Comput. & Ops. Res. Vol. 13, No. 4, pp. 387–394, 1986
Printed in Great Britain.

0305-0548/86 $3.00 + 0.00

THE PROBLEM OF ASSIGNING STUDENTS TO COURSE SECTIONS IN A LARGE ENGINEERING SCHOOL*

GILBERT LAPORTE† and SYLVAIN DESROCHES‡
Ecole des Hautes Etudes Commerciales de Montréal, 5255 avenue Decelles, Montréal, Canada H3T 1V6

(*Received September* 1984; *revised June* 1985)

Scope and Purpose—Problems of course and examination scheduling in schools and universities have received considerable attention in the Operational Research literature [see the recent survey by Vincke (Timetabling in Belgium: a survey. Paper presented at TIMS XXVI, Copenhagen, June 1984)]. This paper describes the solution of a particular problem related to course scheduling in some universities. Complete course schedules are often made available to the students at the time of registration. Typically, these schedules give the room number, instructor and time for each section of each course. Students use this information to select their courses. A computer program then allocates students to course sections in order to respect various criteria. The authors designed and implemented such a program in a large Canadian Engineering School.

Abstract—In some universities, complete course schedules are made available to students at the time of registration. Typically these schedules give the room number, instructor and time for each session. Students use this information to select their courses. A computer program then allocates students to course sections so as to provide students with satisfactory schedules, to equalize roughly the number of students in all sections of the same course and to respect room capacities. The authors designed and implemented such a program at the Ecole Polytechnique de Montréal, one of Canada's leading Engineering Schools.

INTRODUCTION

Problems of course and examination scheduling in schools and universities have received considerable attention in the Operational Research literature (see the surveys of Klein [1], Vincke [2], Schmidt and Ströhlein [3] and Carter [4]). This can be explained in part by the fact that the nature of these scheduling problems varies significantly according to their context. This paper describes a particular problem related to course scheduling encountered in a large Canadian Engineering School and, with some variants, in other higher education institutions [5–9].

In some universities, complete course schedules are made available to students at the time of registration. Typically, these schedules give the room number, instructor and time for each section of each course. Students use this information to select their courses. A computer program then allocates students to course sections while respecting various criteria. The authors designed and implemented such a program at the Ecole Polytechnique de Montréal (EPM), one of Canada's leading Engineering Schools.

Twice a year, some 2800 EPM undergraduate students make their course selection from a master schedule. This schedule is built by taking into account room availabilities and professors' preferences. The number of sections of each course is determined from enrollment forecasts. About one-third of these students take courses offered in only one section and present no major interest to us. All other students have at least one course given in more than one section. Students specify their course selection, but no section preferences. A difficulty arises from the fact that courses are offered on two campuses located about 2 miles apart.

*The authors are grateful to the Natural Sciences and Engineering Research Council (grant A4747) and to the Ecole Polytechnique de Montréal for their support. They also thank an anonymous referee for his helpful comments.

†Gilbert Laporte obtained his Ph.D. in Operational Research from the London School of Economics in 1975. He is now a full Professor at the Ecole des Hautes Etudes Commerciales de Montréal. His main research interests lie in the areas of scheduling, location and routing. He is the author or the co-author of several publications in OR journals.

‡Sylvain Desroches holds a B.Sc. in Computer Science from the University of Montreal. He has worked for several years as a programmer and as an analyst on various projects including automatic translation, and the scheduling of courses and examinations in universities. He is now employed as an analyst in a telecommunications firm.

Our aim is to allocate students to course sections so as to satisfy the following criteria:

Strict constraint

(i) student course selections must be respected;

EPM criteria

(ii) section enrollments should be balanced, i.e. all sections of the same course should have roughly the same number of students;
(iii) room capacities should not be exceeded;

Student criteria

(iv) *conflicts* should be avoided (students having to take two courses at the same time);
(v) students must have a lunch break between noon and 14:00 (in periods 4 or 5);
(vi) days with too many or too few class periods should be avoided;
(vii) class spreads on the same day should be minimized (i.e. the number and length of free periods embedded between two classes should be minimized);
(viii) the number of moves between the two campuses should be minimized;
(ix) *infeasible moves* (i.e. classes scheduled in two consecutive periods and in two different campuses) should be avoided.

It is important to mention that the distinction between EPM criteria and student criteria is not clear cut: the school is as interested as the students in avoiding conflicts and in providing satisfactory timetables. We mainly use this distinction to simplify the design and the description of our algorithm.

These criteria, which cannot all be labelled unequivocally as *objectives* or as *constraints*, are sometimes conflictual; they are not equally important and cannot be achieved fully in all but trivial cases. They are described in the next section.

The authors are aware of a limited number of studies on this type of problem. Macon and Walker [8] and Busam [6] present algorithms which take into account section balancing and student preferences for given sections or instructors. But these preferences are not considered by Abell [5] and by Winters [9] whose algorithm often produces solutions containing a high proportion of infeasible schedules. Colijn's algorithm [7], on the other hand, produces conflict-free schedules, keeps section enrollments balanced and satisfies various side conditions (such as limited enrollment in some specified sections).

Our algorithm appears to be more general and more flexible than those published in previous studies. One of its interesting features is the weight it gives to the overall quality of student schedules. Section preferences are not considered in the program developed for EPM but could easily be taken into account, if desired.

CONSTRAINTS AND OBJECTIVES

Some of the criteria enumerated in the introduction will be treated as constraints, while others will be considered as objectives.

Only the first criterion is treated as an absolute constraint by the algorithm. Surprisingly, some requirements such as avoiding conflicts and not exceeding room capacities are not treated as strict constraints: they do, however, carry considerable weight. There are two main reasons for this: first, the total enrollment for a given course may well exceed the total room capacity for that course; second, conflicts are frequently tolerated by EPM if they are not excessive. Infeasible situations generated by the algorithm are dealt with manually in a subsequent phase: some room reallocations are made, and in the case of conflicts, students may be advised to make a different course selection.

The remaining criteria can mostly be described as objectives: balancing sections, providing pleasant schedules and minimizing the number of moves between the two campuses. The algorithm attempts to construct student schedules which attain these objectives simultaneously.

In order to assess the quality of the schedule of student i on day l, we define a penalty index P_{il}. The

Table 1. Penalties

Occurrence	Weight currently used	Example 1	Example 2	Example 3
Conflict	1000	0	0	0
No lunch break in periods 4 or 5	30	0	0	30
Number of *holes*, i.e. free time intervals between two classes	2 per hole	4	0	2
Total length of holes	2 per hour	6	0	2
Campus change during a hole	60	60	0	60
Campus change during a class	1200	0	0	1200
Number of classes in the day				
0	0			
1	12			
2	6			
3	0		0	
4	0			
5	0	0		
6	0			
7	6			6
8	12			
9	18			
Total		70	0	1300

total penalty for student i is defined as

$$P_i = \sum_{l=\text{Monday}}^{\text{Friday}} P_{il}. \tag{1}$$

To optimize student objectives, P_i should be minimized for each i.

The P_{il}s are obtained by assigning a non-negative weight to various unpleasant aspects of the schedule of student i on day l and by adding these weights. The weights currently used are provided in Table 1. The entries of the last three columns refer to the examples given below.

Examples

Here are examples of three different schedules. The following notation is used:

0: no class scheduled,
i: class scheduled at campus i ($i = 1, 2$).

	Period number								
	1	2	3	4	5	6	7	8	9
Schedule 1	1	0	0	1	0	2	2	2	0
Schedule 2	0	1	1	1	0	0	0	0	0
Schedule 3	1	2	2	2	2	0	1	1	0

The various penalties used are somewhat arbitrary in the sense that they strongly reflect the authors' experience and judgement of what constitutes a good or a bad schedule on a given day. However, the registrar's office agreed with these penalties and the quality of the schedules produced generated no complaints from the students.

Balancing sections requires shifting students between the various sections of the same course. As this objective frequently conflicts with that of providing pleasant schedules for students, it can only be partially attained. The technical details of this procedure are described in the next section. As we shall see, the algorithm we have developed is a fine balancing act between satisfying student objectives and those of the university.

ALGORITHM

The algorithm is divided into three main phases:

Phase 1: constructing student schedules without taking into account section enrollments and room capacities;
Phase 2: balancing section enrollments;
Phase 3: respecting room capacities.

Phase 1: constructing student timetables

Step 1: Students with trivial timetables (a selection of courses offered in only one section each) are scheduled. Students with the same course selection are identified; if there are k identical students, these are aggregated into one *k-student* (k may be equal to 1).

Step 2. Set the iteration index t equal to 1. Identify for each k-student i an optimal or near optimal set of sections S_i^t, i.e. a selection providing a low penalty P_i^t. This is done by means of a branch and bound tree.

Tree structure. If the k-student has selected r courses, the maximum level of the tree will be r. The jth level of the tree corresponding to a given node has one branch for each way of choosing a section for the jth course of i.

Nodes. Each node created at level r is a *full node* and corresponds to a complete schedule; all other nodes are *partial nodes* and correspond to incomplete section selections.

Bounding. At each full node, the sum of penalties associated with the complete schedule is a valid lower bound on P_i^*, the cost of the optimal schedule for i. At each partial node, we define a lower bound as the sum of penalties associated with moves taking place between two consecutive or nonconsecutive periods, conflicts and missed lunch breaks. Other types of penalties are not taken into account at this stage as they can only be computed from a complete schedule.

Branching. We use a LIFO rule: This has the advantage of finding a feasible schedule quickly while using little computer memory. At each node of the tree, the first branch which is explored is the one corresponding to the most empty section.

Stopping rule. The search ends when all branches of the tree have been explored according to the usual branch and bound rules or after 5000 full nodes have been created.

Phase 2: balancing sections

At the end of phase 1, all students have been scheduled, but many sections may be unbalanced. In phase 2, the algorithm attempts to balance sections without causing too much deterioration in students schedules and by spreading the penalty increase as fairly as possible among all students. To achieve this aim, the algorithm performs four iterations. If P_i^t represents the schedule cost of k-student i at the end of iteration t, then the new schedule must be such that

$$P_i^t \leqslant P_i^{t-1} + \delta_t, \qquad t \geqslant 2, \tag{2}$$

where δ_t is a parameter of the algorithm. After some experimentation, and considering the penalties given in Table 1, we have opted for the following values:

$$\delta_2 = 5, \qquad \delta_3 = 10, \qquad \delta_4 = 20, \qquad \delta_5 = 70.$$

More discussion is required before we can proceed to the step by step description of phase 2. Consider S_i^t, the course selection of k-student i at the end of iteration t. Let $j \in S_i^t$ and

$$m_j = \text{the average number of students in all sections of the course to which section } j \text{ corresponds,} \tag{3}$$

$$s_j^t = \text{the number of students in section } j \text{ at the end of iteration } t. \tag{4}$$

Sections to which i belongs have a total disequilibrium defined by

$$D(S_i^t) = \sum_{j \in S_i^t} (s_j^t - m_j)^2. \tag{5}$$

The algorithm will attempt to find a new section assignment S_i^{t+1} for i which will reduce the disequilibrium. Removing a k-student i from all his sections decreases the disequilibrium by

$$\begin{aligned} \Delta_1^t &= \sum_{j \in S_i^t} (s_j^t - m_j)^2 - \sum_{j \in S_i^t} (s_j^t - m_j - k)^2 \\ &= 2k \sum_{j \in S_i^t} (s_j^t - m_j) - k^2 |S_i^t|. \end{aligned} \tag{6}$$

Reassigning i to S_i^{t+1} increases the disequilibrium by

$$\begin{aligned} \Delta_2^{t+1} &= \sum_{j \in S_i^{t+1}} (s_j^t - m_j + k)^2 - \sum_{j \in S_i^{t+1}} (s_j^t - m_j)^2 \\ &= k^2 |S_i^{t+1}| + 2k \sum_{j \in S_i^{t+1}} (s_j^t - m_j). \end{aligned} \tag{7}$$

We therefore seek a section assignment S_i^{t+1} for which

$$\Delta_2^{t+1} < \Delta_1^t. \tag{8}$$

We now describe the procedure used in phase 2.

Step 3. Consider in turn all k-students created in step 1. If $k > 3$, disaggregate the k-student into several k-students with $k = 2$ or 3. This is done in order to avoid moving too many students at a time between sections of the same course (and thus taking the risk of not being able to reduce the disequilibrium); of course, all k-students could have been split into 1-students, but then the computational effort required in phase 2 might have been excessive.

Execute steps 4 and 5 for $t = 2, 3, 4, 5$

Step 4. Consider a k-student i. If $s_j^t \leqslant m_j$ for all $j \in S_i^t$, consider the next i and repeat this step. Otherwise, proceed to step 5.

Step 5. By using the branch and bound procedure described in step 2, identify a new section assignment S_i^{t+1} such that (2) and (8) hold and that for all sections $j \in S_i^{t+1}$, $s_j^t \leqslant m_j$. However, the four most empty sections for a given course are always considered as suitable candidates for inclusion in S_i^{t+1}. Consider the next i and go to step 4.

Phase 3: respecting room capacities

At the end of phase 2, some room capacities may be exceeded. Let c_j be the room capacity of section j. Then we require that

$$s_j^6 \leqslant c_j, \qquad \text{for all } j. \tag{9}$$

The total *room overflow* associated with a particular section assignment S_i^t is equal to

$$W(S_i^t) = \sum_{j \in S_i^t} \max(0, s_j^t - c_j). \tag{10}$$

As in phase 2, we seek a reassignment which will this time reduce the room overflow.

Step 6. Consider a k-student i. If $W(S_i^t) = 0$, consider the next i and repeat this step. Otherwise, set $t = 6$, $\delta_6 = 400$ and proceed to step 7.

Step 7. This step is similar to step 5 except that the following condition is added: Section j is not considered as a suitable candidate for inclusion in S_i^{t+1} if $s_j^t \geq c_j$. However, if all sections of a given course overflow, then they are all permitted. Consider the next i and go to step 6.

ADDITIONAL FEATURES

To simplify the discussion, we have omitted from the description made in preceding section two features which were in fact incorporated in the program written for EPM:

(i) courses given every second week (this is in fact restricted to some lab sessions);
(ii) twinning: for lab sessions, some students may wish to be paired with a given friend; such requests are traditionally granted by EPM.

These two particularities are dealt with in the branch and bound procedure. It would also be relatively easy to take into account at that stage student preferences for certain sections, in the computation of the (P_i^t)s.

IMPLEMENTATION AND COMPUTATIONAL RESULTS

At time of writing, the program has been used three times at EPM, between January 1984 and January 1985. In these three instances, very satisfactory schedules both from the students' and from the school's point of view were produced in a relatively short time. This was viewed by EPM as a distinct improvement over the situation that prevailed in previous years when a sluggish computer program ran for hours to produce a schedule without any optimization attempt. However, no data on past schedules are available, and so it is impossible to measure the increase in student well-being that resulted from the use of our program.

In order to illustrate some elements of the above discussion, we present the results obtained in January 1984. Table 2 summarizes the main statistics for that registration period.

The 2136 "nontrivial students" can be grouped into 1818 k-students. The distribution is given in Table 3.

In phase 1, the algorithm examined an average of 500.34 schedules (full nodes) per k-student. The distribution is given in Table 4.

Table 2. Basic statistics for January 1984

Number of courses	326
Number of sections	518
Number of students with trivial course selections	663
Number of students with nontrivial course selections	2136
Total number of students	2799

Table 3. Distribution of k-students

k	No. of k-students
1	1658
2	106
3	28
4	10
5	3
6	4
7	2
8	2
9	1
10	1
11	1
15	1
28	1
Total	1818

Table 4. Number of schedules (full nodes) examined in phase 1

No. of schedules	Frequency
1– 9	456
10– 99	511
100– 999	561
1000–1999	116
2000–2999	51
3000–3999	34
4000–4999	14
5000	75
Total	1818

Table 5. Statistics on the algorithm

Iteration t	δ_t	Average number of full nodes examined	Average P_i^t	Average section disequilibrium	Time
1	—	500.34	719.21	19.67	5.97
2	5	137.10	718.81	12.48	1.65
3	10	151.70	720.36	6.37	1.72
4	20	159.58	720.71	3.96	1.80
5	70	178.12	722.28	2.55	1.55
6	400	46.30	723.32	2.69	0.35
Total					13.04

This indicates that an optimal schedule was found for at least 95.8% of all k-students (corresponding to 96.4% of all students). These schedules had an average cost P_i^1 of 719.21. Out of the 2136 students considered, 737 or 34.5% had a schedule with a cost exceeding 1000, meaning that they probably had at least one conflict or one illegal move. This proportion of "infeasible" schedules may look high at first glance. However, the high proportion of optimal schedules indicates that the fault lies not with the algorithm but with the hand-made master schedule or with the students' selections of courses. A closer examination of this question revealed that around 75% of all conflicts were due to the fact that after their first or second year, many students are lagging behind in their normal curriculum and often have to take courses in two different years; the master schedule can hardly accommodate all possible situations. The remaining 25% of conflicts were caused by students wishing to take two optional courses overlapping in the master schedule. Here again the master schedule cannot be such that all selections of optional course are feasible for all students. As it turns out, about half the conflicts are tolerated by EPM; in all other cases, students are urged to make a different course selection. Given the situation that prevails at EPM, it is unlikely that all conflicts can be avoided. However, the authors feel that a more satisfactory result could be achieved if an algorithm similar to the one described in this paper was used to construct the master schedule as well as optimal section assignment.

After the first iteration, the average section disequilibrium was equal to 19.67. After five extra iterations, this value was brought down to 2.69, whereas the average schedule penalty increased to only 723.32 and still contained the same proportion of "infeasible" schedules. After the last iteration, the average section disequilibrium was very low at 2.69 and only five sections out of 518 had an overflow exceeding nine (the maximum overflow was equal to 26).

The main statistics are presented in Table 5. The column "time" refers to the CPU time (in minutes) on the University of Montreal CYBER 173 computer.

CONCLUSION

The algorithm described in this paper was perceived by the EPM registrar's office as a very welcome and useful tool. Apart from constructing a very satisfactory assignment of students to course sections, it revealed that about one-third of all students made "conflictual choices": there were many instances in which the *optimal* schedule produced after the first iteration still contained conflicts or illegal moves. However, from the point of view of section equilibrium and room capacity restrictions, the solution produced is almost perfect.

From a more general standpoint, the multiphase approach developed in this paper appears well suited to this type of multiobjective problem and may be appropriate to other contexts.

REFERENCES

1. D. Klein, The application of computerized solution techniques to the problem of timetabling in high schools and universities. M.B.A. dissertation, Concordia University (1983).
2. P. Vincke, Timetabling in Belgium: a survey. Paper presented at TIMS XXVI, Copenhagen (1984).
3. G. Schmidt and T. Ströhlein, Timetable construction—an annotated bibliography. *Comput. J.* **23**, 307–316 (1980).
4. M. W. Carter, A survey of practical applications on examination scheduling. Working paper 83-07, Department of Industrial Engineering, University of Toronto (1983).
5. V. A. Abell, Purdue academic student scheduling. Mimeographed, Purdue University, Lafayette, Ind. (1965).
6. V. A. Busam, An algorithm for class scheduling with section preference. *Commun. ACM* **10**, 567–569 (1967).
7. A. Colijn, A sectioning algorithm. *Infor* **11**, 210–225 (1973).
8. N. Macon and E. E. Walker, A Monte Carlo algorithm for assigning students to classes. *Commun. ACM* **9**, 339–340 (1966).
9. W. K. Winters, A scheduling algorithm for a computer assisted registration system. *Commun. ACM* **14**, 166–171 (1971).