ELSEVIER

European Journal of Operational Research 137 (2002) 512–523

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Production, Manufacturing and Logistics

Design and implementation of a course scheduling system using Tabu Search

Ramon Alvarez-Valdes [a,*], Enric Crespo [b], Jose M. Tamarit [a]

[a] *Department of Statistics and Operations Research, Faculty of Mathematics, University of Valencia, Doctor Moliner 50, 46100 Burjassot, Valencia, Spain*

[b] *Department of Financial and Mathematical Economy, University of Valencia, Valencia, Spain*

Received 20 January 2000; accepted 12 January 2001

Abstract

Building a course timetable is a difficult and lengthy task which universities devote a large amount of human and material resources to every year. We have developed a computer package to solve this problem. The program runs on a PC and the user may set the objectives and parameters from among a wide range of possibilities. It has a user-friendly interface for the user to input the relevant data and obtain the corresponding results. The optimization process is based on a set of heuristic algorithms. The core is a Tabu Search procedure for which several strategies have been developed and tested in order to get a fast and powerful algorithm. The first tests of the package have produced satisfactory results. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Timetabling; University; Heuristics; Tabu Search

1. Introduction

The problem of building a university timetable consists of assigning both starting and finishing times and an adequate room to each of the lessons in the planned courses. This assignment has to satisfy existing material constraints as well as other conditions related to the way in which each university wants to organize its teaching. This problem must be solved by the university administration every year, or even term, and it involves a large amount of human and material resources.

The main difficulty is related to the size of the problem. It involves a large number of students, teachers, courses and rooms, linked in many ways by objectives and conditions, and therefore each solution procedure must take into account very large numbers of variables and constraints. Moreover, the structure of the timetables varies from country to country, due to differences in the academic systems. Even within one academic system there are major differences among universities depending on the particular ways in which teaching is organized.

* Corresponding author. Tel.: +34-96-3864308; fax: +34-96-3864735.

E-mail address: ramon.alvarez@uv.es (R. Alvarez-Valdes).

PII: S0377-2217(01)00091-1

For these reasons, the problem of building university timetables has been extensively studied in Operations Research literature over the last 25 years. The solution techniques have been evolving alongside the developments in mathematics and computer sciences. The survey by Carter and Laporte [4] describes several subproblems which appear in university timetabling: *Course Timetabling*, *Class-Teacher Timetabling*, *Student Scheduling*, *Teacher Assignment* and *Classroom Assignment*. Each real life problem contains a combination of several of them, but will always include one of the first two subproblems.

The solution techniques range from graph coloring to complex metaheuristic algorithms, including LP formulations and heuristics tailored to the specific problem at hand. The more efficient procedures which have appeared in recent years are based on metaheuristics. Hertz [13,14] has applied Tabu Search techniques, Dowsland [6] and Elmohamed et al. [7] use simulated annealing and several authors like Burke et al. [3], Corne, Ross et al. [5] and Paetcher et al. [17] have developed procedures based on variants of genetic algorithms. A different approach is Constraint Logic Programming, as in Guéret [12]. Finally, some commercial packages have been developed such as ACT [16], SAPHIR [8], Neeps and Tatties [18], reflecting the special characteristics of each academic system.

Carter and Laporte [4] distinguish course timetabling problems as being either master timetable or demand driven systems. The main difference between them lies in the sequence in which the subproblems are solved. In master timetabling the assignment of times to lessons is done first and then the students choose courses and sections from published timetables. In demand driven systems, the students first choose from a list of courses and from this information the number of sections of each course is determined and times are assigned to the lessons of these sections. A section is each group into which a course is divided according to size or language considerations.

In the Spanish university system we have to build a master timetable. Nevertheless, we think that information about students' individual course choices would be helpful in the process of building the timetable. This information may come from preregistration or can be taken from the students' registrations of previous years. With this information we solve two subproblems: assigning times to lessons and assigning students to sections. We do it in an iterative way, where the solution of one problem is the starting point for the other one, until no further improvement is obtained. The problem of assigning students to sections is an auxiliary problem because that assignment is not going to be used in practice, but it helps to improve the results of the timetable being built.

In this paper, we describe the development of the program HORARIS. This program builds a timetable which is suitable for the characteristics of the Spanish university system, though it takes particular account of the special features of the University of Valencia.

In Section 2, we present the elements of the problem, data, objectives and constraints. The complete model appears in Section 3, where we describe its two subproblems and the relationship between them. We focus our attention on the problem of building the timetable, because the students' assignment problem has been solved elsewhere [1]. Section 4 develops the solution procedure, specially the Tabu Search algorithm which is its core. In Section 5, we describe the characteristics of the school in which we have tested our procedures, the numerical results obtained and the conclusions about the best strategies to be implemented in the computer program. Finally, in Section 6, we explain the features of the HORARIS program and its first real life test.

2. Data, objectives and constraints

A course timetabling problem has the following elements:

- A set of *courses* $\mathscr{A} = \{A_1, \ldots, A_j, \ldots, A_s\}$. Each course has a fixed number of hours per week which can be taught in one session or in several sessions during the week.
- According to the number of students and other considerations, each course A_j can be divided into a set of *sections* $\mathscr{S}_j = \{S_{j1}, S_{j2}, \ldots, S_{jn_j}\}$ to be taught separately. For instance, our

university is in a bilingual region (Spanish and Catalan) and some courses are offered in both languages.

- A set of *lessons* $\mathscr{L} = \bigcup_{j,k} L_{jk}$ where $j = 1, \ldots, s$ and $k = 1, \ldots, n_j$. $L_{j,k}$ are the lessons of section S_{jk}. These lessons may have different lengths, even in the same course or section.
- A set of *classes* $\mathscr{C} = \{C_1, C_2, \ldots, C_n\}$. A class is a set of courses taken, basically, by the same group of students. These classes are designed by the planners, in line with the academic structure of the degrees offered and their past experience of students' choices. Each class will have quite a stable set of students and it is therefore very important that their lessons will not be taught simultaneously.
- A set of *teachers* $\mathscr{T} = \{T_1, T_2, \ldots, T_t\}$. Each section of each course will have a teacher previously assigned to it.
- A set of *rooms* $\mathscr{R} = \{R_1, R_2, \ldots, R_r\}$, which can be of different types, including computer rooms and laboratories for specific subjects. Each room has a fixed capacity.
- A *week*, made up of a set of *periods* $\mathscr{P} = \{P_1, P_2, \ldots, P_p\}$, and divided into days. Each day has the same number of periods of equal length. The breaks and the non-teaching periods must be specified. The lessons lasting for more than one period cannot be interrupted by a break.
- It is also possible to have some *preassignments* or some *forbidden periods* for classes, sections, teachers or rooms.

2.1. Objectives and constraints

When we think of a good timetable we consider several types of conditions to be met though these conditions are not all of the same importance. We distinguish, on the one hand, *hard constraints*, that is, conditions that every acceptable timetable must satisfy. These constraints are:

1. Every lesson has to be assigned to a period or a number of periods, depending on the lesson's length.
2. No teacher can give two simultaneous lessons.
3. Two lessons of the same course cannot be assigned to the same day.
4. No room can be assigned to two simultaneous lessons.
5. The room assigned to a given lesson must be of the required type.
6. All the preassignments and forbidden periods for classes, sections, teachers and rooms must be respected.

Our university has one more type of constraints. In many cases there are theoretical and practical courses in which the student must register simultaneously. Their sections are organized in such a way that sections of the practical course are considered as subsets of sections of the theoretical one. For instance, if there are three sections A, B, C of a theoretical course, it is quite common to have nine sections of the practical course, AA, AB, AC, BA, BB, BC, CA, CB, CC, linked according to the first letter. The University wants a student assigned to section A in the first course to be assigned to section AA or AB or AC in the second one.

On the other hand, there are some conditions that are considered helpful but not essential in a good timetable. The more these conditions are satisfied, the better the timetable will be. We call them *soft constraints* and therefore they will have a weight in the objective function. We group these objectives according to the elements they involve.

1. For classes:
   - Simultaneity of lessons of one class should be avoided, because a group of students have registered for all the courses of the class. This is usually the most important objective for users.
   - Class timetables should be as compact as possible, eliminating idle times for students.
   - The number of moves from one room to another of students belonging to one class should be minimized. Each class should have its lessons assigned to a minimum number of different rooms.
2. For students, taken individually:
   - Simultaneity of lessons of the sections in which he/she is enrolled should be avoided. Note that many students do not belong to a defined class, but choose their own set of courses.
   - Students' individual timetables should be as good as possible. The quality of a timetable

is measured according to criteria such as the number of lessons per day, the idle times between lessons, the moves between rooms, buildings and campuses and the respect for their language preferences.

3. For teachers, their non-desired periods should be avoided.
4. For rooms, the objective is adjusting their capacity to the number of students enrolled in sections assigned to them.
5. The sections of a course should be balanced with regard to the number of students assigned to them.
6. Each university also has its own specific objectives. We have tried to include in our program those which are more important and those which are usual. The objectives explicitly used in our tests are described in Section 5.

3. Problem formulation

The problem can be formulated as follows:

$$\begin{array}{ll} \text{Min} & f(x,y) \\ \text{s.t.} & x \in S_x, \quad y \in S_y. \end{array}$$

Variables x represent the array of assignments of lessons to starting periods and rooms and variables y the vector of assignments of students to sections. The function $f(x,y)$ is the value of each timetable for x and y given. The sets S_x and S_y are the feasible sets according to the hard constraints described above.

The solution procedure decomposes the problem into two components which are solved in successive steps. These components, though solved separately, interact in a global process. For a given course timetable, the problem is that of assigning students to sections which maximize the quality of individual timetables while keeping sections balanced. When the students' assignments are fixed, the problem is building the best course timetable. This strategy follows that of Aubin and Ferland [2].

The steps to follow in the process of Fig. 1 are:

Step 0: Initialization

• Let x^* be a course timetabling feasible solution.

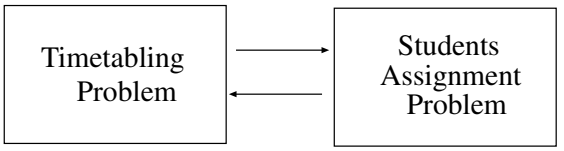

Fig. 1. Relationship between the two subproblems.

Step 1: Student assignment problem

• Given $x = x^*$, determine the best solution y' for the student assignment problem.

• $f^* = f(x^*, y')$. Go to Step 2.

Step 2: Master timetabling problem

• Given $y = y'$, determine the best solution x' for the course timetabling problem

• If $f^* \leqslant f(x', y')$, stop. Otherwise, $f^* = f(x', y')$, $x^* = x'$. Go to Step 1.

When using our program HORARIS, it is possible to build a master timetable without solving the subproblem of the students' assignment to sections, that is, without taking into account the information about the students' choices. This is the strategy usually followed by the planners when building manual timetables.

However, this strategy presents two main drawbacks. On the one hand, the students not choosing the courses defined for a class are not taken into account. On the other hand, for the sections not belonging to a class we do not have any assignment criterion. If we know the students' choices, we can assign these sections to periods which minimize simultaneity conflicts in students' timetables.

For solving the subproblem of students' assignment to sections, we use a procedure developed in 1996 [1] which combines two procedures. First, an algorithm based on the construction of maximal independent sets obtains a set of good timetables for each student. Then, a Tabu Search process looks for a balance in the number of students assigned to the sections of every course, with minimum deterioration of the quality of individual timetables.

Therefore, we focus our attention here on the problem of building a course timetable, considering the student assignment to sections as fixed. In order to give an integer programming formulation, we define the variables:

$$x_{ita} = \begin{cases} 1 & \text{if lesson } i \text{ starts at period } t \text{ in room } a, \\ 0 & \text{otherwise.} \end{cases}$$

Besides the sets defined in Section 2, we need the following elements:

LT_k	Set of lessons of teacher T_k
$\mathscr{D}$	Set of days
d_i	Duration of lesson i
P_l	Periods of day l
p_i	Preassigned period for lesson i
m_{rt}	Number of rooms of type r avalaible at period t
R_r	Set of rooms of type r
LR_r	Lessons requiring rooms of type r
TR	Different types of rooms
$\mathscr{F}$	Set of preassigned lessons
$\mathscr{U}_i$	Set of forbidden periods for lesson i

The problem is:

$$\begin{aligned} \text{Min} \quad & \omega_c f_c(x) + \omega_{st} f_{st}(x, y^*) \\ & + \omega_t f_t(x) + \omega_r f_r(x) + \omega_{sc} f_{sc}(x) \end{aligned}$$

subject to

$$\forall i \in \mathscr{L}, \quad \sum_a \sum_t x_{ita} = 1, \tag{1}$$

$$\forall t \in \mathscr{P}, \ \forall k \in \mathscr{T}, \quad \sum_a \sum_{i \in LT_k} \sum_{\tau=t-d_i+1}^{t} x_{i\tau a} \leqslant 1, \tag{2}$$

$$\forall S_{jk} \in \mathscr{S}, \ \forall l \in \mathscr{D}, \quad \sum_a \sum_{t \in P_l} \sum_{i \in L_{jk}} x_{ita} \leqslant 1, \tag{3}$$

$$\forall a \in \mathscr{R}, \ \forall t \in \mathscr{P}, \quad \sum_i \sum_{\tau=t-d_i+1}^{t} x_{i\tau a} \leqslant 1, \tag{4}$$

$$\forall t \in \mathscr{P}, \ \forall r \in TR, \quad \sum_{a \in R_r} \sum_{i \in LR_r} \sum_{\tau=t-d_i+1}^{t} x_{i\tau a} \leqslant m_{rt}, \tag{5}$$

$$\forall i \in \mathscr{F}, \quad \sum_a x_{ip_i a} = 1, \tag{6}$$

$$\forall i \in \mathscr{L}, \quad \sum_a \sum_{t \in \mathscr{U}_i} \sum_{\tau=t-d_i+1}^{t} x_{i\tau a} = 0. \tag{7}$$

In the objective function x is the array of variables x_{ita} and y^* is the assignment of students to sections. Each objective appears with its corresponding weight ω: ω_c corresponds to classes, ω_{st} to students, ω_t to teachers, ω_r to rooms and ω_{sc} to the specific objectives of centre. Constraints (1)–(7) are the conditions every feasible timetable must satisfy.

This formulation cannot be solved by exact methods. Firstly, the number of variables would be too large. Secondly, some parts of the objective function are non-linear, because they involve the assignment of more than one lesson to periods. That is the case of the simultaneity of lessons of one class or the moves between rooms. If we consider only the relation between pairs of lessons, the problem has a quadratic formulation, as in Ferland and Roy [9]. For some specific objectives, like having the lessons of a course evenly spread over the week, more complex relationships are needed.

4. Solution method

The algorithm for building a course timetable has three phases:

1. Phase I: Constructing an initial timetable.
2. Phase II: Improving the timetable built in Phase I.
3. Phase III: Improving the room assignment.

Building the timetable involves the assignment of lessons to rooms. Nevertheless, it is quite frequent, for reasons of simplicity, for rooms to be assigned as a last step after the assignment of lessons to periods. In HORARIS we have adopted a mixed strategy. In Phases I and II rooms are taken into account and are assigned to lessons, but they do not have a large weight in the objective function. Phase III improves room assignment without changing the assignment of lessons to periods.

In our algorithmic scheme, the Tabu Search algorithm of Phase II is the most important part. Phases I and III prepare and marginally improve

the results of Phase II. When solving a timetabling problem, Phase I is used only once at the beginning of the solution process to produce an initial solution. Phases II and III can be called several times, changing the search parameters and/or the objective function weights, in order to explore different feasible solutions.

4.1. Phase I: Building an initial timetable

The timetable is built by taking the lessons one at a time, first of all those preassigned to periods, then those of sections belonging to a class and finally those of sections not belonging to a class.

Classes are taken sequentially, according to the priority given by the user. Before assigning periods to lessons, we assign to the class a room where the teaching of the class will whenever possible take place, thus trying to minimize the number of moves. Then, we take one section at a time and assign its lessons. The order in which the sections are taken depends on the number and length of a section's lessons. Each lesson is assigned to the period of minimum value of the objective function.

Sections not belonging to a class are taken sequentially, prioritized by the ratio of the total length of lessons to the number of possible periods. The lessons are assigned to the best period according to the objective function and the rooms are chosen trying to adjust their capacities to the number of students of the section.

4.2. Phase II: Improving the initial timetable

In order to design a powerful algorithm we have developed and tested several strategies for each element of Tabu Search [11]. Overall, 52 different combinations of elements have been tried in order to determine the best characteristics of the algorithm to be implemented in the program.

The elements of the Tabu Search procedure are as follows.

4.2.1. The solution space X

The set of solutions satisfying constraints (1)–(7).

4.2.2. The objective function

This is the combination of objectives with weights given by the user as appears in Section 3.

4.2.3. The neighborhood $N(s)$

We have defined three alternative neighborhoods created by the following moves:

- The *simple move* in which a solution $s' \in X$ is a neighbor of solution $s \in X$ if it can be obtained from s by changing the assignment of one lesson l from one period t to another period t'.
- The *swap* or interchange of lessons, in which a solution $s' \in X$ is a neighbor of solution $s \in X$ if it can be obtained by interchanging the periods assigned to two lessons l and l'.
- The *multiswap* or multiple change of lessons (see Fig. 2). We take a lesson l, assigned to periods $t, t+1, \ldots$, and change its assignment to periods $t', t'+1, \ldots$ Then, the lessons $l_1, l_2, \ldots, l_n$ which were assigned to these periods $t', t'+1, \ldots$ are now assigned to $t, t+1, \ldots$

In all the cases a lesson l with d periods of duration can be assigned to period t only if there is an available room in periods $t, t+1, \ldots, t+d-1$. Hence, the move can modify the previous room assignment.

We decided to try increasingly complex moves because it was very difficult to move in the neighborhood defined by the simple move. Quite often good assignments for a lesson, according to the objective function, were impossible due to room shortages. The second type of move, swap, allowed us to overcome this problem, because the lessons can interchange their rooms. Nevertheless, the

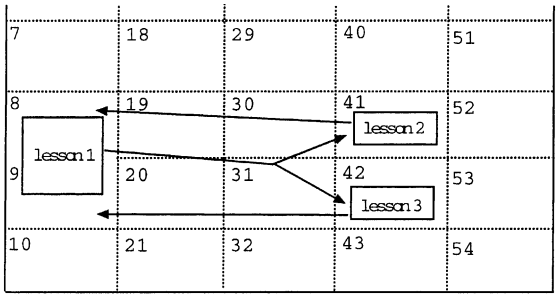

Fig. 2. Multiswap outline.

swap is not powerful enough for situations like these:

- When we try to interchange lessons of different lengths, the periods assigned to the shortest one are not sufficient to accommodate the longest, and therefore some other lessons should be involved in the move.
- When we want to move a theoretical lesson of a section (section A) to the periods assigned to a set of practical lessons (sections AA, AB and AC), we should move all these three lessons in order to avoid conflicts.

The multiswap allows major changes in the solution structure, thus producing a greater diversification in the search process.

4.2.4. The tabu list

The tabu list keeps, for every iteration, the lesson starting the change and its initial period.

We have used lists with a constant length of 24 and dynamically changing lists in which length varies between 6 and 48. The value 24 has been chosen taking as reference the square root of the number of lessons in the problem, as is usual in many implementations of Tabu Search. Nevertheless, additional testing shows that the performance of the algorithm is quite robust to changes in the list length. For dynamically changing list lengths, the value is randomly chosen in the interval $(0.25l, 2l)$, where l is the length of the static list.

4.2.5. The aspiration criterion

If an explored move produces a solution s' with objective function lower than the best current solution, the move is made in spite of its tabu status.

4.2.6. Selection of moves

For a given solution s, it is computationally too expensive to explore its whole neighborhood $N(s)$. We determine which lesson to move by two different methods. In the first one, we choose it randomly from among all the lessons. In the second, we make a list of candidates. If the solution s has some conflicts of simultaneity among the lessons of a class, these lessons will constitute the list. If there are no conflicts of this type, all lessons such that when leaving their assigned periods produce an improvement of the objective function make part of the list. Once the lesson has been chosen, all its possible periods are explored.

The use of a candidate list is an intensification strategy that tries to reduce the search to the moves which are more likely to improve the objective function.

4.2.7. Strategic oscillation

We have developed a strategic oscillation [15] of the search consisting of temporal changes of the objective function. More precisely, for a given number of iterations some of the weights of the components of the objective function are set to zero. Therefore, for some iterations the corresponding objectives are not considered when looking for better solutions.

This oscillation is a diversification strategy which leads the search to unexplored regions in the solution space.

4.2.8. Recovering the best current solution

We have also tested the strategy of recovering the best solution found if the process has not improved it after a given number of iterations. It is an intensification strategy in which we concentrate the search once again in the neighborhood of good solutions.

4.3. Phase III: Improving the room assignment

In previous phases, the assignment of a lesson to a period always involves a feasible room assignment. In the last phase of the procedure we improve the room assignment without changing the assignment of lessons to periods. The objective is to reduce to a minimum the number of room changes for classes.

In our objective function the cost of assigning a room for a lesson has two components: a fixed cost, related to room capacity and the number of students attending the lesson, and a variable cost that accounts for the changes of rooms and depends on the rooms of previous and posterior lessons. This second aspect differs from other existing approaches [4,10]. Therefore, we need a specific algorithm for this case. We decided to implement a simple and fast heuristic procedure

because this last phase only aims to obtain marginal improvements.

The heuristic algorithm has two linked procedures. First, given a lesson with its corresponding room we try to propagate this room to adjacent lessons, that is, we try to use this room in lessons given consecutively to the same class, avoiding moves between rooms. The move we explore consists of changing the room of one of these adjacent lessons, if it improves the objective function. Second, we identify the room used more often by a class and try to assign it to the lessons of that class which have been assigned to other rooms. Again, the move consists of changing the assignment of a room for one of these lessons. The algorithm works sequentially, taking the classes according to a user-defined priority.

5. Numerical results

The University of Valencia, Spain, has 65 000 students divided into four main areas: Social Sciences, Health Sciences, Humanities and Basic and Technical Sciences. Each area has several Faculties or Schools that are responsible for the academic organization, including the course timetables. The area of Social Sciences contains the largest number of students and the Faculties and Schools of this area face the most complex scheduling problems. From among them, the Business School was selected for the computational tests. This School offers a three-year Degree in Business Administration. Course scheduling is done on a semester basis. For instance, in the first semester of the academic year 1997/98 there were 4300 students registered, 84 courses with 724 sections, some of them defining a total of 93 classes. More than 200 teachers gave 993 lessons per week. The school week had 13 periods per day, Monday to Friday, with a lunch break. The School has 24 rooms with an average use of 83%.

There are some specific requirements to be added to the problem. The classes of the first and second year must have two free days dedicated to optional courses. It is convenient for sections of practical subjects linked to a theoretical course not to be simultaneous, in order to allow the same teacher for all of them. The solution would be even better if these sections were consecutive. Finally, for courses with more than one lesson per week, it would be convenient for these lessons to be assigned to the same time every day and to be spread out evenly throughout the week.

The weights of the objective function components appear in Table 1, where we note that avoiding simultaneity conflicts for a class is the most important criterion for the users. In this case, the School is located in one building and so there are no moves between buildings. Also, information about teacher preferences was not available and was therefore not included.

5.1. Analysis of Phase II results

The combination of strategies has produced 48 different implementations of Tabu Search algorithm. When we run them on the test problem described above, we obtain the 48 values of Tables 2–4.

Table 1
Weights of criteria

	Criterion	Element involved	Weight
1	Conflicts of simultaneity	Class	5000
2	Compactness of timetable	Class	100
3	Moves between buildings without break	Class	200
4	Moves between buildings with break	Class	100
5	Changes of room	Class/ room	50
6	Use of the same room	Class	100
7	Simultaneity conflicts	Students	1
8	Forbidden periods	Teacher	100
9	Non-desired periods	Teacher	5
10	Lessons at the same time of the day	Section	10
11	Lessons spread out in the week	Section	10
12	Simultaneity of practical lessons	Section	100
13	Non-consecutive practical lessons	Section	10
14	Excess of students	Room	5
15	Empty seats	Room	1

Table 2
Results of simple move

		With candidate list		Without candidate list	
		Recover	Non-recover	Recover	Non-recover
Static tabu list	Change weights	77 046	77 387	86 842	87 003
	Non-change weights	77 046	77 387	86 529	87 003
Dynamic tabu list	Change weights	77 486	74 845	87 002	86 692
	Non-change weights	75 783	77 863	86 992	87 003

Table 3
Results of swap

		With candidate list		Without candidate list	
		Recover	Non-recover	Recover	Non-recover
Static tabu list	Change weights	63 320	62 836	59 579	65 092
	Non-change weights	64 638	65 232	60 944	66 533
Dynamic tabu list	Change weights	63 399	63 467	59 791	60 057
	Non-change weights	63 607	63 242	59 381	59 439

Table 4
Results of multiswap

		With candidate list		Without candidate list	
		Recover	Non-recover	Recover	Non-recover
Static tabu list	Change weights	65 398	64 764	59 717	58 614
	Non-change weights	66 467	67 036	60 116	61 948
Dynamic tabu list	Change weights	63 213	65 246	57 150	54 801
	Non-change weights	64 341	64 695	58 691	56 950

We have analyzed these results to determine the best combination of strategies. We have used an analysis of variance to evaluate the influence of strategies on the solutions. We consider as significant the factors or interactions with a level of significance (SL) lower than $\alpha = 0.05$.

The type of move is the most influential factor with a $SL < 0.0001$. The average values of the solutions for the simple move, swap and multiswap were 81935.7, 62534.8 and 61821.7, respectively. It is clear that a simple move produces results significantly worse than the other two moves. We therefore decided to eliminate the simple move and study the remaining 32 cases in Tables 3 and 4, in order to avoid possible interferences of those low quality results.

We repeated the analysis of variance for the 32 values and the results show that the small difference in favor of multiswap is not significant ($SL = 0.1455$). Instead, the strategy used with the candidate list is the most influential factor ($SL < 0.0001$)). The average result with a candidate list (64431.3) is clearly worse than that obtained without a candidate list (59925.2).

The type of tabu list is also a significant factor ($SL = 0.0003$). If we use a dynamically changing tabu list length we obtain a better average (61091.9) than by using a static list length (63264.6). The strategic oscillation of the weight vector is also significant ($SL = 0.0361$), with an average of 61652.7 when we oscillate and 62703.7 if we do not. Recovering the best known solution

is not significantly influential ($SL = 0.1902$), but the results are slightly better when we do recover (61859.5 vs 62497.0).

If we look at the interactions between factors, the only interaction that can be considered significant is that between the type of move and the use of a candidate list ($SL = 0.0003$). The combination of multiswap with the strategy of not using a candidate list produces very good results, as can be seen in Fig. 3.

An illustration of the counterintuitive behavior of the candidate list is Fig. 4, which shows the evolution of the solutions of both strategies in an example. Though the implementation with a candidate list finds better solutions in the first iterations, in the long run it is overtaken by the implementation without a candidate list. The reason may be that when working with an aggressive move, as is multiswap, the random choice of the move allows the search to explore the solution space better if a large number of iterations is performed. However, the strategy without a candidate list may take many iterations to obtain solutions without simultaneity conflicts for classes, which are the largest component of the objective function. In the example of Fig. 4, if we use a candidate list, a solution without conflicts is obtained in the first 100 iterations while without a candidate list a conflict-free solution is not obtained in the first 12 000.

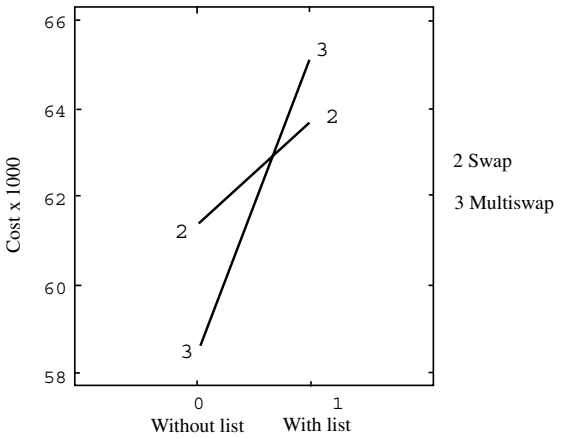

Fig. 3. Interaction between type of move and use of candidate list.

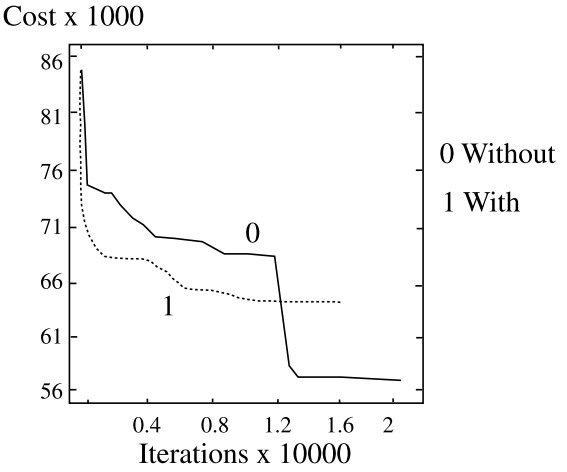

Fig. 4. Evolution of the objective function with and without candidate list.

Therefore in hard problems, or when the user does not allow the algorithm to perform many iterations, we could end up without a solution free of conflicts, despite the fact that they exist.

As that situation would not be acceptable for the user, we designed a mixed strategy. We use a candidate list until a solution without conflicts is found and from that iteration onwards we do not use it. We compared these three strategies fixing the multiswap and dynamic tabu list. The results appear in Table 5.

Doing an analysis of variance once again we found that the use of a candidate list is a significant factor ($SL = 0.0079$). We then performed a Newman–Keuls test of multiple comparisons to find where the significant differences among the three strategies lie. The results of the test show that there is no significant difference between the temporal use (average value 57460.5) and the non-use of the list (average 56898.0), but there are significant differences between these two cases with the continuous use of a candidate list (average 64373.7).

5.1.1. Conclusions for implementing the computer program HORARIS

Taking into account the above results, the computer program was implemented with the following characteristics:

Table 5
Results of the three strategies for the use of candidate list

	With candidate list		Without candidate list		Temporal use of list	
	Recover	Non-recover	Recover	Non-recover	Recover	Non-recover
Change weights	63 213	65 246	57 150	54 801	58 551	57 180
Non-change weights	64 341	64 695	58 691	56 950	58 829	55 282

- The move is multiswap.
- Temporal use of candidate list.
- Tabu list with dynamically changing length.
- Strategic oscillation of weights.
- Recovering the best known solution after a given number of iterations without improving.

5.2. Results of Phase III of room assignment improvement

The average number of room changes before improving the room assignment was 389, slightly more than four changes per class. The improvement process reduces the changes by 4.5%.

The other objective involving the rooms is to concentrate all the theoretical lessons of a class in one room and all its practical lessons in another room. At the end of Phase II, only three lessons per class, on average, did not satisfy this condition. The improvement process reduces this situation by 7%.

The joint improvement in these two objectives, according to their weights in the objective function, is 4.64% in the global value of the solution. This result confirms the usefulness of this fast improvement procedure.

6. The program HORARIS

6.1. Characteristics

The algorithms described above have been implemented in the program HORARIS, coded in C++. The program is designed to run on a PC with a Pentium processor, because this is the usual equipment of School and Faculty administrations.

The interface has been designed on Windows, with three types of screens for inputting data, interacting with the solution procedure and obtaining the results.

The data editing windows allow the user to input and modify the characteristics of the centre (starting and finishing times every day, breaks, length of period), the courses (number and length of their lessons), the sections (forbidden periods), the classes (composition, priority, forbidden periods) and the teachers (forbidden and non-desired periods). Other information involving students' registration and fixed data about courses and teachers are directly read from the corresponding files of the university computing centre.

The windows of interaction with the solution procedure allow the user to modify the weights of the objective function criteria, start the solution process and stop it when it is considered convenient. The program is continuously giving information about the quality of the best current solution. Besides the possibility of building a timetable from scratch, it is also possible to improve a given timetable. In this case, the program skips Phase I and proceeds to Phases II and III.

The output windows show the timetable from several points of view: classes, courses, teachers or rooms. This information can be printed or saved in ASCII files to be used with other programs.

6.2. A test for academic year 1999–2000

The Business School has tested the program for building the timetables for the academic year 1999–2000 and compared them with their manual timetables.

The main problem when building the timetable is planning the third year in which the students

may choose from among several streams of optional subjects, each one leading to a different professional orientation (Business Management, Foreign Commerce, Financial Management, Accounting and Auditing). Some courses may be shared by several streams and students of the second year may also choose them as a complement to their compulsory subjects. Due to the complexity of this problem, the manual timetables do not address it as a whole, but consider each course separately.

The problem was solved by the program HORARIS and the results were considered satisfactory by School administrators, specially in the aspects corresponding to the third year, for which a global solution without simultaneity conflicts for classes was provided.

Acknowledgements

We are very grateful to the anonymous referees for their helpful comments and suggestions that have contributed to clarifying and improving this paper.

References

[1] R. Alvarez-Valdés, E. Crespo, J.M. Tamarit, Assigning students to course sections using Tabu Search, Annals of Operations Research 96 (2000) 1–16.
[2] J. Aubin, J.A. Ferland, A large scale timetabling problem, Computers and Operations Research 16 (1) (1989) 67–77.
[3] E. Burke, J.P. Newall, R.F. Weare, A memetic algorithm for university exam timetabling, in: E. Burke, P. Ross (Eds.), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153, Springer, Berlin, 1996, pp. 241–250.
[4] M.W. Carter, G. Laporte, Recent developments in practical course timetabling, in: E. Burke, M. Carter (Eds.), Practice and Theory of Automated Timetabling II, Lecture Notes in Computer Science, vol. 1408, Springer, Berlin, 1997, pp. 3–19.
[5] D. Corne, P. Ross, Peckish initialisation estrategies for evolutionary timetabling, in: E. Burke, P. Ross (Eds.), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153, Springer, Berlin, 1996, pp. 227–240.
[6] K.A. Dowsland, A timetabling problem in which clashes are inevitable, JORS 41 (10) (1990) 907–918.
[7] M.A.S. Elmohamed, P. Coddington, G. Fox, A comparison of annealing techniques for academic course scheduling, in: E. Burke, M. Carter (Eds.), Practice and Theory of Automated Timetabling II, Lecture Notes in Computer Science, vol. 1408, Springer, Berlin, 1997, pp. 92–112.
[8] J.A. Ferland, C. Fleurent, SAPHIR: A decision support system for course scheduling, Interfaces 24 (2) (1994) 105–115.
[9] J.A. Ferland, S. Roy, Timetabling problem for university as assignment of activities to resources, Computers and Operations Research 12 (2) (1985) 207–218.
[10] C.R. Glassey, M. Mizrach, A decision support system for assigning classes to rooms, Interfaces 16 (5) (1986) 92–100.
[11] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Dordrecht, 1997.
[12] C. Guéret, N. Jussien, P. Boizumault, C. Prins, Building university timetables using constraint logic programming, in: E. Burke, P. Ross (Eds.), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153, Springer, Berlin, 1995, pp. 130–145.
[13] A. Hertz, Tabu Search for large scale timetabling problems, EJOR 54 (1991) 39–47.
[14] A. Hertz, Finding a feasible course schedule using Tabu Search, Discrete Applied Mathematics 35 (1992) 255–270.
[15] J.P. Kelly, B.L. Golden, A.A. Assad, Large-scale controlled rounding using Tabu Search with strategic oscillation, Annals of Operations Research 41 (1993) 69–84.
[16] K. MyounJae, K. TaeChoong, Development of automatic class timetabler (ACT) for university, in: Proceedings of Second International Conference on the Practice and Theory of Automated Timetabling, Toronto, 1997, pp. 182–186.
[17] B. Paechter, A. Cumming, M.G. Norman, H. Luchian, Extensions to a memetic timetabling system, in: E. Burke, P. Ross (Eds.), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153, Springer, Berlin, 1996, pp. 251–265.
[18] B. Paechter, R.C. Rankin, A. Cumming, T.C. Fogarty, Timetabling the classes of an entire university with an evolutionary algorithm, Parallel Problem Solving from Nature (PPSN)V, Springer LNCS 1498, 1998.