors present an array of plans shown in Table 3. Table 2 displays the solution obtained through Model (P), using a linear preference where each coefficient p_{ij} is displayed in the right upper most corner of each cell. A blank cell means that a professor is not prepared to teach the given course. A number (including zero) indicates, among courses envisaged by the professor, how many sections have been allocated by the model (P). In Table 2, Marie and Jean would teach all sections of the same course. On the other hand, the column generation model II selects some plans (indicated by ** in Table 3). The plans proposed by Marie and Jean offer a variety of course teachings respecting their preferences shown in the last column of Table 3.

Course / Instructor	ADM1000	ADM2000	ADM3000	Teaching Load
Jean	0 (1)		2 (2)	2
Marie	0 (2)	2 (2)		2
Number of sections required	1	2	2	

Table 2 - Small example with Model (P)

Course / Instructor/Plan	ADM1000	ADM2000	ADM3000	Preference
Jean - Plan 1 **	1		1	8
Jean - Plan 2			2	3
Marie - Plan 1	2			9
Marie - Plan 2 **		2		5
Number of sections required	1	2	2	

Table 3 - Small example with Model (II)

2.8. Model decomposition

The Faculty has long discontinued its departmental structure in favour of 10 areas. In 1993, it opted to be restructured along 3 sections. Nearly every instructor belongs to one section and, at the end of each academic year, forwards a course plan to the coordinator of her or his section as well as to the dean of the Faculty. Course planning, preparation and delivery is supervised by the academic vice-dean. However, most courses are still earmarked by one of the former 10 areas of the Faculty. In practice, nearly all the sections of a given course are taught by instructors of the same section, as shown in Table 4 below. Hence, areas tend to be viewed as sub-groups of each section, to the extent that Faculty databases assign each instructor to an area.

	1995-1996	1996-1997	1997-1998	1998-1999
Total number of sections taught	507	438	541	594
Total number of sections taught (with instructor recorded)	505	425	536	508
Number of course sections unrelated to instructor's area (9) (excluding general courses)	27 (5.3%)	27 (6.4%)	32 (6.0%)	40 (7.9%)
Number of course sections unrelated to instructor's administrative section (3) (excluding general courses)	22 (4.4%)	19 (4.5%)	18 (3.4%)	26 (5.1%)
Total number of General courses (with instructor recorded)	25 (4.9%)	24 (5.5%)	38 (7.0%)	57 (9.6%)

Table 4 - Number of course sections

Note: - includes overload courses and contractual instructors assignments

This table shows the number of course sections unrelated to instructors' areas and the number of course sections unrelated to instructor's Faculty section in each of the past four academic years. These numbers remain small percentages of the total number of course sections offered by the Faculty. Hence, areas or sections remain convenient groupings for the planning of course assignment. In the following paragraphs, the feasibility of performing the assignment within each of the units, i.e, a system decomposition, is reviewed. We assume that a decomposition by area is simply a finer rendering of a decomposition by section. Since the term *section* is often in conflict with that of course section, it is simpler to use the term *area*, keeping an option of using a coarser decomposition with the sections of the Faculty.

In the decomposition, we consider that each instructor belong to an area ℓ, which appears as a superscript of the corresponding variables x_{ij}^{ℓ} , y_{ik}^{ℓ} and related coefficients p_{ij}^{ℓ} , π_{ik}^{ℓ}, β_{ijk}^{ℓ} and a_i^{ℓ}. The L areas would comprise a group currently called *General*. The model could therefore be re-written as:

PIIA: $\text{Max} \sum_{\ell=1}^{L} \{ \sum_{i=1}^{I,J} p_{ij}^{\ell} x_{ij}^{\ell} + \sum_{i=1}^{I,K_i} \pi_{ik}^{\ell} y_{ik}^{\ell} \}$

s.t.

(31a) $\sum_{j=1}^{J} \alpha_j x_{ij}^{\ell} + \sum_{j=1}^{J,K_i} \alpha_j \beta_{ijk}^{\ell} y_{ik}^{\ell} \le a_i^{\ell}$ for all $i = 1, ..., I$ and $\ell = 1, ..., L$

(32b) $\sum_{\ell=1}^{L} (\sum_{i=1}^{I} x_{ij}^{\ell} + \sum_{i=1}^{I,K_i} \beta_{ijk}^{\ell} y_{ik}^{\ell}) \le b_j$ for all $j = 1, ..., J$

(33p) $\sum_{k=1}^{K_i} y_{ik}^{\ell} \le 1$ for all $i = 1, ..., I$ and $\ell = 1, ..., L$

(34x) $x_{ij}^{\ell} \ge 0$ and integer for all $i = 1, ..., I$, $j = 1, ..., J$ and $\ell = 1, ..., L$

(35y) $y_{ik}^{\ell} = 0$ or 1 for all $i = 1, ..., I$, $k = 1, ..., K_i$ and $\ell = 1, ..., L$

This representation shows that Equations (31a), (33p), (34x) and (35y) can be separated into groups (31a.ℓ), (33p.ℓ), (34x.ℓ) and (35y.ℓ), essentially replicating in their original forms the variables and coefficients related to Area ℓ. On the other hand, Equation (22b) and the objective function involve all variables. However, it was noted above that many of the equations (22b) involve only variables of one area ℓ, which can therefore be denoted as:

(36b$^{\ell}$) $\sum_{i=1}^{I} x_{ij}^{\ell} + \sum_{i=1}^{I,K_i} \beta_{ijk}^{\ell} y_{ik}^{\ell} \le b_j^{\ell}$ for all $j = 1, ..., J$

whereas some constraints (22b) involving several areas would remain. This forms a common optimization structure in which L independent linear programs, represented in their constraint matrix form $A^{\ell} x^{\ell} \le b^{\ell}$, are related in an overall linear program:

$$\text{Max} \sum_{\ell=1}^{L} c^{\ell} x^{\ell}$$

$$A^{\ell} x^{\ell} \le b^{\ell} \qquad \text{for } \ell = 1, ..., L$$

$$(37) \quad \sum_{\ell=1}^{L} d^{\ell} x^{\ell} \le q.$$

where d^{ℓ} and q denote the additional requirements of Constraints (22b). Standard solution methods entail a relaxation of Constraints (37) with dual multipliers u , as:

$$\text{Max} \sum_{\ell=1}^{L} (c^{\ell} - u\, d^{\ell})\, x^{\ell}$$

$$A^{\ell} x^{\ell} \le b^{\ell} \qquad \text{for } \ell = 1, ..., L$$

and separate solution of each of the L subproblem as a course assignment model for one area and its associated courses. Using the original objective function of model (PII) presented in Section 2.3, the detailed objective function becomes:

$$\text{Max} \sum_{i=1}^{I} \Big(\sum_{j=1}^{J} p_{ij}^{\ell} - \sum_{j \in J_o} u_j \Big)\, x_{ij}^{\ell} + \sum_{\substack{i=1,K_i \\ k=1}} \Big(\pi_{ik}^{\ell} - \sum_{j \in J_o} u_j\, \beta_{ijk}^{\ell} \Big)\, y_{ik}^{\ell}$$

which correspond to modifying some of its coefficients p_{ij}^{ℓ} to become $p_{ij}^{\ell} - u_j$ and π_{ik}^{ℓ} to become $\pi_{ik}^{\ell} - u_j\, \beta_{ijk}^{\ell}$, for $j \in J_i$, the set of courses taught by several areas.

Various methods are proposed to update the dual multipliers u_j, with various speed of convergence. Problem decomposition corresponds closely to the structure of the Faculty and related course assignment process. If each iteration corresponded to an administrative decision of the areas, the dual multipliers could be evaluated via the preceding changes of objective

coefficients. However, care should be taken that the same values of multipliers are used by each area to avoid unequal treatment of some assignments by each area.

The following chapter compares two strategies that have been concurrently adopted by the planners of the Faculty:

- Solve the general Problem P, reducing it by assignments well known to satisfy the students' needs, administrative requirements and colleagues' expectations. Solve the reduced problem through interactive negotiations aiming at finding at optimal course assignment.

- Separate Problem P into subproblems P^{ℓ} for $\ell = 1, ..., L$ each containing an appropriate subset of variables and constraints, which are delegated to each section of the Faculty or even disaggregated to the level of sub-sections, each representing a distinct teaching area.

The decomposition proposed bears mostly on dual methods. For very large problems, another decomposition of each subproblem P^{ℓ}, or the entire problem P, is possible. Given one subproblem P^{ℓ}, suppose every variable y^{ℓ}_{ik} has been given a value. Then the remaining problem becomes an integer transportation program that can be solved much faster than the larger problem P^{ℓ}. Its linear relaxation can be solved much faster than the linear relaxation of Problem P^{ℓ}. This could either lead to an integrated algorithm based on primal decomposition or to heuristic approaches that would first assign plans, then elect to resort to assign sections individually.

Chapter 3 Model implementation

Several implementations of the model were undertaken as the administrative practices of the Faculty evolved. Such a straining exercise was mostly motivated by the imperative goal of adapting the implementation to the actual needs and current practice of the Faculty. It is hoped that the final implementation truly meets the original objective. It embraces:

- the current allocation procedures used by academics to support the ensuing decisions. For example, under typical academic policies, each full-time instructor is requested to submit a plan of teaching (i.e., a schedule) for the following academic year to the administration;
- the current information collected about past course allocations. For example, the Faculty of Administration maintains a database of such allocations with relevant information about the past course delivery (number of students enrolled, schedules, etc.)
- the allocation process itself, given the structure of the academic unit and accepted practices.

To facilitate decentralized, interactive planning, the support system is designed around two contrasting media. Spreadsheet planners provided to the heads of the sections (or some of their subgroups) enhance immediate ease-of-use, what-if analysis and quick response time. At the other end, an integrated Faculty planner uses database management capabilities to verify technical aspects of the plans and, if necessary, reconcile the sections' proposals.

Although the development of a database system took the largest part of the work related to the thesis, it was actually preceded by two tasks , i.e. the design of a spreadsheet model and a model generated in the mathematical modelling language AMPL.

3.1. Spreadsheet submodels

One objective of the new support system is to promote cooperation between the sections of the Faculty, both for research and for teaching. However, relatively few courses are taught by instructors of different sections, with some exceptions that tend to recur yearly. From an operational viewpoint, this insularity allows areas to establish relatively independent subplans, as was presented in Section 2.8.

A series of smaller models was therefore implemented on spreadsheets as a flexible tool for interactive joint examination by coordinator and instructor, prior to formal submission of plans to the Dean of the Faculty. The data set for each area is reduced to the courses typically taught by its members. Manual adjustments must be made for courses outside of this partial list (Faculty-wide adjustments are also required for instructors teaching outside the Faculty). Figure 2 displays part of the input and output tables corresponding to the transportation submodel P.

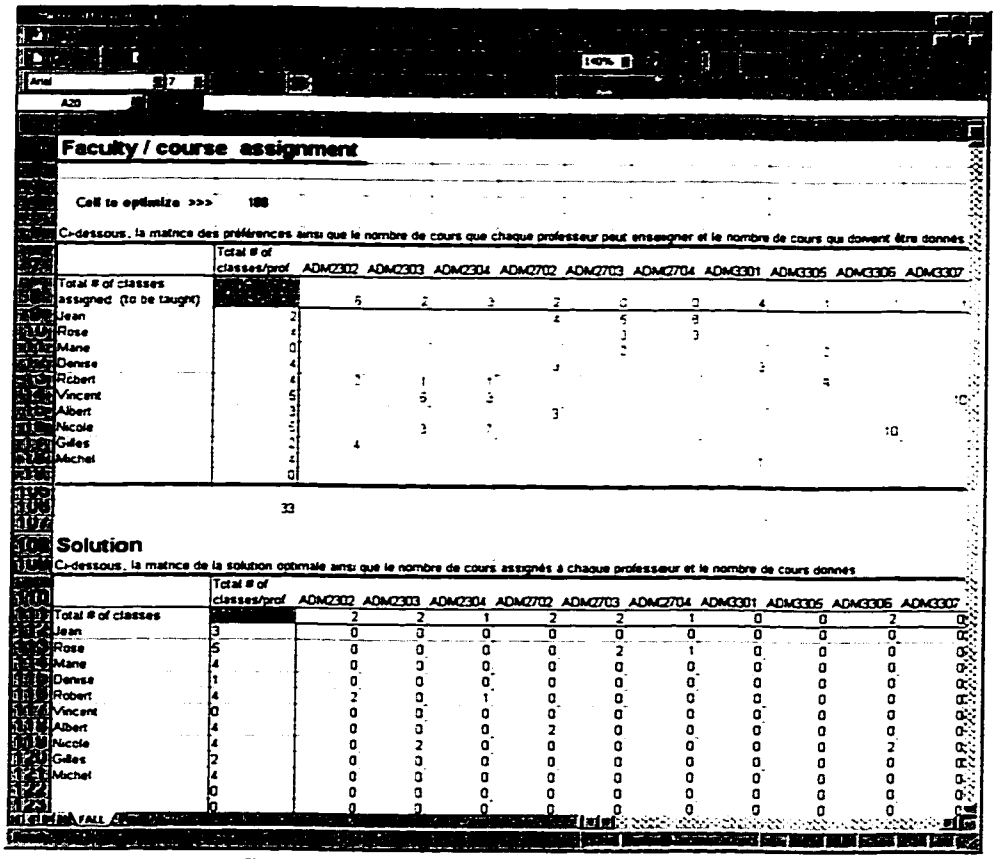


Figure 2 - Input and output tables (MS Excel)

An excerpt of the input data corresponding to Model (III) is presented below:

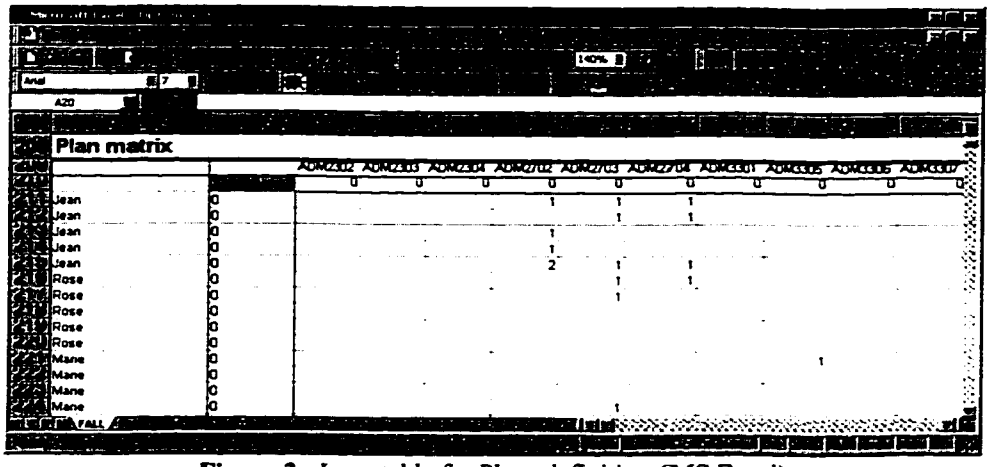


Figure 3 - Input table for Plans definition (MS Excel)

Preliminary results (1995-96) showed that the small independent models yielded results fairly similar to those of the Faculty-wide model introduced in Section 2.8 and implemented with the matrix generator AMPL described in Section 3.2. In light of the amount of clerical details needed to fully validate a solution, such results are only presented in Section 3.3.5. Inter-area adjustments can be performed by varying the objective coefficients (e.g. via dual multipliers, as shown in Section 2.8.) , as well as adding ad-hoc constraints. The teaching plans established can in turn be introduced in the Faculty-wide model, reducing its solution time. Unfortunately, the commercial spreadsheet solvers tested (Microsoft Excel) have no provision for warm starts; therefore each execution of such small adjustments may require as long to optimize as the Faculty-wide problem. Given the slowness of the spreadsheet solver, solution time varies critically according to the quality of data accepted, ranging from a few minutes for realistic plans to 20 hours for dated or hastily designed plans. Figure 4 below summarizes the solution under the form of individual plans.

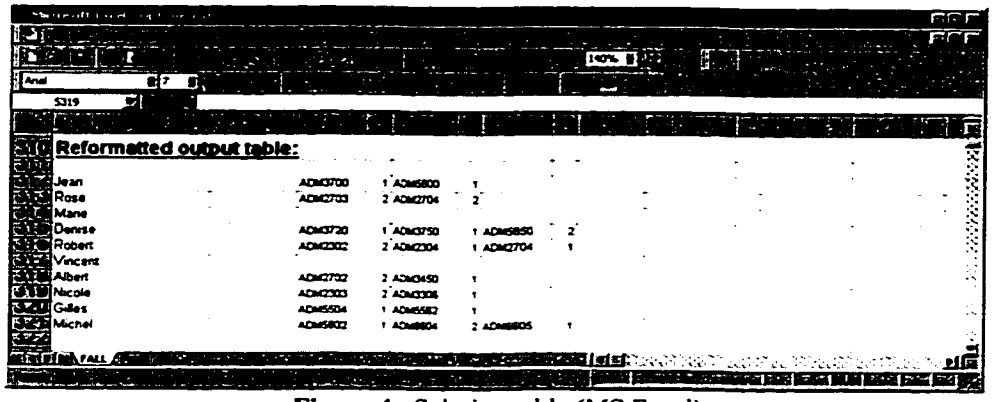


Figure 4 - Solution table (MS Excel)

One of the advantages of spreadsheet decision models is their user-friendliness and the immediate access to the data. On the other hand, their numerical capacities are quite limited; solutions taking several hours defy the purpose of interactivity. Moreover, spreadsheets are best suited to tabular format that has a immediate visual appeal. A layout of the variables x_{ij} and y_{ik} in a table as shown in Figure 2 and 3 is very convenient for decision making. However, both take

physical and memory space, so that storage capabilities may be stretched even for one area of the Faculty. To rearrange the variables x_{ij} from sparse display tables to a denser coefficient matrix for the linear program, the data handling is valid only for a particular data set and must be entirely redefined with each new application. This is compounded by the conventions used by the spreadsheet to define the linear program, which forces cell blocks to be redefined for each change of problem dimension. In summary, spreadsheets have fairly poor data definition utilities. Restructuring a coefficient matrix for a new course assignment exercise, such as at the end of an academic year, is nearly as difficult as setting up the original model.

As the next section shows, another modelling tool was investigated, for which the features and characteristics contrast with those of spreadsheet tools.

3.2. An AMPL prototype

AMPL [Fourer, 1990] is a comprehensive and powerful algebraic modelling language for linear and nonlinear optimization problems with discrete or continuous variables. AMPL uses common algebraic notation and, provided with input data, furnishes a numerical instance of a linear program sent to a numerical solver. The implementation relied on two solvers: LINDO [Schrage, 1991] and Cplex [Bixby, 1990].

One of the strengths of spreadsheet optimization is the ease with which users can manage data. On the other hand, the size and structure of the model is hard to modify. At the opposite, AMPL enables users to perform interactive modelling by easily transforming the structure of the constraints and their relative sizes. It is particularly beneficial when a more sophisticated set of constraints is introduced in the model. For example, one set of constraints tested specified that each instructor must teach a predetermined number of undergraduate required courses. Although the model was never validated numerically, AMPL provided the ability to readily define such constraints in a few hours. Therefore, AMPL can be viewed as a rule generator governing the

relationships between the administrative requirements imposed on the allocation of courses.

The syntax of the AMPL language provides for tabular data input linked to relational database management. The version available did not offer direct database retrieval; data must therefore be formatted and imported in the formulation of the model. This was accomplished with the versatile statistical package SAS. Interactive solution would require a dedicated program for mundane data transformations that are required. The implementation of the small example of Section 2.7 is presented in Appendix A.

The other main advantage of AMPL resides on its reliance on standard numerical solvers, thus giving it ability to handle the integer variables introduced in the formulation of Model (PII). The development of the AMPL model was actually the only occasion to solve the model as an integer program. In the case of the Faculty-wide model, the presence of over 300 binary variables basically precluded any guaranteed solution performance, which in practice depended critically on the quality of the plans proposed (a conclusion valid for every solution strategy explored in the thesis). In fact, for numerically erroneous or poorly adapted plans (dating from several years before), the solver LINDO took several hours to obtain an integer solution. A preliminary choice of plans by instructors in their sections is therefore a crucial ingredient of success.

The development of the spreadsheet and the AMPL models, motivated by the need to assess the robustness of the data and the constraints, answered practical needs:

- each of them requires little system integration;
- although fully operational, they are designed to work independently from the data repository;
- their actual adoption by decision maker is less likely since they would require more technical knowledge from the administrative users and their user interface is not as good as the final Decision Support System (DSS) adopted and presented next;

- finally, the growing importance of cross-area activities or Faculty-wide affiliation calls for an increasingly centralized course assignment system.

The final implementation will promote full integration, interactivity, quick solution time by foregoing some advantages of the AMPL based model such as flexible constraint or rule formation. This implementation is described in the following section.

3.3. An integrated Decision Support System

A strict partition of courses between sections would defeat the purpose of the Faculty restructuring, i.e. to encourage broad teaching expertise. Since no section is given purview on any course, the overall model encompassing all courses and full-time instructors of the Faculty serves as a potential arbitrator and long-term planning tool. Yet, its efficiency relies on adequate plans formed by the sections. Using data for the academic years 1997-98 and 1998-99, the model can furnish realistic calendars provided that section coordinators instruct the planner about idiosyncratic cases.

3.3.1 Requirements

Solution requirements

The proposed program (PIIA) needs to include approximately 500 constraints and over 1,000 variables, depending of the number of plans proposed by Instructors. Considering the size of the problem, it is still desirable to allow for a reasonable solution time (maximum of a few minutes).

System requirements

The model needs to be integrated with the actual academic unit's information system. Additional data gathering must be at a minimum in order to reduce the burden of clerical work on

administrative staff and the adoption of arbitrary parameters that need to be revised in time, with loss of continuity.

Information requirements

The following items need to be collected and managed in order to use and test the model proposed above:

- the set of courses that will be offered and the number of sections of each course;
- the list of faculty members and their teaching workloads;
- the list of courses (and plans) each instructor is qualified to teach and the respective preferences for each course (and plan), derived from historical records and prospective submissions.

The following figure shows the flow of information in the decision support system, including input requirements and the treatment of the solution.

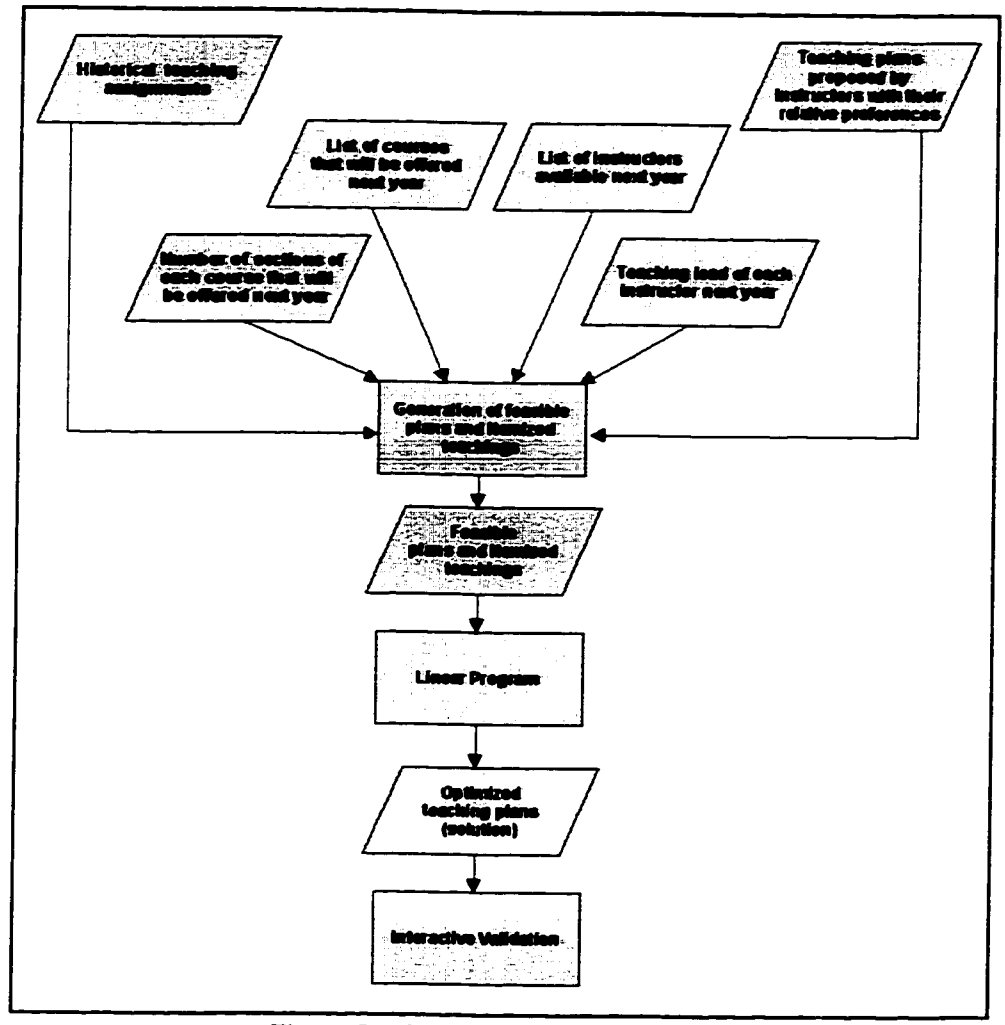


Figure 5 - Flow of information in the DSS

3.3.2. Database functions at the Faculty of Administration

The Faculty already uses a simple Paradox database system to plan and keep track of teaching assignments over the years. The initial database system was developed in a VAX environment around 1986. It was adapted to a microcomputer environment (DOS) in 1994. The following figure depicts the actual system:

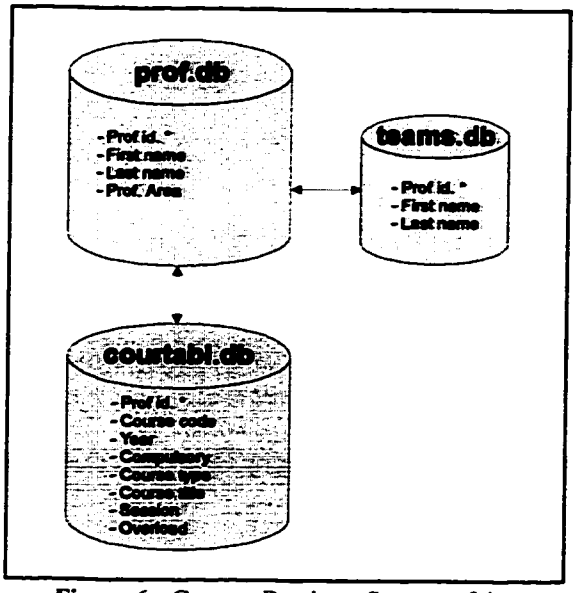


Figure 6 - Current Database System of the Faculty of Administration

A single file named *courtabl.db* acts as the main repository for all the current and past teaching of all the instructors. This file contains information focussing mainly on courses such as the course code, the title of the course, its weight (credits) and when it was taught. The file also holds some information on the instructor assigned to each course section such as her/his name (*PROF_ID*) and her/his area of specialization. A separate database file (*prof.db*) records information pertaining to each instructor, namely her or his first and last name. Both files can be linked together via the instructor's name (*PROF_ID*). The Faculty also uses a separate database (*teams.db*) to maintain joint teaching. In *courtabl.db*, the *PROF_ID* of a team consists of the last names of each member and is linked to *teams.db* on a one-to-multiple relationship.

Although the data repository is formally supported by a relational model, its design entails much redundancy. For example, in *courtabl.db,* the field "Course title" coexists with a course code that acts as a key to course information.

3.3.3. Description of the new DSS

The proposed system uses object-based Borland Delphi to maintain a new, flexible and interactive relational database management system. Its implementation entailed a design phase and included data validation, database design, the programming of the interface, the implementation of report generation features and finally, software testing and validation with real data. Chapter 2 described the algorithmic aspects of the optimization now integrated in a working Decision Support System (DSS).

In this database system, all information is maintained using the relations currently used by the Faculty with their exact format, plus additional ones. As depicted in Figure 7, the input data for the new system is generated from an existing relation that records the former plans of each instructor (*courtabl.db*). This revised table is based on the preceding *courtabl.db* and serves as an "assignment" table (ie. it does not include information on courses other than the course code and on the instructors others that the prof id, contrary to the previous *courtabl.db*). A new and separate table accepts prospective individual plans (*prodraft.db*) along with *planpref.db*, which records the plan preferences of each instructors. A distinct procedure verifies the accuracy of all the plans and adds them to *prospec.db* for the next year as candidate plans. Logical relationships between these two tables and three other tables containing information on courses (*cours.db*) and on instructors (*instruct.db* and *teams.db*) are defined using common attributes, in this case the instructor's name (*prof_id*) and the course code (indicated with * in Figure 7).

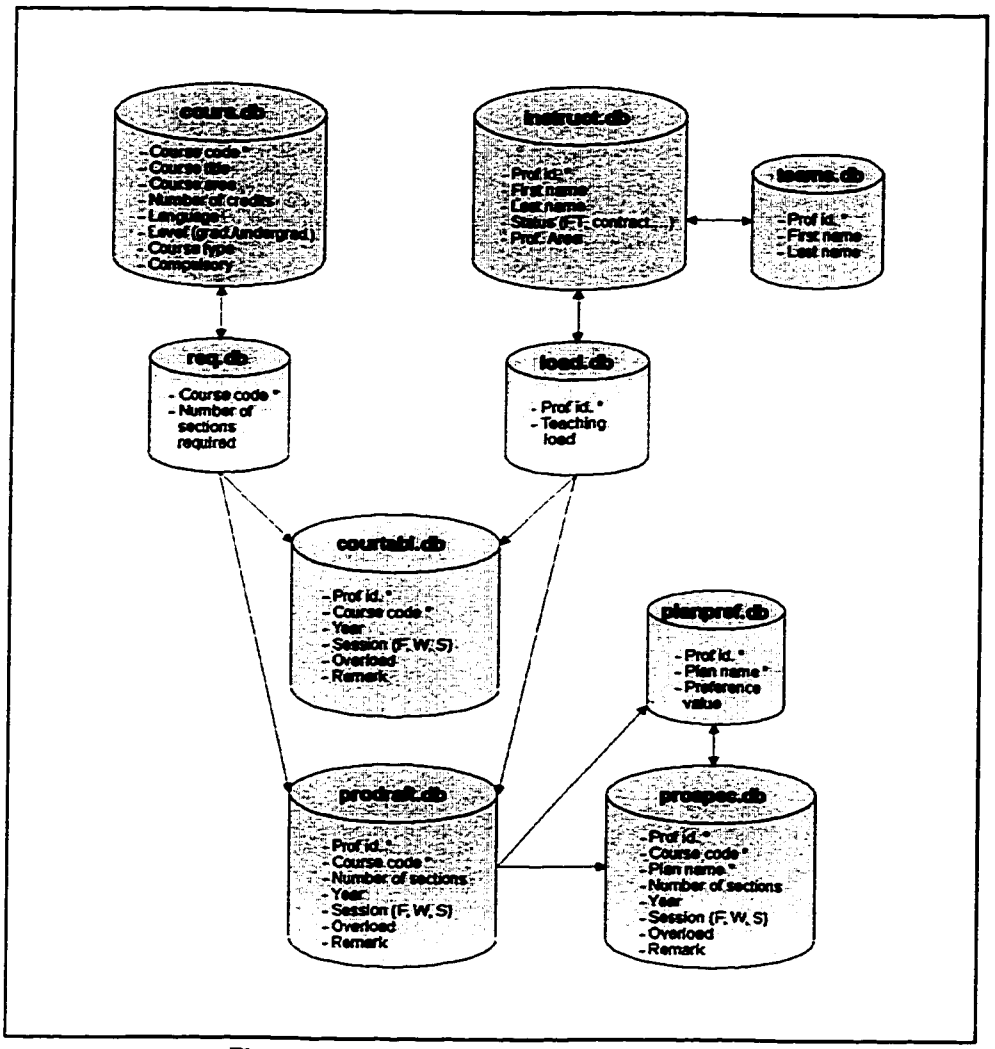


Figure 7 - Proposed Relational Database System

There are many advantages to using a relational database such as the one above. It helps to ensure data integrity, simplifies data entry and updates, improves performance and reduces disk space requirements for the application.

The programming of the interface was written in Borland Delphi and proceeded over several phases. A fully working prototype was designed with Paradox in 1996. A contractor was then hired to recast the same application in Delphi with goals of speeding and improving the visual appearance of the interface. Concomitantly, the algorithmic procedure of the optimization

was written in Delphi/Pascal and later added to this new environment. This new system also allows for expert rules to adjust fine features such as sabbatical leaves, fractional teaching loads, joint teaching, etc.

Upon starting the program, the user is presented with the following screen:

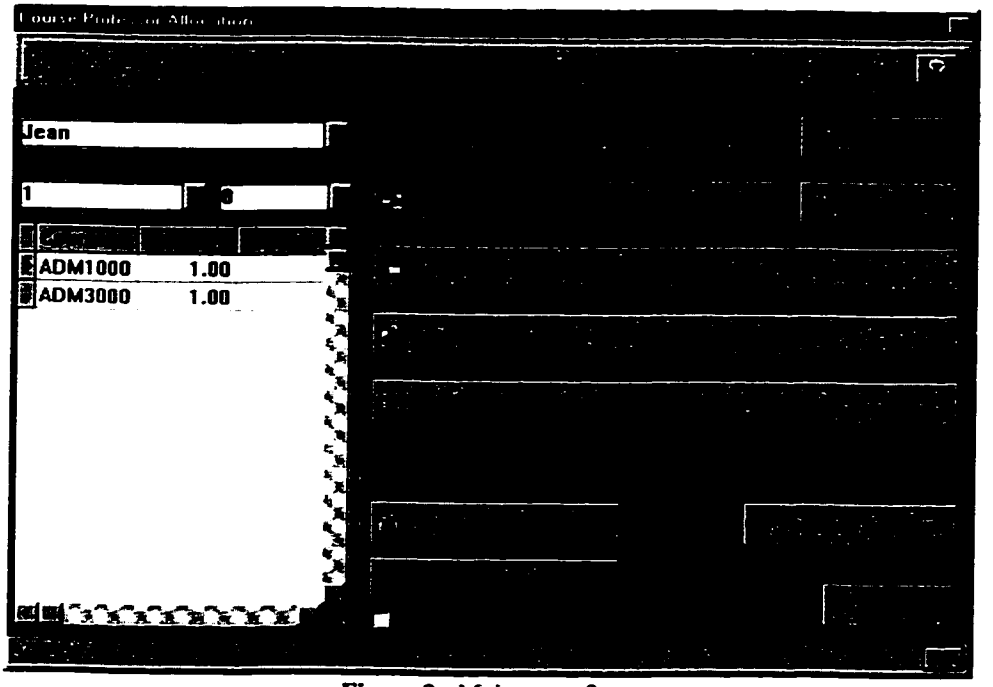


Figure 8 - Main entry form

Figure 8 represents the main menu for the formation of instructors' plans. The left window displays each course of a plan proposed by a given instructor along with her or his preference for teaching the plan. Plans defined on the left pane of the screen are temporarily saved in *prodraft.db*. A database validation of each plan checks that the course codes are valid and indeed being offered and that the specified teaching load is met. The program displays a warning if the teaching load is different from the sum of all the courses within a plan.

Selecting the button [redacted] opens a list of all the instructors such as the following:

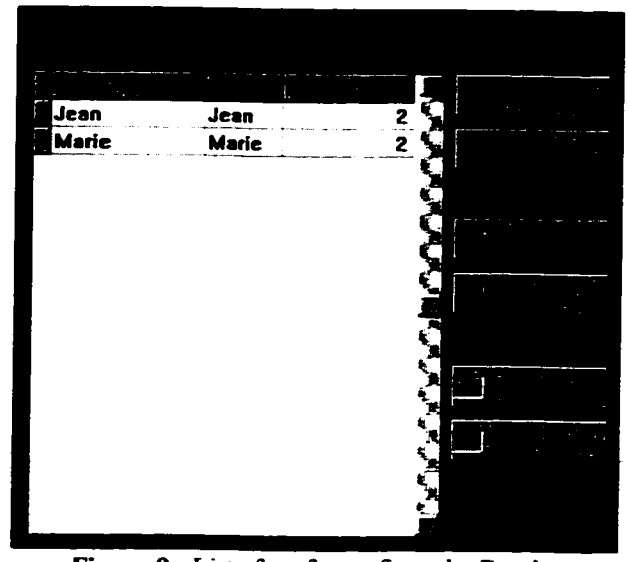


Figure 9 - List of professor from the Faculty

From the right pane, the user can select an instructor to subsequently create a new plan. It is also possible to manage the instructor's database (*instruct.db*) from this form. The two upper right most buttons of Figure 8 serve either to delete all the plans of the current instructor or to delete the plan currently displayed.

To edit an existing plan and add a course, the user can consult the list of courses to be offered next year by clicking on the button [redacted]. The right pane will show a list of courses to be offered and the user can add or delete courses from the course database (*cours.db*), as presented in Figure 10.

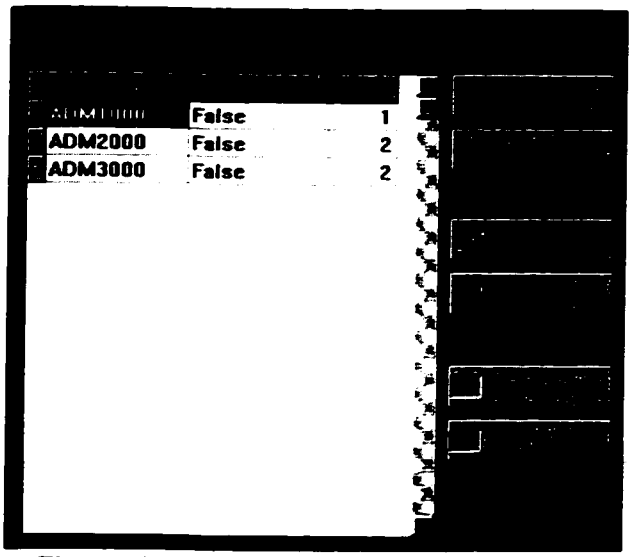
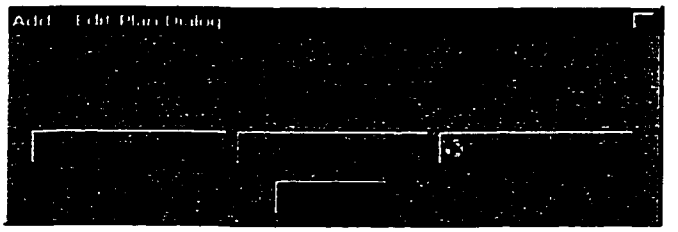


Figure 10 - List of courses to be offered next year

From this screen, the user can also add a course to the current plan and specify the number of sections to be taught by the instructor. Another convenient way of adding a course to a plan is to consult the list of all past courses taught by the instructor by clicking on the button [redacted] (these courses are drawn from *courtabl.db*). The program will warn the user if the course selected is not offered in the next academic year.

To add a new plan to the database, the user may either manually change the name of the plan or consult the list of past teaching assignments. Directly changing the name of the plan opens the following dialogue box:



The user can then choose to create an empty plan for the current instructor to later add courses to it or to edit the current plan while giving it a new name. It is also possible to add a new plan to

the database by selecting a previous teaching schedule from *courtabl.db*. By clicking on the button [redacted], all the previous plans will be displayed and the user can choose to create a new one based on past teaching to subsequently modify it as needed.

The two lower buttons serve to verify the plans from *prodraft.db* and record them in the database *prospec.db*. The user can select [redacted] to record the plan currently displayed or [redacted] to choose the plans she/he wants to record from a list of new or modified plans.

Finally, clicking on the button [redacted] invokes the schedule generation facility. Its program is presented in Appendix B. It takes the input tables (namely *courtabl.db*, *prospec.db*, *cours.db*, *instruct.db* and *planpref.db*), formats them into a linear program and uses the simplex algorithm to find an optimal solution to the problem. The actual programming of the Simplex algorithm is derived from the book *Numerical Recipes in Pascal: the art of scientific computing* [Press, Flannery, Teukolsky and Vetterling, 1989]. This program also produces a file of "optimized" teaching schedules and diagnosis tables as a list of instructors with incomplete teaching load and a list of fractional teaching assignments. The following is an example of such a list:

```
Partial teaching load
-------------------------

Jean                0.50


Fractional teaching (may be
part of team teaching)
-------------------------

Jean     ADM1000      0.50
Marie    ADM2000      1.50
```

Figure 11 - Example of incomplete teaching load and fractional teaching assignments.

3.3.4. Validation

During the course of the development of the support system, initial testing of the program was done, using the small database introduced in Section 2.7. Subsequently, during the validation phase, another database was used that consisted of only three instructors assigned to teach a total of eight courses. Although this testing data did not reflect the intended use of the system, the experience gained from repeatedly solving this problem provided very good ground for a more elaborate validation phase. Using this data, the formulated problem consisted of 46 variables and 19 constraints. An optimal allocation was successfully generated within a fraction of a second. This small example was small enough to be verified by hand and it was found that all constraints of the model were satisfied with no unassigned teaching loads.

To further test the adequacy of the decision support system, two forecasting exercises were undertaken, using previous years' data (1998-1999 and 1999-2000) for the Faculty of Administration. All the optimisation runs were executed on a 200Mhz Pentium microcomputer with 64 MB of memory. Experimental results will be discussed later, but since they involve actual members of the Faculty of Administration, names will not be disclosed.

Table 5 shows some statistics on the input data of the two forecasting exercises for the Faculty-wide model. It is important to note that, traditionally, the number of course sections planned for the upcoming year is much larger than the teaching capacity of regular professors, ie. there are not enough regular professors to teach all the courses offered by the Faculty. The residual assignment is filled by contractual professors.

	1998-1999	1999-2000
Number of courses	322	330
Number of course sections (in credits)	1,281	1,353
Number of regular faculty	66	54
Regular teaching capacity (in sections)	717	603
Number of prof/course combinations from *courtabl.db* (transportation variables)	332	341
Number of derived plans from *courtabl.db*	201	228

Table 5 - Input data

For both forecasting exercises, several tests were undertaken:

a. only with the transportation variables $x_{ij,}$ corresponding to Model (P)
b. only the plans derived from $x_{ij,}$ corresponding to Model (Π)
c. all the transportation variables x_{ij} plus the plans derived from these variables, corresponding to Model (PΠ)

The intent is to distinguish the effect of the teaching plans on the computational efficiency and the quality of the solutions proposed. For the forecasting exercise of the academic year 1998-1999, the results of the execution runs are presented in Table 6.

	Test a	Test b	Test c
Number of variables	332	201	533
Number of constraints	388	454	454
Average execution time in seconds	32	5	43
Number of courses assigned	138	153	191
Unassigned teaching load (in credits)	90.75	163.75	62.25
Fractional teaching (in credits)	9	0	12.25

Table 6 - Tabulated results for 1998-1999

Owing in major part to the restructuring of the Faculty between 1995 and 1997, the sets J_i and K_i are predominately derived from the 1997-1998 academic year in order to find an optimal course assignment for 1998-1999. Therefore professors who were on sabbatical leave in 1997-1998, especially those teaching at the graduate level, where all the course codes changed in 1996, have very few transportation variables and plans defined in the input matrix.

Execution time displayed in Table 6 and 7 increases with the number of constraints and variables defined in the model. Most conspicuous is the high solution speed of Tests b. On the other hand, the combined model of Tests c requires more time than the sum of the times taken by Tests a and Tests b, which can probably be explained by the combinatorial search performed by the Simplex algorithm.

In the last row of the tables 6 and 7, fractional teaching denotes occurrences where, for Instructor i, $\sum_{i=1}^{I} + \sum_{\substack{i=1 \\ k=1}}^{I,K_i} \beta_{ijk}\, y_{ik}$ is not a multiple of b_j. In both tests b, the optimal solution is integer, probably because there is little conflict between teaching plans, which are all derived from recent academic years. No test b produced any fractional value, although fractionality did occur in some intermediate results not reported here.

The influence of fractional variables x_{ij} is not noticeably different in Tests a and Tests c. A cursory look at the results would indicate that Model (II) is very effective. However, the satisfaction of teaching loads tells a very different story. In Tables 6 and 7, Tests c clearly dominate Tests a, which themselves dominate Tests b. One can simply explain this better performance by the presence of the variables x_{ij}, each of which acts independently. However, it is significant that the variables x_{ij} perform a much better role in the presence of plans. Note in Table 6/Test a, that the amount of 9 credits of fractional teaching is difficult to determine and can actually be raised to 10.5 because the course ADM6961 had only intermittent history of team-teaching.

In Tables 6 and 7, the total number of courses assigned shows that the use of plans as decision units (Tests b) yields assignments that contain a greater variety of courses than produced by Tests a. This difference was already noticed in the small example of Section 2.7. Tests c show that Model PII, that contains individual courses (x_{ij}) further increases the diversity of each plan. Obviously, diversity is no supreme virtue, but it reflects instructors' preference (as expressed in their plans) which the transportation model (P) diminishes arbitrarily.

Table 6 offers some particularities: there were ten appointments of new professors for 1998-1999 (for a total teaching load capacity of 91.5 credits). Since they had no previous teaching experience at the Faculty of Administration, the procedure did not assign any courses to their schedules. They are therefore excluded from the unassigned teaching load of Table 6. To fill the teaching load of these instructors, the sets J_i and K_i were constructed manually, based on academic year 1998-1999. It was felt that using actual teaching assignments would bring less distortion to the overall course roster than not assigning any teaching to the new professors. At the implementation level, the sets and their preferences are contained in *prospec.db* and *planpref.db*. By adding 10 new variables to the model, the time required to find an optimal solution increased by 7 seconds. Using these adapted plans for 1998-1999 further decreased both the number of unassigned teaching capacity (to 16.5 credits) and removed all fractional teaching.

The same procedure preluded to an actual planning exercise for academic year 1999-2000. Table 7 presents the results.

	Test a	Test b	Test c
Number of variables	341	228	569
Number of constraints	384	438	438
Average execution time in seconds	41	9	60
Number of courses assigned	137	164	203
Unassigned teaching load (in credits)	30.5	124.5	8
Fractional teaching (in credits)	6	0	5.5

Table 7 - Tabulated results for 1999-2000

For 1999-2000, the increased volume of data lengthened solution time. On the other hand, the better quality of the input data yielded a solution with less fractional teaching and markedly less unassigned teaching. To further decrease these realistically, courses of previous years were carefully chosen to make new plans. The model comprised 15 more variables and the program executed in about 66 seconds, producing no fractional teaching and 8 credits of unassigned teaching load. Hence, in the optimal solution using *prospec.db*, essentially all regular professors receive assignments corresponding to their full teaching load.

In spite of the slightly longer execution time, we find that Model (PII) satisfies some of the criteria much better than Model (P). For 1998-1999, it would be natural to compare the output of the optimisation procedure with the actual teaching assignments, a technique called backcasting. The results of backcasting are not favourable: roughly half of the assignment differs from the historical one. Additional experiments using data prior to 1997 are not attempted for several reasons. Academic years 1995-1996, 1996-1997 and 1997-1998 saw important changes in course planning. In 1995 and 1996, following a university-wide course rationalization, the Faculty reduced its course offering from 507 sections in 1995-1996 to 438 in 1996-1997. In 1997, the new MBA program entailed radical changes in the courses offered, which weakened considerably the relevance of plans based on historical information. This change actually propelled an important modification of the support system and the thesis itself.

Conclusion and recommendations

One of the strengths of the thesis is its methodical development, entailing prototyping, model development, refinement, redesign, in virtually all early phases of a complete system development life-cycle. The merits of its components are reviewed next, in the order of its development.

Modelling

The originality of the proposed model is that the information provided is self-correcting: instructors have a direct interest in providing adequate plans, lest the model assign them courses that they might contest. The significant information and guidance that the plans offer avert the design of sophisticated integer programming techniques. Its multiple objectives and its column generation maps the interactive process that presides over the allocation. Most important, the model enlarges the spectrum of acceptable plans that can be considered and may reduce conservative attitudes.

The validity of the model stretches far beyond that of an academic planning system. In its generality, it is a constraint-based model which accepts facts and events under the form of plans. Numerous such models combining constraints and observations could be applied to emerging expert systems, in which knowledge is distributed as much in its base as in the rules that can be established.

On an algorithmic level, it may be interesting to implement integer programming for variables x_{ij} or decomposition techniques to coordinate assignments within areas, such as in a Faculty comprising departments.

Implementation

Given the objective of designing a practical decision support system, part of the success

of the thesis can be measured by the factual recommendations cited below - most of which are considered for implementation by the Faculty of Administration:

- In the database, teaching teams are identified by the same type of keys as individuals. Providing a different structure where the individuality of each member of the teams is preserved, would greatly contribute to normalizing the database.

- The status of contractual (non-permanent) instructors is identified in *courtabl.db* with the special character "@" preceding their names. Some instructors do become professors of the Faculty and thus acquire dual identity! It would be preferable to use an independent variable linked to each instructor in *instruct.db* with a date of change if applicable.

- Some administrative symbols are recorded in *courtabl.db*, such as "&" to mark funding by special programs (CEFO). These typically address courses, not any of their particular offerings, and should be stored in *course.db*

- All course titles are kept in *courtabl.db*. They should be tied to the University wide academic information system that keeps one entry per course code, plus the dates of change of title for the corresponding codes. This would provide a tighter integration into a wider information system that the Faculty has developed. For example, the instructors database can be integrated within an already existing repository of the human resources of the Faculty, comprising administrative data (employee status, compensation profile, etc.) Similarly, the courses database meshes well with a University-wide curricular (course offerings, time, room, etc.).

Decision making

It is hoped that the development and implementation of a decision support system can add efficiency and fairness to the assignment process. It should significantly reduce response

time to the on-going changes faced by academic units, leading to repetitive faculty assignment exercises. It further aims at promoting inter-area and general course assignment according to the newest directions of the Faculty. To augur its success, the system is currently tested to the actual planning of academic year 1999-2000.

Bibliography

Andrew, G.M., and Collins, R., "Matching Faculty to Courses", *Colleges and Universities*, Vol. 46, No. 2, (Winter), 1971, pp. 83-89.

Badri, Masood A. "A two-stage multiobjective scheduling model for faculty-course-time assignments, *Europeen Journal of Operational Research*, Vol.94, no.1, Octobre 1996, pp.16-28.

Breslaw, J.A., "A Linear Programming Solution to the Faculty Assignment Problem", *Socio-Economic Planning Sciences*, Vol. 10 (6), 1976, pp.227-230.

Charnes, A. and W.W. Cooper, "Goal Programming and Multiple Objective Optimization - Part I", *European Journal of Operations Research*, Vol. 1, 1977, pp.39-54.

Czarina Cheng, Le Kang, Norrus Leung, George M. White. "Investigations of a Constraint Logic Programming Approach to University Timetabling", Lecture Notes in *Computer Science*, vol.1153, pp.112-129, (1996).

Dhar, Vasant and Nicky Ranganathan, "Integer Programming vs. Expert Systems: An Experimental Comparison", *Communications of the ACM*, Vol.33 (3), March 1990.

Dyer, J.S. and Mulvey, J.H., "An Integrated Optimization/Information System for Academic Planning", *Management Science*, Vol.22 (12), 1976, pp. 1332-1341.

Dyer, J.S. and Mulvey, J.H., "Computerized scheduling and planning", *New Directions for Institutional Research*, Vol. 13, Spring 1977, pp.67-86.

Fahrion, R., and Dollansky, G., "Construction of University Faculty Timetables Using Logic Programming", *Discrete Applied Mathematics*, Vol. 35, No. 3, 1992, pp. 221-236.

Fontaine, Gilbert and Truchon, Michel, "Teaching assignment by linear programming", *Cahier 8914*, Laboratoire d'économétrie, Université Laval, May 1989.

Fourer, R., D.M. Gay, and B.W. Kernighan, "A Modeling Language for Mathematical Programming", *Management Science*, 1990, pp. 519-554.

Harwood, G.B. and Lawless, R.W., "Optimizing Organizational Goals in assigning Faculty Teaching Schedules", *Decision Sciences*, Vol. 6, No.3, 1975, 513-524.

Hitchcock, F.L., "The distribution of a product from several sources to numerous localities", *Journal of Mathematics and Physics*, Volume 20, number 3, 1941 pp. 224-230.

Kang, L. and White, G.M., "A Logic Approach to the Resolution of Constraints in Timetabling", *European Journal of Operational Research*, Vol. 61, 1992, pp. 306-317.

Kang L., Von Schoenberg G.H., White G.M., "Computer University Timetabling Using Logic", *Computers and Education*, Vol. 1(2) 1991, pp. 145-153.

Le Kang and George M. White. "A Logic Approach to the resolution of Constraints in Timetabling", European Journal of Operational Research, vol.61, pp.306-317, (1992).

Le Kang, George H. von Schoenberg and George M. White. "Complete University Timetabling Using Logic", Computers in Education, vol. 17, pp.145-153, (1991).

McClure R.H. and Wells, C.E., "A Mathematical Programming Model for Faculty Course Assignments", *Decision Sciences*, Vol.15, No. 3, 1984, 409-420.

Press, W.H., B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in Pascal: the art of scientific computing*, Cambridge University Press, Cambridge, 1989.

Schniederjans M.J. and Kim G.C., "A Goal Programming Model to optimize Departmental Preference in Course Assignments", *Computers and Operations Research*, Vol.14 (2) 1987, pp. 87-96.

Schrage, Linus, "LINDO: An Optimization Modeling System", Boyd & Fraser/The Scientific Press, Danvers, MA, 1998.

Shih, W. and Sullivan, J.A., "Dynamic Course Scheduling for College Faculty via Zero-One Programming", *Decision Sciences*, Vol.8, No.4, Oct. 1977, pp. 711-721.

Tillett, P.I., "An Operations Research Approach to the Assignment of Teachers to Courses", *Socio-Economic Planning Sciences*, Vol. 9 (3/4) 1975, pp. 101-104.

White, D.J., "A Note of Faculty Timetabling", *Operations Research Quarterly*, Vol. 26 (4) 1975, pp. 875-878.

Appendix A - A course assignment model in AMPL

```
***** SETS AND PARAMETERS *****

set prof;
param load{prof}  >= 0;

set course;
param req{course} >= 0;

set prof_course within {prof,course};
param pref_prof_course{prof_course};

param plan_max;
set plan:=1..plan_max;

set plan_prof within {plan,prof};
param pref_plan_prof{plan_prof};

set plan_prof_course within {plan_prof,course};
param numb_sect{plan_prof_course};


***** VARIABLES *****

var X_prof_course{prof_course} >= 0;
var Y_plan_prof{plan_prof} integer >= 0;


***** PROGRAM *****

maximize total_pref:
    sum {p,c} in prof_course} pref_prof_course[p,c] * X_prof_course[p,c] +
    sum {n,p} in plan_prof}   pref_plan_prof[n,p]  *  Y_plan_prof[n,p]   ;

***** CONSTRAINTS *****

subject to

cover_course {c in course: req[c]>0 }:
     sum {p,c} in prof_course} X_prof_course[p,c]  +
     sum {n,p,c} in plan_prof_course} Y_plan_prof[n,p] * numb_sect[n,p,c]
                                                          <=  req[c];

respect_load {p in prof: load[p]>0 }:
  sum {p,c} in prof_course} X_prof_course[p,c] +
  sum {n,p,c} in plan_prof_course} Y_plan_prof[n,p] * numb_sect[n,p,c]
                                                          <= load[p];

one_plan_per_prof {p in prof: load[p]>0 }:
    sum {n,p} in plan_prof}    Y_plan_prof[n,p] <= 1;


***** DATA *****

data;

param: prof:      load :=
    Jean             2
    Marie            2
;

param:  course:     req :=
     ADM1300          1
     ADM1301          2
     ADM1701          2
;

param:  prof_course:    pref_prof_course :=
Jean        ADM1300           1
Jean        ADM1701           2
Marie       ADM1300           2
Marie       ADM1301           2
;

param: plan_prof_course:  numb_sect :=
1 Jean        ADM1300         1
1 Jean        ADM1701         1
2 Jean        ADM1701         2
```

```
1 Marie       ADM1300         2
2 Marie       ADM1301         2
;

param plan_max:=99;

param: plan_prof:  pref_plan_prof :=
1 Jean        8
2 Jean        3
1 Marie       9
2 Marie       5
;


solve;
option display_width 35;
display {(p,c) in prof_course: X_prof_course[p,c] >0} (0, X_prof_course[p,c]) ;
display {(n,p) in plan_prof: Y_plan_prof[n,p] >0} (0, Y_plan_prof[n,p]) ;
end;
```

Appendix B - Model implementation in Delphi

```
unit Main;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, DBCtrls, DBLookup, Grids, DBGrids,
  Buttons, DB, DBTables, FileCtrl, uPlanDlg, PCourses, uDBDebg, About, PlanList,
  ShellAPI, Val;

const ProDraftDB   = 'ProDraft';    {*** short name for ProDraft db ***}
      DraftDBPart  = 'Draft';       {*** short name for Draft db ***}

const iForDebug     = 0;
      iInform       = 1;

type
    string7 = string[7];
    actArrayType  = array [1..450] of string7;

    TMainFrm = class(TForm)
    Panel1: TPanel;
    Pages: TNotebook;
    btnAddProf: TBitBtn;
    btnDelProf: TBitBtn;
    btnPrevPlans: TBitBtn;
    btnDelPlan: TBitBtn;
    GridMain: TDBGrid;
    Label1: TLabel;
    Label2: TLabel;
    btnFacCourses: TBitBtn;
    btnProfCourses: TBitBtn;
    btnRemCourse: TBitBtn;
    btnRecordPlan: TBitBtn;
    btnQuit: TBitBtn;
    GridPage2: TDBGrid;
    lblTitle: TLabel;
    btnAddIt: TBitBtn;
    btnMainPage: TBitBtn;
    StatusBar: TPanel;
    tblProfProsPec: TTable;
    tblProspecDraft: TTable;
    tblProspecDraftSet: TDataSource;
    tblProfProsPecSet: TDataSource;
    cProf: TComboBox;
    cPlan: TComboBox;
    qPlan: TQuery;
    Bevel1: TBevel;
    Query: TQuery;
    QuerySet: TDataSource;
    Label4: TLabel;
    qProspec: TQuery;
    qProspecSet: TDataSource;
    tblCours: TTable;
    PDCourse: TStringField;
    PDSection: TFloatField;
    PDCompulsory: TBooleanField;
    PDProf: TStringField;
    PDPlan: TStringField;
    PDValue: TFloatField;
    PDNewModified: TBooleanField;
    btnValidateSched: TBitBtn;
    qMisc: TQuery;
    btnDelete: TBitBtn;
    btnSave: TBitBtn;
    Database: TDatabase;
    SGridPage2: TStringGrid;
    btnLastRec: TBitBtn;
    btnFirstRec: TBitBtn;
    btnAbout: TBitBtn;
    tblPlanPref: TTable;
    tblProspec: TTable;
    tblProspecDraftProfID: TStringField;
    btnLastMsg: TBitBtn;
    StatBarTimer: TTimer;
    cbDebug: TCheckBox;
    btnRecordSomePlans: TBitBtn;
    cPlanPref: TComboBox;
```

```
    btnCopy: TBitBtn;
    tblProspecASCII: TTable;
    btnGeneratecourses: TBitBtn;
    tblProfcrs: TTable;
    tblProfcrsSet: TDataSource;
    BitBtn1: TBitBtn;
    tblpostopt: TTable;
    tblPostoptSet: TDataSource;
    procedure btnQuitClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure cProfClick(Sender: TObject);
    procedure cPlanClick(Sender: TObject);
    procedure ShowPage2(Sender: TObject);
    procedure btnMainPageClick(Sender: TObject);
    procedure btnAddItClick(Sender: TObject);
    procedure qProspecCalcFields(DataSet: TDataset);
    procedure QueryBeforeDelete(DataSet: TDataset);
    procedure QueryBeforeEdit(DataSet: TDataset);
    procedure cProfKeyPress(Sender: TObject; var Key: Char);
    procedure cProfExit(Sender: TObject);
    procedure cPlanKeyPress(Sender: TObject; var Key: Char);
    procedure cPlanExit(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure btnDelProfClick(Sender: TObject);
    procedure btnDeleteClick(Sender: TObject);
    procedure btnRemCourseClick(Sender: TObject);
    procedure btnDelPlanClick(Sender: TObject);
    procedure btnSaveClick(Sender: TObject);
    procedure btnFirstRecClick(Sender: TObject);
    procedure btnLastRecClick(Sender: TObject);
    procedure btnRecordPlanClick(Sender: TObject);
    procedure btnValidateSchedClick(Sender: TObject);
    procedure btnLastMsgClick(Sender: TObject);
    procedure btnAboutClick(Sender: TObject);
    procedure StatBarTimerTimer(Sender: TObject);
    procedure tblProspecDraftSetDataChange(Sender: TObject; Field: TField);
    procedure btnRecordSomePlansClick(Sender: TObject);
    procedure tblProspecDraftBeforeCancel(DataSet: TDataset);
    procedure cPlanPrefClick(Sender: TObject);
    procedure btnGeneratecoursesClick(Sender: TObject);
    procedure btnCopyClick(Sender: TObject);
    procedure FillGridAddCourseProfs; {(Sender: TObject);}

  private
    { Private declarations }
    lProfIDs:      TStringList;
    LastMsg:       string;
    DraftDBName:   string;
    IgnoreModsInDraft: Boolean;
    NewPlanNameList: TStringList;

    procedure DisplayHint(Sender: TObject);
    procedure FillGridAddPlan;
    procedure FillGridAddProf;
    procedure FillGridAddFacCourses;
    procedure FillGridAddProfCourses;
    procedure cProfValidate;
    procedure cPlanValidate(FromPage2: boolean);
    function  DeletePlanRecs(const tables: array of string; Prof, Plan: string;
                             Prompt: boolean): boolean;
    procedure AddNewPlan(NewPlanName, OldPlanName: string);
    procedure ModifyPlanName(ProfID, OldPlan, NewPlan: string);
    procedure InformUser(msg: string; DebugFlag: integer);
    procedure InitializeDBs;
    procedure RecordPlan(TheProfID, Plan: string);
  public
    { Public declarations }
    DebugOn:   boolean;
    PBuffer:   PChar;
  end;

var
  MainFrm: TMainFrm;


procedure Delay(msecs:integer);
procedure RTrim(var s: string);

implementation

{$R *.DFM}


procedure Delay(msecs:integer);
var
   FirstTickCount:longint;
begin
   FirstTickCount:=GetTickCount;
   repeat
       Application.ProcessMessages; {allowing access to other
                                     controls, etc.}
   until ((GetTickCount-FirstTickCount) >= Longint(msecs));
end;


procedure TMainFrm.DisplayHint(Sender: TObject);
begin
   if not StatBarTimer.Enabled then
```

```
        StatusBar.Caption := Application.Hint;
end;

procedure TMainFrm.btnQuitClick(Sender: TObject);
var msgrc: word;
begin
   with qMisc do
   begin
        Active := false;
        SQL.Clear;
        SQL.Add('Select Distinct a."Prof ID", a."Prof", a."Plan" ' +
                'from ' + DraftDBName + ' a');
        SQL.Add('Where NewModified = ''t'' Order By a."Prof ID", ' +
                'a."Prof", a."Plan"');
        Active := true;

        if RecordCount = 0 then
        begin
            msgrc := MessageDlg('It is safe to quit, there are no pending modifications.'
                                + chr(13) +
                                'Do you really want to quit?',
                                mtConfirmation, [mbYes, mbNo], 0);
        end
        else
        begin
           msgrc := MessageDlg('There are ' + IntToStr(RecordCount, +
                               ' modified plan(s) that are not yet saved!' +
                               chr(13) +
                               'Do you really want to quit?',
                                mtConfirmation, [mbYes, mbNo], 0);
        end;
   end;

   if msgrc = mrYes then
     begin
      tblProspecDraft.Active := false;
      tblProsPec.Active := false;
      tblprofProspec.Active := false;
      tblcours.Active := false;
      tblplanpref.Active := false;
      tblprofcrs.Active := false;
      release;
      Application.Terminate;
     end;
end;

FUNCTION findAct( target :  string7;
                  VAR actIndex : integer;
                  VAR actArray : actArrayType;
                  numCrs : integer ) : boolean;

{ this function can also be found in val.pas
  simple search and indexing of an array containing titles of academic activities}

   BEGIN          {DEFINITION OF FUNCTION}
   actIndex := one;
   WHILE (actIndex<= numCrs) AND (target <> actArray[actIndex]) DO
         BEGIN    {Check next Course }
         actIndex := one + actIndex;
         END;     {Check next Course }

   { the value of findAct tells whether target appears in the activity table }
   findAct := actIndex <= numCrs ;
   END;           {DEFINITION OF FUNCTION}

{------------------------- PROCEDURE sect2num -------------------------------}
PROCEDURE sect2num(course: string7;
                   VAR numSect: real;
                   year: integer
                );
BEGIN

       IF ((COPY ( course,five,one) = '2' ) OR
           (COPY ( course,five,one) = '4' ) OR
           (COPY ( course,five,one) = '6' ) OR
           (COPY ( course,five,one) = '8' )
          { there was an additional condition below, which would qualify
           the preceding ones properly for ADM, ADX, HAH, SYS
           it may need to be re-introduced to separate from ECO, LAW
          and ((COPY ( course, one,two) ='AD' ) )

          )
          THEN numSect := 0.5*numSect    {1.5 credit courses}
          ELSE
            {the following logical test reflects concern viz. speed, e.g.
            since few records pass the 1st test set up for IMBA,
            we need encode neither 1997 nor 7 as variables}
            IF (COPY (course,five,one) = '9' )
              THEN
                 BEGIN
                 IF (Year > 1997)
                    THEN
                       IF (COPY (course,one,7) = 'ADM6931') OR
                          (COPY (course,one,7) = 'ADM6932') OR
                          (COPY (course,one,7) = 'ADM6964') OR
                          (COPY (course,one,7) = 'ADM6966')
                          THEN numSect := 0.5*numSect
                          ELSE
```

```
                              BEGIN
                              IF (COPY (course,one,7) = 'ADM6970') OR
                                 (COPY (course,one,7) = 'ADM6932')
                                 THEN numSect := 1./3.*numSect;
                              IF (COPY (course,one,7) = 'ADM6931') OR
                                 (COPY (course,one,7) = 'ADM6980')
                                 THEN numSect := 2./3.*numSect;
                              END
                        END
                     ELSE
                       IF (COPY (course,one,7) = 'ADM2380') OR
                          (COPY (course,one,7) = 'ADM2780')
                                THEN numSect := 2./3.*numSect;

END;  {Procedure}

procedure TMainFrm.FormCreate(Sender: TObject);
var i: integer;
    canmod: boolean;
    dbpath: string;
begin
   Application.OnHint := DisplayHint;
   Pages.PageIndex := 0;

   IgnoreModsInDraft := false;
   DebugOn := ParamCount > 0;

   with SGridPage2 do
   begin
        ColCount        := 3;
        ColWidths[0]    := 15;

        top     := GridPage2.top;
        left    := GridPage2.left;
        width   := GridPage2.width;
        height  := GridPage2.height;
        visible := false;
   end;

   {*** find the path of the database ***}
   with DataBase do
   begin
        Connected := false;
        dbpath := Copy(Params[0], 6, 200);
        if not DirectoryExists(dbpath) then
        begin
            dbpath := ExtractFilePath(Application.ExeName) + '..\c95-97\';
            dbpath := ExpandFileName(dbpath);
            if not DirectoryExists(dbpath) then
            begin
                dbpath := ExtractFilePath(Application.ExeName) + 'data\';
                dbpath := ExpandFileName(dbpath);
            end;
            Params[0] := 'PATH=' + dbpath;
        end;
        Connected := true;
   end;

   InitializeDBs;
end;


procedure TMainFrm.InitializeDBs;
var i:          integer;
    tname:      string;
    dbpath:     string;
    dtable:     TTable;
begin
   {*** first set the name for the draft database ***}

   i            := 0;
   DraftDBName  := '';

   dbpath := Copy(DataBase.Params[0], 6, 200);
   if dbpath[length(dbpath)] <> '\' then dbpath := dbpath + '\';

   while (i<1000) and (DraftDBName = '') do
   begin
        tname := DraftDBPart;
        if i < 100 then tname := tname + '0';
        if i < 10 then tname := tname + '0';
        tname := tname + IntToStr(i) + '.db';

        if not FileExists(dbpath + tname) then
        begin
            DraftDBName := Copy(tname, 1, Length(tname) - 3);
            break;
        end;
        Inc(i);
   end;

   dtable := TTable.Create(self);
   dtable.DataBaseName := tblProspecDraft.DataBaseName;
   dtable.TableName := DraftDBName + '.db';
   tblProsPecDraft.FieldDefs.Update;
   tblProsPecDraft.IndexDefs.Update;
   dtable.FieldDefs.Assign(tblProsPecDraft.FieldDefs);
   dtable.IndexDefs.Assign(tblProsPecDraft.IndexDefs);
```

```
    dTable.CreateTable;

    tblProsPecDraft.TableName := DraftDBName + '.db';
    tblProfProsPec.TableName  := DraftDBName + '.db';
    qPlan.SQL[1] := 'from ' + DraftDBName + ' a';


    tblProsPecDraft.Active := true;
    tblProfProsPec.Active := true;

    lProfIDs := TStringList.Create;
    lProfIDs.Clear;

    NewPlanNameList := TStringList.Create;
    NewPlanNameList.Clear;

    with qMisc do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('Select distinct t.Prof, p."Prof ID" from Prospec p, Prof t');
        SQL.Add('where p."Prof ID" = t."Prof ID"');
        active := true;
        cProf.Items.Clear;
        while not eof do
        begin
            i := cProf.Items.Add(Fields[0].AsString);
            lProfIDs.Insert(i, Fields[1].AsString);
            next;
        end;
        Active := false;
        cProf.ItemIndex := 0;
        cProfClick(NULL);
    end;

    dTable.Free;
end;

procedure TMainFrm.FormDestroy(Sender: TObject);
begin
    tblProfProsPec.Active := false;

    {*** now delete the Draft database ***}
    with tblProspecDraft do
    begin
        Active := false;
        InformUser('Deleting table ' + tblProspecDraft.TableName, iInform);
        DeleteTable;
    end;
end;

procedure TMainFrm.cProfClick(Sender: TObject);
var i, idx: integer;
    profid: string;
begin
    {*** first, try to locate this prof in the Draft database ***}
    idx    := cProf.Items.IndexOf(cProf.Text);
    profid  := lProfIDs[idx];
    tblProfProsPec.FindNearest([profid]);
    with tblProspecDraft do
    begin
        if FieldByName('Prof ID').AsString <> profid then
        begin
            {*****************************************************************
             * this prof info is not yet copied in the Draft database, so we
             * have to copy all the prof's info in the Draft database, marking
             * all these new record as unmodified
             *****************************************************************}
            with qProspec do
            begin
                Active := false;
                Params[0].AsString := profid;
                Active := true;

                if RecordCount > 0 then
                begin
                    InformUser('Copying all records for this prof', iForDebug);
                    first;
                    with tblProspecDraft do active := true;
                    while not eof do
                    begin
                        tblProspecDraft.AppendRecord([
                                        FieldByName('Course').AsString,
                                        FieldByName('Section #').AsFloat,
                                        nil,
                                        FieldByName('Prof').AsString,
                                        FieldByName('Prof ID').AsString,
                                        FieldByName('Plan').AsString,
                                        FieldByName('Value').AsFloat,
                                        False]);
                        next;
                    end;
                end
                else
                begin
                    {*** prof has no records but must add him anyway ***}
                    InformUser('Creating new record for prof', iForDebug);
                    with tblProfProsPec do
```

```
                    begin
                        Append;
                        FieldByName('Prof').AsString    := cProf.Text;
                        FieldByName('Prof ID').AsString := profid;
                        Post;
                    end;
                end;
                tblProspecDraft.Refresh;
                qProspec.Active := false;
            end;
        end;
    end;

    tblProfProsPec.FindKey([profid]);


    if DebugOn then
        if DBDebugFrm.ShowDebugWindow(tblProsPecDraft) = mrCancel then Exit;


    with qPlan do
    begin
        {*** next we fill up the Plan combo box with all the prof's plans ***}
        try
            Active := false;
            ParamByName('leprof').AsString := profid;

            Active := true;
            if DebugOn then
                if DBDebugFrm.ShowDebugWindow(qPlan) = mrCancel then
                    Exit;

            cPlan.Items.Clear;
            while not eof do
            begin
                cPlan.Items.Add(FieldByName('Plan').AsString);
                next;
            end;
        finally
            cPlan.ItemIndex := 0;
            cPlanClick(nil);
            first;
        end;
    end;

end;

procedure TMainFrm.cPlanClick(Sender: TObject);
begin
   {tblProfProsPec.FindNearest([cProf.Text, cPlan.Text]);}
   tblProfProsPec.FindKey([lProfIDs[cProf.ItemIndex], cPlan.Text]);
end;

procedure TMainFrm.ShowPage2(Sender: TObject);
var    DontShowPage: boolean;
       tag:          integer;
begin
   btnMainPage.Caption := '&Cancel';

   tag := TWinControl(Sender).tag;
   GridPage2.ReadOnly  := (tag in [1,3]);
   btnDelete.enabled   := (tag in [1,3]);
   btnSave.enabled     := btnDelete.Enabled;

   if tag = btnPrevPlans.tag then
      btnAddIt.Caption := 'Edit it'
   else
       btnAddIt.Caption := 'Add it';

   SGridPage2.visible  := false;
   GridPage2.visible   := true;

   btnFirstRec.enabled := GridPage2.visible;
   btnLastRec.enabled  := btnFirstRec.enabled;

   DontShowPage := false;
  { tag := 5; }
   case tag of
       1: FillGridAddProf;
       2: FillGridAddPlan;
       3: FillGridAddFacCourses;
       4: FillGridAddProfCourses;
       5: FillGridAddCourseProfs;
     else DontShowPage := true;
   end;

   if not DontShowPage then Pages.PageIndex := 1;
   Pages.Tag := tag;
end;

procedure TMainFrm.btnMainPageClick(Sender: TObject);
begin
   Pages.PageIndex := 0;
end;

procedure TMainFrm.btnAddItClick(Sender: TObject);
var i:             integer;
    s,planvalue:   string;
```

```
    ChangePage:    Boolean;
    Compulsory:    Boolean;
    CourseExists:  boolean;
begin
   ChangePage          := true;
   btnMainPage.Caption := '&Close';

   case Pages.tag of
       1:  begin   {*** add a prof in combo box list, if not already there ***}
               s := Query.FieldByName('Prof').AsString;
               if cProf.Items.IndexOf(s) < 0 then
               begin
                   i := cProf.Items.Add(s);
                   lProfIDs.Insert(i, Query.FieldByName('Prof ID').AsString);
                   cProf.ItemIndex := i;
                   cProfClick(nil);
               end;
           end;

       2:  begin   {*** add a plan in combo box list, if not already there ***}
               with SGridPage2 do
               begin
                   s := Cells[1, Row];
               end;

               i := cPlan.Items.IndexOf(s);
               if i < 0 then
               begin
                   PlanValue := InputBox('Edit Plan',
                           'Please enter name for the plan ' + chr(13) +
                           'that you are creating'  + chr(13) + chr(13) +
                           'WARNING: Re-using an existing name will' + chr(13) +
                           'generate an error message',
                           '');
                   AddNewPlan(PlanValue, s);
               end
               else
               begin
                   MessageDlg('The plan already existing will be displayed',
                           mtInformation, [mbOK], 0);

                   cPlan.ItemIndex := i;
                   cPlanClick(cPlan);
               end;
           end;

       3,4:begin   {*** add a course in plan grid ***}
               try
                   {* first let's see if course exists in next year course list *}
                   s := GridMain.DataSource.Dataset.FieldByName('Course').asstring;
                   CourseExists :=
                       tblCours.FindKey([GridPage2.DataSource.Dataset.FieldByName('Course')]);

                   if CourseExists = false then
                   begin
                      MessageDlg('Course is not scheduled to be offered by faculty next year' + chr(13) +
                                 'It must first be added to list of course offerings',
                                 mtInformation, [mbOK], 0);

                      exit;
                   end;

                   Compulsory := tblCours.FieldByName('Compulsory').AsBoolean;

                   ChangePage := false; {*** to add many courses quickly ***}
                   with GridMain.DataSource.Dataset do
                   begin
                       planValue := cPlanPref.Text;
                       {*** to avoid empty record ***}
                       if FieldByName('Course').IsNull then
                       begin
                           Edit;
                       end
                       else
                       begin
                           Append;
                       end;

                       {*** NOTE: if Fields order changes must update this ***}
                       SetFields([   Query.Fields[0].AsString, {*Course*}
                                     Query.Fields[1].AsFloat,  {*Section #*}
                                     {Mario Boileau's implementation: Query.Fields[2].AsBoolean,*Compulsory*}
                                     Compulsory,               {*Compulsory*}
                                     cProf.Text,               {*Prof*}
                                     lProfIDs[cProf.ItemIndex],{*Prof ID*}
                                     cPlan.Text,               {*Plan*}
                                     planValue,                {*Value*}
                                     true                      {*NewModified*}
                                 ]);
                       Post;
                   end;
               except
                   on E: EDataBaseError do
                   begin
                       if Pos('Key violation', E.Message) > 0 then
                           s := 'This course is already in current plan.'
                       else
                           s := E.Message;
```

```
                    GridMain.DataSource.Dataset.Cancel;
                    MessageDlg(s, mtInformation, [mbOK], 0);
                    ChangePage := false;

                end;
            end;
        end;
    end;

    if ChangePage then Pages.PageIndex := 0;
end;

procedure TMainFrm.AddNewPlan(NewPlanName, OldPlanName: string);
type
    RowArray   = array [1 .. 450] of real;
    RowPointer = ^RowArray;
var
  Present: TDateTime;
  currYear, Month, Day : Word;
  t : RowPointer;
  varIndex : integer;
  actIndex, numCrs : integer;
  numSect: real;
  actArray : actArrayType;
  i: integer;
  s: string;
  fld: TField;

  begin
   {*** add all courses for this plan from CourTabL*** }

   { this is the original search written by Mario Beaulieu, replaced by elements patterned
     after val.pas in order to properly count less-than-3cr. courses, so as to add up to an
     adequate teaching load. Similar queries should also be replaced, e.g. which courses/plans
     have been delivered by a given professor or which professors have taught a given course

  with qMisc do
       begin
       Active := false;
       SQL.Clear;
       {*****!!!!Question where take Course info  Courtabl et/ou Cours ***
       SQL.Add('Select Distinct Course, WF, Compulsory, c."Acad yr"');
       SQL.Add('from CourTabl c, Prof p');
       SQL.Add('where (c."Prof ID" = p."Prof ID") and');
       SQL.Add('(p."Prof ID" = ''' + lProfIDs[cProf.ItemIndex] + ''') and');
       SQL.Add('(c."Acad yr" = ''' + OldPlanName + ''') ');
       Active := true;

       InformUser('Selected all courses from CourtTabl and Prof', iForDebug);    }

  new(t);
  with tblProfcrs do
          begin   {find prospective courses in COURTABL }
          active := True;
          tblProfcrs.First;
          numCrs:= 0;
          WHILE (NOT tblProfcrs.eof) DO
              BEGIN   {next record of COURTABL }
              IF(
                    (tblProfcrs.FieldByName('Prof ID').AsString = lProfIDs[cProf.ItemIndex] )
                     and
                    (tblProfcrs.FieldByName('Acad yr').AsString = OldPlanName)
                  )
                  THEN
                    BEGIN      { only prospective courses }
                    IF not findAct( tblProfcrs.FieldByName('Course').AsString, actIndex, actArray, numCrs)
                      THEN
                         BEGIN  {extend actArray}
                         numCrs := numCrs + one;
                         t^[numCrs] := 0.;
                         actArray[numCrs] := tblProfcrs.FieldByName('Course').AsString;
                         END;   {extend actArray}
                    numSect:=one;
                    sect2num (tblProfcrs.FieldByName('Course').AsString,numSect,currYear);
                    t^[actIndex] := t^[actIndex] + numSect;
                    END;       { only prospective courses }
              tblProfcrs.Next;
              END;      { next record of COURTABL }
          END;       { find prospective courses in COURTABL}

{originally written by Mario and re-implementable
if DebugOn then
      if DBDebugFrm.ShowDebugWindow(tblProsPecDraft) = mrCancel
          then      Exit      ;                                    }

{*** add these records in the Draft table ***}

InformUser('Adding all records for this plan in Draft table', iForDebug);

with tblProspecDraft do
        begin { add record to prospectDraft}
        {active := True;}
        IgnoreModsInDraft := true;
        for  actIndex:= 1 to numCrs do
           BEGIN  { write a[] in draft.db}
           append;
           FieldByName('Prof').AsString := cProf.Text;
           FieldByName('Prof ID').AsString :=
```

```
                        lProfIDs[cProf.ItemIndex];
            FieldByName('NewModified').AsBoolean := true;
            FieldByName('Course').AsString := actArray[actIndex];
            FieldByName('Section #').AsFloat := t^[actIndex];
            FieldByName('Compulsory').AsBoolean := true;
               { it really should be as below, but need to match the course names again
               (
               TblProfcrs.FieldByName('Course').AsString = 'C'
               or
               TblProfcrs.FieldByName('Course').AsString = 'Cc
               );    }
            FieldByName('Plan').AsString := NewPlanName;

            {*** corrected bug Key Violation when added existing plan
                from prof, then changing plan name then Edit as new plan...}
            IgnoreModsInDraft := false;
            end;    { write a[] in draft.db}
          next;
          end; { add record to prospectDraft}

  if numCrs (RecordCount} = 0 then  with qMisc do
          begin
          Active := false;
          SQL.Clear;
          {*** be careful for Field pos defined by Draft Table ***}
          SQL.Add('Select c."Prof ID", ''' + NewPlanName +
                  ''', c.Course, c.Prof ');
          SQL.Add('c."Value", c."Section #", c.Compulsory, c.NewModified ');

          SQL.Add('from ' + DraftDBName + ' c');
          SQL.Add('where (c."Prof ID" = ''' + lProfIDs[cProf.ItemIndex]+'''}' ;
          SQL.Add(' and (c."Plan" = ''' + OldPlanName + '''}');
          Active := true;

          tblProspecDraft.BatchMove(qMisc, batAppend);

          {*** set the NewModified flag to true ***}
          Active := false;
          SQL.Clear;
          SQL.Add('Update ' + DraftDBName + ' c Set NewModified = ''t'' ');
          SQL.Add('where (c."Prof ID" = ''' + lProfIDs[cProf.ItemIndex] +
                  ''') and  ');
          SQL.Add('(c."Plan" = ''' + NewPlanName + ''') ');
          ExecSQL;

          end;


  { end; from mario}
  i := cPlan.Items.Add(NewPlanName);
  cPlan.ItemIndex := i;
  cPlanClick(nil);
  end;


procedure TMainFrm.ModifyPlanName(ProfID, OldPlan, NewPlan: string);
var idx: integer;
    oName: string;
begin
   {*** modify this plan name in draft db ***}
   with qMisc do
   begin
       Active := false;
       SQL.Clear;
       SQL.Add('Update ' + DraftDBName + ' a Set a.''Plan'' = ''' +
               NewPlan + ''', a.NewModified = ''t'' ');
       SQL.Add('Where (a."Prof ID" = ''' + profid + ''') and ');
       SQL.Add('(a.''Plan'' = ''' + OldPlan + ''')');
       ExecSQL;
       Active := false;
   end;

   {*************************************************************************
    * now we have to keep track of this modification so when the user records
    * this plan we can use the old plan name to delete associated records
    * in proper tables
    *
    * only keep track of original plan name and latest new name
    *************************************************************************}
    with NewPlanNameList do
    begin
       idx := IndexOf(OldPlan);
       if idx >= 0 then
       begin
           {*** only update the new plan name, keep original name ***}
           oName := Values[OldPlan];
           Strings[idx] := NewPlan;
           Values[NewPlan] := oName;
       end
       else
       begin
           Add(NewPlan);
           Values[NewPlan] := OldPlan;
       end;
    end;


  InformUser('Changed plan name in Draft table', lForDebug);
```

```
    InformUser('Plan name was changed from ' + OldPlan + 'to ' + NewPlan,
               iInform);

    {*** modify this plan name in PlanPref db   ???ASK???***}

    {*** modify this plan name in ProsPec  db   ???ASK???***}
end;


procedure TMainFrm.FillGridAddProf;
begin
   lblTitle.Caption := 'List of Professors from the Faculty';
   with Query do
   begin
       Active := false;
       SQL.Clear;
       SQL.Add('Select Prof, p."Prof ID", Load from Prof p');
       Active := true;
       Fields[0].DisplayWidth := 20;
       Fields[1].DisplayWidth := 5;
   end;
   InformUser('Filled grid with records from Prof', iForDebug);

   {*** give user possibility to add/modify/delete a record from grid ***}
   GridPage2.ReadOnly := false;
end;


procedure TMainFrm.FillGridAddPlan;
var i: longint;
begin
   { here we have to query both Courtabl and DraftDB, so we use to queries }
   lblTitle.Caption := 'Plan from Professor''s previous years';


   {*** !!!! it would know how to <<join>> two tables it would be better !!! }

   {*** query CourTabl ***}
   InformUser('Querying CourTabl for plan records', iForDebug);
   with Query do
   begin
       Active := false;
       SQL.Clear;
       SQL.Add('Select c."Acad yr", Sum(WF) from CourTabl c, Prof p');
       SQL.Add('where c."Prof ID" = p."Prof ID" and');
       SQL.Add('p."Prof ID" = ''' + lProfIDs[cProf.ItemIndex] + '''');
       SQL.Add('group by c."Acad yr"');
       Active := true;
       TFloatField(Fields[1]).DisplayFormat := '#0.00';
       Fields[0].DisplayLabel := 'Plan (Yr)';
       Fields[0].DisplayWidth := 15;
       Fields[1].DisplayLabel := 'Section #';

       {*** add these records in the string grid ***}
       SGridPage2.RowCount := 1 + RecordCount;
       SGridPage2.ColWidths[1] := Fields[0].DisplayWidth * 8;
       i := 1;
       while not eof do
       begin
           SGridPage2.Cells[1,i] := Fields[0].DisplayText;
           SGridPage2.Cells[2,i] := Fields[1].DisplayText;
           next;
           Inc(i);
       end;
       Active := false;
   end;


   {*** query DraftDB ***}
   InformUser('Querying Draft for plan records', iForDebug);
   with Query do
   begin
       Active := false;
       SQL.Clear;
       SQL.Add('Select c."Plan", Sum(c."Section #") from ' + DraftDBName + ' c');
       SQL.Add('where c."Prof ID" = ''' + lProfIDs[cProf.ItemIndex] + '''');
       SQL.Add('group by c."Plan"');
       Active := true;
       TFloatField(Fields[1]).DisplayFormat := '#0.00';
       Fields[0].DisplayLabel := 'Plan (Yr)';
       Fields[0].DisplayWidth := 15;
       Fields[1].DisplayLabel := 'Section #';

       {*** add these records in the string grid ***}
       SGridPage2.RowCount := 1 + RecordCount;
       while not eof do
       begin
           SGridPage2.Cells[1,i] := Fields[0].DisplayText;
           SGridPage2.Cells[2,i] := Fields[1].DisplayText;
           next;
           Inc(i);
       end;
       Active := false;
   end;
   SGridPage2.visible := true;
   GridPage2.visible := false;
end;
```

```
procedure TMainFrm.FillGridAddFacCourses;
begin
   lblTitle.Caption := 'List of Courses to be offered next year';
   with Query do
   begin
       Active := false;
       SQL.Clear;
       SQL.Add('Select * from Cours');
       Active := true;
       Fields[0].DisplayWidth := 20;
       Fields[1].DisplayWidth := 5;
       Fields[2].Visible := true;
   end;
   InformUser('Filled Grid with records from Cours', iForDebug);

   {*** give user possibility to add/modify/delete a record from grid ***}
   GridPage2.ReadOnly := false;
end;


procedure TMainFrm.FillGridAddCourseProfs; {(Sender: TObject);}
   var InputString: string;
begin
   InputString:= InputBox('Enter course code', 'Please enter the course code'+ chr(13) +
                          '(format xxxx, Ex: ADM1700 or ADX5310)', 'ADM');
   lblTitle.Caption := 'Professors having taught a course';
   with Query do
   begin
       Active := false;
       SQL.Clear;
       SQL.Add('Select p.Prof, Sum(WF) from CourTabl c, Prof p');
       SQL.Add('where c."Prof ID" = p."Prof ID" and');
       SQL.Add('c.Course = ''' + InputString + '''');
       SQL.Add('group by p.Prof');
       Active := true;
       TFloatField(Fields[1]).DisplayFormat := '#0.00';
       Fields[0].DisplayLabel := 'Professor';
       Fields[0].DisplayWidth := 15;
       Fields[1].DisplayLabel := 'Section #';
   end;
   InformUser('Filled Grid with records from CourTabl and Prof', iForDebug);
end;

procedure TMainFrm.FillGridAddProfCourses;
begin
   lblTitle.Caption := 'List of past courses taught by Professor';
   with Query do
   begin
       Active := false;
       SQL.Clear;
       SQL.Add('Select Course, Sum(WF) from CourTabl c, Prof p');
       SQL.Add('where c."Prof ID" = p."Prof ID" and');
       SQL.Add('p."Prof ID" = ''' + lProfIDs[cProf.ItemIndex] + '''');
       SQL.Add('group by Course');
       Active := true;
       TFloatField(Fields[1]).DisplayFormat := '#0.00';
       Fields[0].DisplayLabel := 'Course';
       Fields[0].DisplayWidth := 15;
       Fields[1].DisplayLabel := 'Section #';
       {*Fields[2].Visible := true;*}
   end;
   InformUser('Filled Grid with records from CourTabl and Prof', iForDebug);

end;


procedure TMainFrm.qProspecCalcFields(DataSet: TDataset);
var str: string;
begin
   if IgnoreMods1nDraft then Exit;   {*** added this for Edit from Plan bug ***;

   with tblCours do
   begin
       if not Active then Active := true;
       if FindKey([DataSet.FieldByName('Course').AsString]) = true then
       begin
          PDCompulsory.AsBoolean := FieldByName('Compulsory').AsBoolean;
       end
   end;

end;

procedure TMainFrm.QueryBeforeDelete(DataSet: TDataset);
begin
   {*** advise user then delete references to prof/course in other databases ***;
   InformUser('Deleted Record from Grid', iForDebug);
end;

procedure TMainFrm.QueryBeforeEdit(DataSet: TDataset);
begin
   {*** advise user then update references to prof/course in other databases ***}
   MessageDlg('Don''t forget to advise user and update other dbs',
              mtInformation, [mbOK, mbCancel], 0);
   InformUser('Updated record from Grid', iForDebug);
end;

procedure TMainFrm.cProfKeyPress(Sender: TObject; var Key: Char);
var s:     string;
```

```
    idx:   integer;
begin
   cProf.tag := 1; {*** flag to indicate that user typed a prof name ***}
   if (Key = chr(13)) or (Key = chr(9)) then
   begin
       cProfValidate;
   end;
end;

procedure TMainFrm.cProfExit(Sender: TObject);
begin
   if cProf.tag = 1 then
       cProfValidate;
end;


{*************************************************************************
 * called when user types in the combo box cProf:  it checks to see if the
 * prof is already in the list: if so, simply simulates a Click event else
 * it checks if this prof exists in the Prof db.  If so it adds it in the list
 * and proper db and simulates a Click event.
 *************************************************************************}
procedure TMainFrm.cProfValidate;
var s:     string;
    idx:   integer;
begin
   s := cProf.Text;
   with qMisc do
   begin
       InformUser('Searching for prof ' + s + ' from table Prof' , iForDebug);
       Active := false;
       SQL.Clear;
       SQL.Add('Select * from Prof where Prof = ''' + s + '''');
       Active := true;

       if RecordCount > 0 then
       begin
           InformUser('Found prof ' + s, iForDebug);
           cProf.tag := 0; {*** indicates prof is valid, user didn't typed it *}
           idx := cProf.Items.IndexOf(s);
           if idx < 0 then
           begin
               {*** add prof in the combo list ***}
               idx := cProf.Items.Add(s);
               iProfIDs.Insert(idx, FieldByName('Prof ID').AsString);

               {*** now add him in the draft db ***}
               with tblProfProsPec do
               begin
                   Append;
                   FieldByName('Prof').AsString := s;
                   Post;
               end;
               InformUser('Created new record for Prof ' + s, iForDebug);
           end;

           cProf.ItemIndex := idx;
           cProfClick(NULL);
       end
       else
       begin
           MessageBeep(0);
           MessageDlg('Prof [' + s + '] does not exist!' + chr(13) +
                      'Please type another name', mtInformation,
                      [mbOK], 0);
           cProf.SetFocus;
       end;
       Active := false;
   end;
end;


procedure TMainFrm.cPlanKeyPress(Sender: TObject; var Key: Char);
var s:     string;
    idx:   integer;
begin
   cPlan.tag := 1; {*** flag to indicate that user typed a plan name ***}
   if (Key = chr(13)) or (Key = chr(9)) then
   begin
       cPlanValidate(false);
   end;
end;

procedure TMainFrm.cPlanExit(Sender: TObject);
begin
   if cPlan.tag = 1 then
   begin
       cPlanValidate(false);
   end;
end;


{*************************************************************************
 * called when user types in the combo box cPlan:  it checks to see if the
 * plan is already in the list: if so, simply simulates a Click event else
 * it adds it in the list and proper db and simulates a Click event.
 *************************************************************************}
procedure TMainFrm.cPlanValidate(FromPage2: boolean);
```

```
var s, oldPlan, prof, profid:  string;
    idx:                       integer;
    exists:                    boolean;
begin
   s := cPlan.Text;

   cPlan.tag := 0; {*** indicates plan has been treated, user didn't typed it *}
   idx := cPlan.Items.IndexOf(s);

   prof    := cProf.Text;
   profid  := lProfIDs[cProf.ItemIndex];
   oldPlan := tblProspecDraft.FieldByName('Plan').AsString;

   if idx < 0 then
   begin
       with PlanDlg do
       begin
           Pages.PageIndex := 0;
           if not FromPage2 then
           begin
               {*** we must check if plan exists in other dbs ***}
               with qMisc do
               begin
                   InformUser('Checking if plan exists in CourTabl', iForDebug);
                   Active := false;
                   SQL.Clear;
                   SQL.Add('Select a."Prof ID" from CourTabL a');
                   SQL.Add('Where (a."Prof ID" = :leprof) and ');
                   SQL.ADD('(a."Acad yr" = :leplan)');
                   ParamByName('leprof').AsString := profid;
                   ParamByName('leplan').AsString := s;
                   Active := True;
                   exists := (RecordCount > 0);
                   if exists then
                       InformUser('Plan was found', iForDebug)
                   else
                       InformUser('Plan was not found', iForDebug);

                   Active := false;
               end;
               lblPlanExists.visible       := exists;
               lblPlanDontExist.visible    := not exists;
               btnUsePrevOne.enabled       := exists;
               ShowModal;
           end
           else
               ModalResult := btnNewPlan.ModalResult;

           if ModalResult = mrCancel then
           begin
               cPlan.SetFocus;
               cPlan.Text := oldPlan;
               cPlan.SelectAll;
               exit;
           end
           else
           if ModalResult = btnNewPlan.ModalResult then
           begin
               {*** add plan in the combo list ***}
               idx := cPlan.Items.Add(s);

               {*** now add it in the draft db ***}
               with tblProspecDraft do
               begin
                   if FieldByName('Plan').IsNull then
                   begin
                       Edit;
                       InformUser('Editing plan in draft table', iForDebug);
                   end
                   else
                   begin
                       Append;
                       FieldByName('Prof').AsString    := prof;
                       FieldByName('Prof ID').AsString := profid;
                       InformUser('Adding plan in draft table', iForDebug);
                   end;
                   FieldByName('Plan').AsString := cPlan.Text;
                   FieldByName('NewModified').AsBoolean := true;
                   Post;
               end;
           end
           else
           if ModalResult = btnModName.ModalResult then
           begin
               ModifyPlanName(profID, oldplan, s);
               cPlan.Items[cPlan.Items.IndexOf(oldPlan)] := cPlan.Text;
               idx := cPlan.Items.IndexOf(cPlan.Text);
           end
           else
           if ModalResult = btnUsePrevOne.ModalResult then
           begin
               AddNewPlan(s, oldPlan);
               idx := cPlan.Items.IndexOf(s);
           end;
       end;
   end
   else
   begin
```

```
        {*** user has typed a name of a plan that is already in the list ***}
        InformUser('Plan already in the list', iForDebug);
        with PlanDlg do
        begin
            Pages.PageIndex := 1;
            ShowModal;

            if ModalResult = mrCancel then
            begin
                cPlan.SetFocus;
                cPlan.Text := oldPlan;
                cPlan.SelectAll;
                exit;
            end
            else
            if ModalResult = btnSelectOtherPlan.ModalResult then
            begin
                {*** simply select this other plan from combo box ***}
                cPlan.ItemIndex := idx;
                cPlanClick(NULL);
            end
            else
            if ModalResult = btnModName2.ModalResult then
            begin
                {*** delete other plan and rename current one ***}
                if DeletePlanRecs([DraftDBName], ProfId, s, true) = true then
                begin
                    idx := cPlan.Items.IndexOf(s);
                    if idx >= 0 then cPlan.Items.Delete(idx);

                    ModifyPlanName(ProfID, OldPlan, s);
                    cPlan.Items[cPlan.Items.IndexOf(oldPlan)] := cPlan.Text;
                    idx := cPlan.Items.IndexOf(cPlan.Text);

                    InformUser('Deleted other plan and renamed this one',
                               iForDebug);
                end;
            end;
        end;
    end;

    cPlan.ItemIndex := idx;
    cPlanClick(NULL);

    if not FromPage2 then
        cPlan.SetFocus;
end;


procedure TMainFrm.btnDelProfClick(Sender: TObject);
var profid, prof, msg2:  string;
    msgrc:               word;
    i, ntables:          integer;
    tables:                array[1..4] of string;
begin
    {*** initialize the tables array ***}
    tables[1] := DraftDBName;
    tables[2] := 'Prospec';
    tables[3] := 'PlanPref';
    tables[4] := 'Prof';

    {*** retreive prof name to be deleted ***}
    if Pages.PageIndex = 0 then
    begin
        profid  := lProfIDs[cProf.ItemIndex];
        prof    := cProf.Text;
        ntables := 3;
        msg2    := '.';
        InformUser('Will delete prof from tables Draft, Prospec and PlanPref',
                   iForDebug);
    end
    else
    begin
        profid  := Query.FieldByName('Prof ID' ).AsString;
        prof    := Query.FieldByName('Prof' ).AsString;
        ntables := 4;
        msg2    := ' as well as all corresponding entries' +
                   chr(13) + 'in next year''s prof list';
        InformUser('Will delete prof from tables Draft, Prospec, PlanPref and Prof',
                   iForDebug);
    end;

    {*** prompt the user if he really wants to do this ***}
    msgrc := MessageDlg('All prof''s plans will be erased' + msg2  + chr(13) +
                        'Do you really want to delete all data for prof ' +
                        prof + '?',
                        mtConfirmation, [mbYes, mbNO], 0);

    if msgrc = mrYes then
    begin
        for i := Low(tables) to ntables do
        begin
            with qMisc do
            begin
                InformUser('Deleting Prof ' + prof + ' from ' + tables[i], iInform);
                Active := false;
                SQL.Clear;
                SQL.Add('Delete from ' + tables[i] + ' a ');
```

```
                SQL.Add('Where a."Prof ID" = ''' + profid + '''');
                ExecSQL;
            end;
        end;

        i := cProf.Items.IndexOf(prof);
        if i >= 0 then
        begin
            cProf.Items.Delete(i);
            lProfIDs.Delete(i);
            InformUser('Prof ' + prof + ' has been deleted.', lInform);
            tblProspecDraft.Refresh;
            FillGridAddProf;  {added by jmt}
        end;
    end
    else
    begin
        InformUser('Prof ' + prof + ' has not been deleted.', lInform);
    end;
end;


procedure TMainFrm.btnRemCourseClick(Sender: TObject);
var course, msg2:  string;
    msgrc:         word;
    i, ntables:    integer;
    cond2:         string;
    tables:        array[1..3] of string;
begin
   {???? Question:  do we remove the course from CourTabl and CourTabln also?}

   {*** Initialize the tables array ***}
   tables[1] := DraftDBName;
   tables[2] := 'Prospec';
   tables[3] := 'Cours';


   {*** retreive course name to be deleted ***}
   if Pages.PageIndex = 0 then
   begin
       course  := tblProspecDraft.FieldByName('Course').AsString;
       ntables := 1;

       {*** erase course for this plan and prof only ***}
       cond2   := ' and (a."Plan" = ''' + cPlan.Text + '''' +
                  ' and (a."Prof ID" = ''' + lProfIDs[cProf.ItemIndex] + ''' ';
       msg2    := 'Do you really want to remove course from current plan?';
       InformUser('Will remove Course from Draft table', lForDebug);
   end
   else
   begin
       course  := Query.FieldByName('Course').AsString;
       ntables := High(tables);
       cond2   := ''; {*** erase all occurence of this course ***}
       msg2    := 'Do you really want to remove course ' + course +
                  'from not only' + chr(13) +
                  'next year''s course list but also from every other' +chr(13)+
                  'teacher''s plans?';
       InformUser('Will delete course from tables Draft, Prospec and Cours',
                  lForDebug);
   end;

   {*** prompt the user if he really wants to do this ***}
   msgrc := MessageDlg(msg2, mtConfirmation, [mbYes, mbNO], 0);

   if msgrc = mrYes then
   begin
       for i := Low(tables) to ntables do
       begin
           InformUser('Deleting course ' + course + ' from ' + tables[i],
                      lInform);
           with qMisc do
           begin
               Active := false;
               SQL.Clear;
               SQL.Add('Delete from ' + tables[i] + ' a ');
               SQL.Add('Where "Course = ''' + course + '''' + cond2);
               ExecSQL;
           end;
       end;
       InformUser('Course ' + course + ' has been deleted.', lInform);
       tblProspecDraft.Refresh;
       tblProspecDraft.Edit;
       tblProspecDraft.FieldByName('NewModified').AsBoolean := true;
       tblProspecDraft.Post;
       FillGridAddFacCourses; {added by jmt}
   end
   else
   begin
       InformUser('Course has not been deleted', lInform);
   end;
end;


procedure TMainFrm.btnDeleteClick(Sender: TObject);
begin
   case Pages.Tag of
```

```
        1: btnDelProfClick(Sender);
        3: btnRemCourseClick(Sender);
    end;
end;


procedure TMainFrm.btnDelPlanClick(Sender: TObject);
var profid, plan:  string;
    i:             integer;
    tables:        array[1..3] of string;
begin
    {*** Initialize the tables array ***}
    tables[1] := DraftDBName;
    tables[2] := 'Prospec';
    tables[3] := 'PlanPref';

    {*** retreive plan and prof name for plan to be deleted ***}
    plan   := cPlan.Text;
    profid  := lProfIDs[cProf.ItemIndex];

    InformUser('About to delete current plan', lInform);
    if DeletePlanRecs(tables, ProfId, Plan, true) = true then
    begin
        i := cPlan.Items.IndexOf(plan);
        if i >= 0 then cPlan.Items.Delete(i);
        if cPlan.Items.Count > 0 then
        begin
            cPlan.ItemIndex := 0;
            cPlanClick(nil);
        end;
        InformUser('Deleted plan from tables Draft, Prospec and PlanPref',
                   lForDebug);
    end;
end;

procedure TMainFrm.btnSaveClick(Sender: TObject);
begin
    with Query do
    begin
        if State in [dsEdit, dsInsert] then
        begin
            Post;
            InformUser('Information was saved.', lInform);
        end;
    end;
end;

procedure TMainFrm.btnFirstRecClick(Sender: TObject);
begin
    GridPage2.DataSource.DataSet.First;
end;

procedure TMainFrm.btnLastRecClick(Sender: TObject);
begin
    GridPage2.DataSource.DataSet.Last;
end;

procedure TMainFrm.btnRecordPlanClick(Sender: TObject);
begin
    RecordPlan(lProfIDs[cProf.ItemIndex], cPlan.Text);
end;

procedure TMainFrm.RecordPlan(TheProfID, Plan: string);
var sumOfSect, load:   real;
    ReLoad, SaveIt:    boolean;
    msgbtn:            integer;
    i:                 integer;
    IncLoad:           boolean;
    s, cPlanName:      string;
begin

    {*** retreive the sum of Section # for this plan ***}
    InformUser('Calculating number of Sections for this plan', lInform);
    with qMisc do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('Select Sum(a."Section #") from ' + DraftDBName + ' a where ');
        SQL.Add('(a."Prof ID" = :leprof) and (a."Plan" = :leplan)');
        ParamByName('leprof').AsString := TheProfID;
        ParamByName('leplan').AsString := Plan;
        Active := true;
        sumOfSect := Fields[0].AsFloat;
    end;

    {*** retrieve the course load for this prof ***}
    InformUser('Retreeving max. teaching load for professor', lInform);
    with qMisc do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('Select Load from Prof a where ');
        SQL.Add('(a."Prof ID" = :leprof)');
        ParamByName('leprof').AsString := TheProfID;
        Active := true;
        load := Fields[0].AsFloat;
        Active := false;
    end;
```

```
{*** now we're ready to do the checking for saving this plan ***}
Reload  := false;   {* flag to reload all the original info for this prof *}
SaveIt  := true;    {* flag to save this plan *}
IncLoad := false;   {* do we have to increase the prof load *}

{*** is sumOfSect > load ***}
if sumOfSect > load then
begin
    InformUser('Number of sections exceeds the teaching load attributed',
               iInform);
    msgBtn := Application.MessageBox(
        'The number of sections exceeds the teaching load attributed.' +
        chr(13) +
        'Overload courses/sections should be treated at a later stage.' +
        chr(13) + chr(13) +
        'Do you want to continue and record the plan even if the number.' +
        chr(13) +
        'of sections exceeds the teaching load attributed?',
        'TEACHING LOAD EXCEEDED',
        mb_YesNo);

    if msgBtn = IDYES then
    begin
        SaveIt  := true;
    end
    else
    begin
        Reload := true;
        SaveIt := false;
    end;
end;

{*** is sumOfSect < load ***}
if SumOfSect < load then
begin
    InformUser('Plan has fewer sections than the load attributed', iInform);
    msgBtn := Application.MessageBox('Do you want to proceed with recording a plan' +
              chr(13) +'that has fewer sections than the load attributed?',
              'NUMBER OF SECTIONS LESS THAN LOAD', mb_YesNO);
    if msgBtn = IDYES then
    begin
        SaveIt := true;
    end
    else
    begin
        Reload := true;
        SaveIt := false;
    end;
end;

{*** should we save the plan ***}
if SaveIt then
begin
    InformUser 'Preparing for recording plan', iInform);
    msgBtn := Application.MessageBox('Before deciding whether to record plan' + chr(13) +
            'would you like to view the number of compulsory' + chr(13) +
            'courses taught by the professor in the past?',
            'VIEW NUMBER OF COMPULSORY COURSES TAUGHT', mb_YesNO);
    if msgBtn = IDYes then
    begin
        with ProfCoursesDlg do
        begin
            ShowModal;
            SaveIt := (ModalResult = btnRecordPlan.ModalResult);
        end;
    end;}

    if SaveIt then
    begin
        {*** first find the original plan name for this plan ***}
        with NewPlanNameList do
        begin
            i := IndexOf(Plan);
            if i >= 0 then
                oPlanName := Values[Plan]
            else
                oPlanName := Plan;
        end;


        {*** delete all records for this plan in Prospec and PlanPref ***}
        InformUser('Deleting old records in Prospec and PlanPref', iInform);
        DeletePlanRecs(['Prospec', 'PlanPref'], TheProfID,
                       oPlanName, false);

        {*** copy all recs for this plan from DraftDB to above ones **}
        with qMisc do
        begin
            InformUser('Coying all new records for this plan to proper tables',
                       iInform);
            Active := false;
            SQL.Clear;
            SQL.Add('Select * from ' + DraftDBName + ' a where ');
            SQL.Add('(a."Prof ID" = :leprof) and (a."Plan" = :leplan)' );
            ParamByName('leprof').AsString := TheProfID;
            ParamByName('leplan').AsString := Plan;
            Active := true;
            First;
```

```
            tblProspec.Active := true;

            {*** update PlanPref ***}
            InformUser('Updating Plan Preference table', iInform);
            with tblPlanPref do
            begin
                Active := true;
                FieldDefs.Update;
                Append;
                for i := 0 to FieldDefs.Count - 1 do
                begin
                    s := qMisc.FieldByName(Fields[i].FieldName).AsString;
                    Fields[i].Assign(
                            qMisc.FieldByName(Fields[i].FieldName) );
                end;
                Post;
                Active := false;
            end;

            InformUser('Updating Prospec table', iInform);
            while not eof do
            begin
                {*** update Prospec ***}
                with tblProspec do
                begin
                    Append;
                    for i := 0 to FieldCount - 1 do
                    begin
                        Fields[i].Assign(
                                qMisc.FieldByName(Fields[i].FieldName) );
                    end;
                    Post;
                end;

                next;
            end;
            tblProspec.Active := false;
        end;

        {*** set NewModified field for the associated records to false ***}
        with qMisc do
        begin
            Active := false;
            SQL.Clear;
            SQL.Add('Update ' + DraftDBName + ' a ');
            SQL.Add('Set NewModified = ''f'' where');
            SQL.Add('(a."Prof ID" = :leprof) and (a."Plan" = :leplan)' );
            ParamByName('leprof').AsString := TheProfID;
            ParamByName('leplan').AsString := Plan;
            ExecSQL;
        end;

        {*** now Increase Prof Load if we have to ***}
        if IncLoad then
        begin
            with qMisc do
            begin
                InformUser('Increasing Prof Load in Prof database', iInform);
                Active := false;
                SQL.Clear;
                SQL.Add('Update Prof a Set a."Load" = ' + FloatToStr(sumOfSect));
                SQL.Add('where (a."Prof ID" = :leprof)' );
                ParamByName('leprof').AsString := TheProfID;
                ExecSQL;
            end;
        end;

    end;
    InformUser('Plan has been successfully recorded', iInform);
  end
  else
  begin
    InformUser('Plan has not been recorded', iInform);
  end;
end;


function TMainFrm.DeletePlanRecs(const tables: array of string;
                                 Prof, Plan: string; Prompt: boolean): boolean;
var i:     integer;
    msgrc: Word;
    msg2:  string;
begin
  if Prompt then
  begin
    {*** check to see if table Prospec in list of tables ***}
    msg2  := '';
    for i := Low(tables) to High(tables) do
    begin
        if UpperCase(tables[i]) = 'PROSPEC' then
        begin
            msg2 := 'The plan will be permanently deleted from the main database.' + chr(13);
            break;
        end;
    end;
    msgrc := MessageDlg(msg2 +'Do you really want to delete plan ' + plan
                        + ' for prof ' + cProf.Text +  '?',
                mtConfirmation, [mbYes, mbNO], 0);
```

```
    end
    else
    begin
        msgrc := mrYes;
    end;

    if msgrc = mrYes then
    begin
        for i := Low(tables) to High(tables) do
        begin
            InformUser('Deleting plan ' + Plan + ' from table ' + tables[i], iInform);
            with qMisc do
            begin
                Active := false;
                SQL.Clear;
                SQL.Add('Delete from ' + tables[i]  + ' a ');
                SQL.Add('Where (a."Prof ID" = ''' + Prof + ''') and ' +
                        '(a."Plan" = ''' + Plan + ''')');
                ExecSQL;
            end;
        end;
    end
    else
    begin
        InformUser('Plan was not deleted.', iInform);
    end;

   Result := (msgrc = mrYes);
end;

procedure TMainFrm.btnValidateSchedClick(Sender: TObject);
var msgrc: word;
{var    tables: array[1..4] of string;
       i:      integer;
       buffer: PChar;}

   begin
      with qMisc do
      begin
          Active := false;
          SQL.Clear;
          SQL.Add('Select Distinct a."Prof ID", a."Prof", a."Plan" ' +
                  'from ' + DraftDBName + ' a');
          SQL.Add('Where NewModified = ''t'' Order By a."Prof ID", ' +
                  'a."Prof", a."Plan"');
          Active := true;

          if RecordCount = 0 then
          begin

          if MessageDlg('The generation of the Faculty roster may take several minutes;' + chr(13) +
                        'do you want to perform it?',
                         mtConfirmation,[mbYes, mbNo],0)
                          = mrYes then
             begin

             with tblProspecDraft do active := false;
             with Validate do
                begin
                ShowModal;
                  (MessageDlg('Test; before or after?',mtConfirmation,[mbYes, mbNo],0);
                 if ModalResult = mrCancel then
                   begin
                      Exit;
                   end;
                   '

                end;

                {--------------}

     if MessageDlg('Would you like to add the new constructed plan in the' + chr(13) +
                   'temporary database of prospective plans (data\prodraft.db)?' + chr(13) +
                   'If this is what you want, you will have the opportunity' + chr(13) +
                   'to record these plans to prospec.db on the main entry form.',
                    mtConfirmation,[mbYes, mbNo],0) = mrYes then
                    BEGIN
                    screen.cursor := crhourglass;
                    WITH tblpostopt do
                       BEGIN
                       (active := true; )
                       tblpostopt.first;
                       WHILE NOT tblpostopt.eof DO
                          BEGIN
                          WITH tblProsPecDraft do
                             BEGIN
                             active := True;
                             if tblpostopt.FieldByName('NewModified').AsBoolean then
                              BEGIN
                                append;
                                tblProsPecDraft.FieldByName('Prof ID').AsString :=
                                   tblpostopt.FieldByName('Prof ID').AsString;
                                tblProsPecDraft.FieldByName('Plan').AsString :=
                                   tblpostopt.FieldByName('Plan').AsString;
                                tblProsPecDraft.FieldByName('Course').AsString :=
                                   tblpostopt.FieldByName('Course').AsString;
                                tblProsPecDraft.FieldByName('Section #').AsString :=
                                   tblpostopt.FieldByName('Section #').AsString;
```

```
                                tblProsPecDraft.FieldByName('NewModified').AsString :=
                                  tblpostopt.FieldByName('NewModified').AsString;
                              END;
                              active := True;
                             END;
                          tblpostopt.Next;
                          END;

                      END;
                  screen.cursor := crArrow;
                  END;

{--------------}


          with tblProspecDraft do active := true;
          end;
      end

      else
      begin

         msgrc := MessageDlg('There is/are ' + IntToStr(RecordCount) +
                             ' modified plan(s) that are not yet saved!' +
                             chr(13) +
                             'You should record them before continuing.',
                              mtInformation, [mbOK], 0);
      end;
   end;
   if msgrc = mrYes then Application.Terminate;

   {*** initialize the tables array ***}
    tables[1] := 'Cours';
   tables[2] := 'Prospec';
   tables[3] := 'PlanPref';
   tables[4] := 'Prof';
   for i := 1 to 4 do
   begin
       tblToExport.TableName := tables[i] + '.db';
       tblProspecASCII.TableName := tables[i] + '.txt';
       bmExportProspec.Execute;
   end;
   MessageDlg('4 tables have been exported in:' + chr(13) +
              DataBase.Params[0],
               mtInformation, [mbOK], 0);
   buffer := StrAlloc(300);
   StrPCopy(buffer, ExtractFilePath(Application.ExeName) + 'Validate.exe');
   ShellExecute(MainFrm.Handle, 'open',
                buffer, '',
                '.\', SW_SHOW);

   StrDispose(buffer);

end;

procedure TMainFrm.InformUser(msg: string; DebugFlag: integer);
begin
   if not   DebugFlag = iForDebug) and (not cbDebug.Checked) ) then
   begin
       StatBarTimer.Enabled := false;
       LastMsg := msg;
       StatusBar.Caption := msg;
       StatusBar.Refresh;

       {** use a timer so the user can have the chance to see the last message **}
       if cbDebug.Checked then
           Delay(3000)
       else
           StatBarTimer.Enabled := true;
   end;
end;
procedure TMainFrm.btnLastMsgClick(Sender: TObject);
begin
   StatusBar.Caption := 'Last message was: ' + LastMsg;
   StatusBar.Refresh;
end;

procedure TMainFrm.btnAboutClick(Sender: TObject);
begin
   AboutDlg.ShowModal;
end;

procedure TMainFrm.StatBarTimerTimer(Sender: TObject);
begin
   StatBarTimer.Enabled := false;
end;

procedure TMainFrm.tblProspecDraftSetDataChange(Sender: TObject;
  Field: TField);
var exists: boolean;
begin
   cPlanPref.Text := tblProspecDraft.FieldByName('Value').asstring;

   if IgnoreModsInDraft then exit;

   if (Field <> nil) and (TDataSource(Sender).State in [dsInsert, dsEdit]) then
   begin
       IgnoreModsInDraft := true;
```

```
        PDNewModified.AsBoolean := true;
        IgnoreModsInDraft := false;

        if (Field.FieldName = PDCourse.FieldName) then
        begin
            {*** check to see if course exists in Cours table ***}
            with qMisc do
            begin
                InformUser('Checking if changed course exists in Cours', iForDebug);
                Active := false;
                SQL.Clear;
                SQL.Add('Select a."Course" from Cours a');
                SQL.Add('Where a.Course =  ''' + Field.AsString + '''');
                Active := True;
                exists := (RecordCount > 0);
                if exists then
                    InformUser('Course was found', iForDebug)
                else
                begin
                    InformUser('Course was not found', iForDebug);
                    MessageDlg('Course ' + Field.AsString + ' does not exist in the database!'
                               + chr(13) +
                               'Please change it or manually remove it from the plan.',
                               mtInformation, [mbOK], 0);
                end;

                Active := false;
            end;
        end;
    end;
end;

procedure TMainFrm.btnRecordSomePlansClick(Sender: TObject);
var i, idx: integer;
    profID, plan: string;
begin
    {*** Post any pending modifications into the Draft table ***}
    if tblProspecDraft.State in [dsEdit, dsInsert] then
        tblProspecDraft.Post;

    {*** query the Draft table for all modified records ***}
    with qMisc do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('Select Distinct a."Prof ID", a."Prof", a."Plan" ' +
                'from ' + DraftDBName + ' a');
        SQL.Add('Where NewModified = ''t'' Order By a."Prof ID", ' +
                'a."Prof", a."Plan"');
        Active := true;

        if RecordCount = 0 then
        begin
            Active := false;
            MessageDlg('No plans have been modified', mtInformation, [mbOK], 0);
            Exit;
        end
        else
        begin
            {*** fill the Plans List in the PlanList dialog ***}
            ModPlansDlg.List.Items.Clear;
            while not eof do
            begin
                idx := ModPlansDlg.List.Items.Add(Format('%-20s %-15s %-10s',
                                    [Fields[0].AsString, Fields[1].AsString,
                                     Fields[2].AsString]));
                ModPlansDlg.List.Selected[idx] := true;
                next;
            end;
        end;
    end;

    { display a list of all modified plans so user can select ones to save }
    with ModPlansDlg do
    begin
        ShowModal;
        if ModalResult = mrCancel then
        begin
            Exit;
        end
        else
        begin
            for i := 0 to List.Items.Count - 1 do
            begin
                if List.Selected[i] or (ModalResult = mrAll) then
                begin
                    profId  := Copy(List.Items[i], 1, 20);
                    RTrim(profID);
                    plan    := Copy(List.Items[i], 38, 10);
                    RTrim(plan);
                    InformUser('Recording plan ' + plan + ' of prof ' + profID,
                               iInform);
                    RecordPlan(profid, plan);
                end;
            end;
        end;
    end;
end;
```

```
procedure TMainFrm.tblProspecDraftBeforeCancel(DataSet: TDataset);
var msgrc: Word;
begin
(*** shouldn't not use this...may cause user to forget a previous change
     which will not be saved if he selects yes...

   msgrc := MessageDlg('Do you want me to mark this plan as unmodified?',
                       mtConfirmation, [mbYes, mbNo], 0);
   if msgrc = mrYes then PDNewModified.AsBoolean := false;
*)
end;


procedure RTrim(var s: string);
var i: integer;
begin
   i := Length(s);
   while (i>0) and (s[i] = ' ') do
   begin
       Dec(i);
       s[0] := Char(i);
   end;
end;


procedure TMainFrm.cPlanPrefClick(Sender: TObject);
var Pref: string;
begin
   Pref := cPlanPref.text;

   (*** Post any pending modifications into the Draft table ***)
   if tblProspecDraft.State in [dsEdit, dsInsert] then
       tblProspecDraft.Post;

   with qMisc do
   begin
       Active := false;
       SQL.Clear;
       SQL.Add('Update ' + DraftDBName + ' a ');
       SQL.Add('Set NewModified = ''t'', a."Value" = ' + Pref);
       SQL.Add('Where a."Prof ID" = "' +
               tblProspecDraft.FieldByName('Prof ID').AsString + '"');
       SQL.Add('and a."Plan" = "' +
               tblProspecDraft.FieldByName('Plan').AsString + '"');
       ExecSQL;
   end;
   tblProspecDraft.refresh;
end;

procedure TMainFrm.btnGeneratecoursesClick(Sender: TObject);
type
    RowArray   = array [1 .. 450] of real;
    RowPointer = ^RowArray;
var
  Present: TDateTime;
  currYear, Month, Day : Word;
  t : RowPointer;
  varIndex : integer;
  actIndex, numCrs : integer;
  numSect: real;
  actArray : actArrayType;

begin
  Present:= Now;
  DecodeDate(Present, currYear, Month, Day);
  if  MessageDlg('This procedure will calculate the number of sections planned for the' + chr(13) +
                 'academic year '+ inttostr(currYear) + '-' + inttostr(currYear+1) +
                 ' by consulting the master database.' + chr(13) +
                 'WARNING: This will replace all the records of the Course Database.' + chr(13) +
                 'Is this what you want to do ?',
                  mtConfirmation,[mbYes, mbNo],0)= mrYes
   then  with tblProfcrs do
     BEGIN                 {find prospective courses in COURTABL }
     screen.cursor := crhourglass;
     active := True;
     tblProfcrs.First;
     numCrs:= 0;
     new(t);
     WHILE (NOT tblProfcrs.eof) DO
        BEGIN   {next record of COURTABL }
        IF (
             (
                (tblProfcrs.FieldByName('Year').AsFloat = currYear+1)  and
                (tblProfcrs.FieldByName('Session').AsString = 'WINTER')
             )
             or
             (
                   (tblProfcrs.FieldByName('Year').AsFloat = currYear)  and
                   (
                     (tblProfcrs.FieldByName('Session').AsString = 'FALL')
                      or
                     (tblProfcrs.FieldByName('Session').AsString = 'SUMMER')
                   )
             )
          )
          THEN
            BEGIN      { only prospective courses }
            IF not findAct( tblProfcrs.FieldByName('Course').AsString, actIndex, actArray, numCrs)
```

```
                THEN
                   BEGIN  {extend actArray}
                     numCrs := numCrs + one;
                     t^[numCrs] := 0.;
                     actArray[numCrs] := tblProfcrs.FieldByName('Course').AsString;
                   END;   {extend actArray}
              numSect:=one;
              sect2num (tblProfcrs.FieldByName('Course').AsString,numSect,currYear);
              t^[actIndex] := t^[actIndex] + numSect;
              {MessageDlg('Course:' + actArray[actIndex] + chr(13) +
                          'Numcrs:' + floattostr(a^[actIndex]^[one]),
                           mtInformation,[mbok],0);           }
              END;       { only prospective courses }
           tblProfcrs.Next;

           END;     { next record of COURTABL }

           WITH tblCours do
                   BEGIN  { write cours.db}
                   active := False;
                   Readonly := False;
                   active := True;
                   tblCours.First;
                   WHILE NOT tblCours.eof DO
                      BEGIN  { refresh cours.db}
                      delete;
                      END;   { refresh cours.db}
                   for actIndex:= 1 to numCrs do
                      BEGIN  { write a[] in cours.db}
                      append;
                      FieldByName('Course').AsString := actArray[actIndex];
                      {FieldByName('Req').AsFloat := t^[actIndex]}
                      if (     (int(0.5+100*t^[actIndex]) - 100*t^[actIndex] < -0.1) or
                               (int(0.5+100*t^[actIndex]) - 100*t^[actIndex] > 0.1)
                         )
                        then  FieldByName('Req').AsFloat := int(1+100*t^[actIndex])/100.
                        else  FieldByName('Req').AsFloat := t^[actIndex];

                      { tblCours.next;}
                      end;    { write a[] in cours.db}
                   active := False;
                   Readonly := True;
                   active := True;
                   END;   { write cours.db}
        END;       { find prospective courses in COURTABL}
     screen.cursor := crArrow;
     MessageDlg('The number of courses was successfully calculated from the' + chr(13) +
                'master database and recorded in the Course database',
                mtInformation,[mbOk],0);
end;

procedure TMainFrm.btnCopyClick(Sender: TObject);
begin
  if not DirectoryExists('u:\users\tessler\c95-97\') then
     MessageDlg('The source directory does not exist.' + chr(13) +
                'Please make sure that you are connected to the' + chr(13) +
                'local area network of the Faculty and try again.',
                mtError,[mbOk],0)
     else
        begin
        screen.cursor := crhourglass;
        Winexec('copyfile.bat',sw_minimize);
        screen.cursor := crArrow;
        MessageDlg('The master database (courtabl.db) and the teams database (teams.db)' + chr(13) +
                   'were successfully copied in the data repository of the application.',
                   mtInformation,[mbOk],0);
        end;

end;

end.
```