A Column Generation Scheme for Faculty Timetabling

Andrea Qualizza[1] and Paolo Serafini[1,2]

[1] Department of Mathematics and Computer Science, University of Udine, Italy

[2] CISM, Udine, Italy

Abstract. In this paper we deal with the problem of building a timetable for the courses of a university faculty. We propose an integer linear programming approach based on column generation. Each column is associated to a weekly timetable of a single course. The constraints referring to classroom occupancy and non overlapping in time of courses are in the integer linear programming matrix. The constraints and preferences related to a single course timetable are embedded in the column generation procedure. Generating a column for a course amounts to selecting the currently best time slots in the week. The interaction between the column generation procedure and the branch-and-bound method is also discussed. Some computational results are shown.

1 Introduction

Building a timetable for a university faculty is a common task which has been carried out in many different ways. The reason of this diversity is that perhaps each faculty has its own peculiarities which suggest home made timetabling methods. Although the core of a timetable is an assignment problem, there are always additional constraints which make the problem difficult and not suitable to a general purpose algorithm. Only some of the work done in designing faculty timetables has emerged into the literature. General characteristics of the problem can be found in the papers [1,2].

The methods fall generally into two categories, heuristic methods and integer linear programming methods. The latter cannot be classified as exact methods because very rarely the branch-and-bound tree is fully explored due to the size of most problems. For a review of recent timetabling methods, especially heuristics, see for instance [3].

In the integer linear programming formulation the problem is usually modeled with binary variables expressing the fact that a certain course has been assigned to certain time slots and to certain classrooms, and constraining the variables

accordingly. This has been the approach taken by several authors like (to quote recent papers) [4,5].

However, the integrality relaxation of this model does not provide a strong bound and this has obvious drawbacks in terms of computing time. Besides, the need of introducing more realistic preferences over the set of variables associated to a single course as a whole, instead of just the preference sum for each time slot, cannot be practically done. This makes sometimes very difficult to 'drive' a solution to a better one.

Therefore this approach is sometimes coupled with heuristic techniques which on one hand reduce the complexity of the model and on the other hand allow the user to friendly interact with the procedure to obtain acceptable solutions. See for instance the SAPHIR system described in [6] and based on [7]. Here the authors face also a problem feature which is not required in our model, namely the grouping of students in different course sections. The rigidity of the syllabi in Italian universities, with very few elective courses, results into a straightforward grouping task.

In this paper we propose an original approach which is essentially based on an integer linear programming formulation, but, instead of the usual assignment binary variables, we use binary variables for each weekly course timetable. Due to the exponential number of different course timetables the formulation requires a column generation scheme. The use of column generation procedures is in general recommended because part of the combinatorial structure of the problem is already embedded in the constraint matrix, thus providing stronger bounds (see next section). The power of column generation techniques has been always recognized. See for instance [8,9]. Recently in [10] this approach has been experimented for the timetable problem in high schools, which however has a quite different structure than in academic faculties.

The problem we face consists of courses, classrooms and time slots. Courses must be assigned to both classrooms and time slots by respecting constraints of non simultaneous use of the same classrooms in the same time slot and non overlapping in time of certain groups of lectures. These are the main constraints we have to take care of. Besides, there are preferences on the time slots due to teaching reasons and lecturers' preferences as well. The lecturers may also express their preferences on the whole set of time slots. However, these preferences cannot be fully arbitrary and must be known in advance for them to be taken care of during the column generation phase.

It is convenient to decompose the problem by first considering classroom types instead of single classrooms, with the idea that classrooms of the same type are interchangeable. The main integer linear programing model considers classroom types. Once a timetable is computed, courses are easily assigned to single classrooms.

In our main model a column is associated to a weekly timetable of a single course. The matrix rows take care of the constraints referring to classroom occupancy and non overlapping in time of some courses.

It turns out that generating a column for a course can be computed very quickly because it amounts to picking the best numbers in an array. Once a fractional solution of the integrality relaxation is found we compute an integral solution by resorting to an integrality solver on the generated columns. This may already provide a good solution.

If we want to improve the solution (or if we need to find a solution at all because there was none with the generated columns) we have to start a branch-and-price procedure. However, the column generation scheme conflicts with the branch-and-bound method. There are some subtle issues connected to this point. We are confronted with an NP-hard problem and we show how dynamic programming techniques can be used to design a pseudopolynomial algorithm to generate columns under the additional requirements that some variables are fixed to zero.

The approach proposed in this paper offers two main advantages: the first one derives from the combinatorial properties of the model and, as already remarked, results in a better bound for the branch-and-bound procedure; the second one consists in the possibility of better "controlling" the structure of the weekly schedule, as we shall see later.

The main features of the model are discussed in Section 2, 3 and 4. A small example is provided in Section 5 to better show how columns are generated. The issues related to the branch-and-price procedure are described in Section 6. In Section 7 we briefly describe the computational results obtained by applying the method to the real data of our faculty. Finally we show in Section 8 how courses are assigned to classrooms.

2 The Integer Linear Programming Model

We assume that the classrooms can be partitioned into sets of classrooms of the same type. Classrooms of the same type are interchangeable, i.e. a course can be assigned to any classroom of a certain type. Hence it is simpler to first assign a course to a classroom type and later, via a simple assignment problem, assign the course to a specific classroom. Therefore in this phase we only consider classroom types. We address the problem of assigning courses to classrooms in Section 8. Let K be the set of classroom types and let n_k be the number of classrooms of type $k \in K$.

Let C be the set of courses. Usually a course is ideally suited to a certain classroom type. However, it may be convenient, in case of unavailability, to assign it to a classroom of a different type, if available and feasible. For instance a course with few students should be placed in a small classroom, but it may also be assigned to a large classroom, although this is less preferred in general. Let $K(c)$ be the set of feasible classroom types for course c.

The time-table we consider is weekly. The week is partitioned in a set of time slots. The time slots do not necessarily have the same duration, although equal time slots (e.g. two hours or one hour each) are preferrable in the column generation phase. In this paper we only consider equal time slots. Let H be the set of time slots. Let $d(c)$ be the required number of time slots for course c.

A typical feature of faculty timetabling is that certain courses must not be taught in the same time slot. The most obvious case concerns courses taught by the same person. Besides, there are always courses which should be attended by the same group of students and therefore must be scheduled in different times. Let us define as $C_q \subset C$, $q \in Q$, the sets of non overlapping courses (with Q an abstract index set). Conversely let $Q(c) := \{q \in Q : c \in C_q\}$, i.e. the list of non overlapping groups to which c belongs.

Let $P(c)$ be the set of timetable patterns for the course c. By 'pattern' we mean an assignment of all the required hours per week for the course to definite time slots and definite classroom types. The number of possible patterns for each course is exponential, but we will generate only a subset of patterns. In order to constrain the patterns we need to define the following matrices:

$$a_{(kh)(jc)} = \begin{cases} 1 & \text{if course } c \text{ is assigned the time slot } h \\ & \text{in a classroom of type } k \text{ for the pattern } j \in P(c), \\ 0 & \text{otherwise} \end{cases}$$

$$a'_{(h)(jc)} = \begin{cases} 1 & \text{if course } c \text{ is assigned the time slot } h \\ & \text{for the pattern } j \in P(c), \\ 0 & \text{otherwise} \end{cases}$$

Clearly $a'_{(h)(jc)} = 0$ if and only if $a_{(kh)(jc)} = 0$ for each $k \in K$ and

$$\sum_{h \in H} a'_{(h)(jc)} = \sum_{h \in H} \sum_{k \in K(c)} a_{(kh)(jc)} = d(c) \qquad j \in P(c), \quad c \in C\ .$$

We introduce the following variables

$$x_{jc} = \begin{cases} 1 & \text{if pattern } j \in P(c) \text{ is used for course } c \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The constraints are as follows: we require that

$$\sum_{c \in C} \sum_{j \in P(c)} a_{(kh)(jc)}\, x_{jc} \leq n_k \qquad k \in K, \quad h \in H \tag{2}$$

to avoid simultaneous use of more than n_k classrooms of type k. We require that

$$\sum_{c \in C_q} \sum_{j \in P(c)} a'_{(h)(jc)}\, x_{jc} \leq 1 \qquad h \in H, q \in Q \tag{3}$$

to impose non overlapping of courses in the same group q. We require that

$$\sum_{j \in P(c)} x_{jc} = 1 \qquad c \in C \tag{4}$$

to impose that a course is assigned to exactly one pattern. The number of rows of the problem is given by $|H| \cdot |K| + |H| \cdot |Q| + |C|$, which can be large but not intractable. We consider as objective function the maximization of a preference

$$\sum_{c \in C} \sum_{j \in P(c)} r_{jc}\, x_{jc} \tag{5}$$

where

$$r_{jc} = \sum_{h \in H} \sum_{k \in K(c)} a_{(kh)(jc)}\, s_{khc}$$

and s_{khc} is the preference of using the time slot h and the classroom type k for the course c.

Therefore the integer linear programming problem consists in the maximization of (5) subject to (1), (2), (3) and (4). The approach to solve it is via branch-and-bound with column generation (i.e. branch-and-price). The integrality constraint is relaxed to $x_{jc} \geq 0$ and the patterns (i.e. the columns) are generated until optimality of the integrality relaxation is reached. At this point, unless the solution is integer, the branch-and-price procedure starts.

If we compare this model to the usual model in which binary variables are associated to course-time slots assignments (assignment model) we may observe that each feasible solution of the integrality relaxation of (1), (2), (3) and (4) can be easily turned into a feasible solution of the integrality relaxation of the assignment model, whereas the converse is not true in general (if for instance the assignment variables have different values for the same course in different time slots). Therefore the integrality relaxation of the model of this paper yields a better bound than the relaxation of the assignment model.

3 Column Generation

Let us define the dual variables w_{kh}, v_{hq}, u_c for the constraints (2), (3) and (4) respectively. Then the dual constraints are

$$\sum_{h \in H} \sum_{k \in K} a_{(kh)(jc)}\, w_{kh} + \sum_{h \in H} \sum_{q \in Q(c)} a'_{(h)(jc)}\, v_{hq} + u_c \geq$$

$$\sum_{h \in H} \sum_{k \in K} a_{(kh)(jc)}\, s_{khc} \qquad j \in P(c), \quad c \in C\,,$$

i.e.

$$\sum_{h \in H} \Big(\sum_{k \in K} a_{(kh)(jc)}\, (w_{kh} - s_{khc}) + \sum_{q \in Q(c)} a'_{(h)(jc)}\, v_{hq} \Big) + u_c \geq 0, \quad j \in P(c),\ c \in C\,.$$

So, in order to generate a pattern j for the course c, we have to minimize, with respect to a and a'

$$\sum_{h \in H} \Big(\sum_{k \in K} a_{(kh)(jc)}\, (w_{kh} - s_{khc}) + \sum_{q \in Q(c)} a'_{(h)(jc)}\, v_{hq} \Big)\,. \tag{6}$$

Let us define

$$\hat{w}_{hc} = \min_{k \in K(c)} w_{kh} - s_{khc}$$

$$\hat{v}_{hc} := \sum_{q \in Q(c)} v_{hq}$$

and

$$t_{hc} := \hat{w}_{hc} + \hat{v}_{hc} \ .$$

Then minimizing (6) is equivalent to minimize, for each c,

$$\sum_{h \in H} t_{hc}\, a'_{(h)(jc)} \ . \tag{7}$$

Minimizing (7) can be subject to some constraints or preferences related to the particular course. Let us consider some relevant cases.

The simplest case is the one without constraints, i.e. any set of $d(c)$ time slots is feasible for course c. In the sequel, to ease the notation, we drop the dependence of d on c. In this case minimizing (7) can be done by selecting the d minimum values of t_{hc}, $h \in H$. This computation can be carried out with complexity $O(|H| \log d)$. It is clear that we deal with fixed and generally small values of $|H|$ and d and consequently, it is seems out of place to provide asymptotic bounds. Yet it is useful to realize that the algorithm for a column generation is a fast one.

Quite often we are not allowed to assign more than one time slot per day to a course. In this case the minimization of (7) is carried out by selecting the d minimum values of t_{hc} on different days. This is simply carried out by taking the best values of t_{hc} for each day, let us denote them by $\hat{t}_1$, $\hat{t}_2$, $\hat{t}_3$, $\hat{t}_4$, $\hat{t}_5$, and then selecting the d best values out of them. Here we need just to scan all $|H|$ values and then to scan at most twice the $\hat{t}_i$ values.

Sometimes teachers prefer to teach on consecutive days without any particular preference for the actual days. Then the pattern is generated by considering the best value among $(\hat{t}_1 + \hat{t}_2 + \hat{t}_3)$, $(\hat{t}_2 + \hat{t}_3 + \hat{t}_4)$, $(\hat{t}_3 + \hat{t}_4 + \hat{t}_5)$ (if for instance $d = 3$).

Another preference expressed sometimes by teachers consists in having all classes either all in the morning or all in the afternoon. This can be easily carried out by considering the d best values of t_{hc} of the morning hours and the d best values of t_{hc} of the afternoon hours and by selecting the better solution.

Finally, in case the time slots do not consist of two consecutive hours (as is anyway advisable in general) but of single hours, one typical requirements is that hours for the same course come out in pairs as much as possible. This type of constraint is particularly nasty in the usual formulation. On the contrary it does not make the pattern generation too much harder. Suppose we have to allocate five hours and four of them must be in pairs. Furthermore let us assume that no more than two hours per day can be taught. Then we compute the best values $\hat{t}_1$, $\hat{t}_2$, $\hat{t}_3$, $\hat{t}_4$, $\hat{t}_5$ as before. We also compute all values $t_{hc} + t_{h'c}$ with h and h' consecutive hours and we take the best of these values for each day. Let $\tilde{t}_1$, $\tilde{t}_2$, $\tilde{t}_3$, $\tilde{t}_4$, $\tilde{t}_5$ be these values. Now we have to select three different indices i, j, k such that $\tilde{t}_i + \tilde{t}_j + \hat{t}_k$ is maximum. Although implementing this computation is not straightforward, the computation itself is quick.

Let M_c be the minimum obtained for the course c (no matter the particular rule to generate the pattern for that course). If $M_c + u_c \geq 0$ the optimality

condition is satisfied for the course c whereas if $M_c + u_c < 0$ the pattern obtained by minimizing (7) has to be inserted into the matrix.

4 Initialization

Since finding a feasible time-table is by itself NP-hard, we cannot initialize the matrix with a set of feasible patterns. It is more convenient to introduce artificial variables z_c to the equality constraints (4), which become

$$\sum_{j \in P(c)} x_{jc} + z_c = 1 \qquad c \in C \tag{8}$$

and the original objective function is replaced by

$$\min_{c \in C} z_c \qquad \text{i.e.} \qquad \max_{c \in C} -z_c \ .$$

The only difference in the column generation procedure is that the values s are zero. The initial solution is taken as $z_c = 1$ and $x_{jc} = 0$. Due to the null value of x_{jc}, any pattern can be used to fill up the matrix initially. However, we may even think of starting without any pattern at all and generate all of them. Indeed, no matter which are the initial patterns, the initial values for the dual variables are $w_{hk} = 0$, $v_{hq} = 0$, $u_c = -1$, and therefore the first generated patterns can be any. In order to speed up the computation it is advisable to use as objective function a weighted sum of the original objective and the artificial one.

We recall that we solve the relaxed problem and therefore the initial solution can be fractional. In other words we may find an initial fractional feasible solution even if there is no feasible integer solution.

5 An Example

We limit ourselves to show one single column generation, because everything else is standard. The example is a small instance for illustration purposes. Let us suppose that there are 4 time slots ($H = 4$) and 5 courses c_1, c_2, c_3, c_4, c_5 with $d(c_1) = 2$, $d(c_2) = 2$, $d(c_3) = 1$, $d(c_4) = 1$, $d(c_5) = 2$. The courses are grouped into two non overlapping sets $C_1 = \{c_1, c_2\}$ and $C_2 = \{c_3, c_4, c_5\}$. There are two classroom types ($K = \{k_1, k_2\}$) and two classrooms for each type, i.e. $n(k_1) = n(k_2) = 2$. The relationship between courses and classrooms is defined as $K(c_1) = \{k_1\}$, $K(c_2) = K(c_5) = \{k_2\}$, $K(c_3) = K(c_4) = \{k_1, k_2\}$. We assume that the ideal classroom type for courses c_3 and c_4 is k_1. They can fit type k_2 but in this case their preferences (columns 3 and 4 of table below) are reset to 0. The preferences are

$$s_{hc} = \begin{pmatrix} 0 & 2 & 1 & 2 & 2 \\ 2 & 0 & 1 & 0 & 2 \\ 1 & 1 & 0 & 1 & 1 \\ 2 & 1 & 2 & 1 & 0 \end{pmatrix}$$

Let us suppose that three columns have been generated for each course, so that the matrix A (constraints (2)) is (the columns in boldface correspond to the current solution):

$$
\begin{array}{c}
k_1 \\ \\ \\ \\ k_2 \\ \\ \\ \\
\end{array}
\left\{
\begin{array}{c}
\\ \\ \\ \\ \\ \\ \\ \\
\end{array}
\right.
\left(
\begin{array}{cccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 1 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \mathbf{1} \\
0 & 0 & 0 & 1 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{array}
\right)
$$

The matrix A' (constraints (3)) is:

$$
\begin{array}{c}
C_1 \\ \\ \\ \\ C_2 \\ \\ \\ \\
\end{array}
\left\{
\begin{array}{c}
\\ \\ \\ \\ \\ \\ \\ \\
\end{array}
\right.
\left(
\begin{array}{cccccccccccccc}
1 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & \mathbf{1} & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{array}
\right)
$$

and the assignment matrix (constraints (4)) is:

$$
\left(
\begin{array}{cccccccccccccc}
1 & 1 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & \mathbf{1} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{1} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{1} & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \mathbf{1}
\end{array}
\right)
$$

The objective function coefficients are

$$
(\,1 \quad 4 \quad \mathbf{3} \quad 1 \quad \mathbf{3} \quad 1 \quad 1 \quad \mathbf{2} \quad 1 \quad 0 \quad \mathbf{2} \quad 1 \quad 1 \quad 4 \quad \mathbf{3}\,)
$$

so that the current solution value is 13. The computed dual variables at this stage are

$$
w = 0, \quad v_{h1} = (0\ \ 2\ \ 0\ \ 1), \quad v_{h2} = (1\ \ 0\ \ 0\ \ 0), \quad u_c = (1\ \ 2\ \ 2\ \ 1\ \ 3)\,.
$$

So we compute the following values for the course c_1

$$
\hat{w}_{h1} = \min_{k \in K(c_1)} w_{kh} - s_{khc} = w_{1h} - s_{1hc} = (\,0 \quad -2 \quad -1 \quad -2\,)\ ,
$$

$$
\hat{v}_{h1} := v_{h1} = (\,0 \quad 2 \quad 0 \quad 1\,)
$$

and

$$
t_{h1} := \hat{w}_{h1} + \hat{v}_{h1} = (\,0 \quad 0 \quad -1 \quad -1\,)\ .
$$

We have to allocate two time slots for course c_1 so that the minimum is in selecting time slots 3 and 4 with minimum value $M_1 = -2$. Since $M_1 + u_1 = -1 < 0$, the optimality condition is not satisfied for c_1 and the pattern $(0, 0, 1, 1)$ has to be generated.

For the course c_3 we have

$$
\hat{w}_{h3} = \min_{k \in K(c_3)} w_{kh} - s_{kh3} = \min\{w_{1h} - s_{h3}\,;\, w_{2h}\} = -s_{h3} = (\,-1 \quad -1 \quad 0 \quad -2\,)
$$

$$\hat{v}_{h3} := v_{h2} = (\,1 \quad 0 \quad 0 \quad 0\,)$$

and

$$t_{h3} := \hat{w}_{h3} + \hat{v}_{h3} = (\,0 \quad -1 \quad 0 \quad -2\,)\ .$$

In this case we have to allocate only one time slots and the best way to do it is to allocate the fourth time slot. So $M_3 = -2$. Since $M_3 + u_3 = 0$ the optimality condition is satisfied and there is no need to generate columns for c_3. We omit the similar computations for the courses c_2, c_4 and c_5.

6 Branch-and-Price Strategy

We first solve the relaxed problem by generating columns until optimality is reached. If we end up with a fractional solution, we invoke a ILP routine on the generated columns to get a first incumbent. Then we start a branch-and-bound search. The branching is done by setting the fractional variables to 0 and to 1. These additional constraints however conflict with the column generation scheme. While fixing a variable to 1 poses no problem, there are problems in fixing a variable to 0. Indeed there is no way to prevent generating again columns whose corresponding variables are forced to 0.

We circumvent the problem as follows. Let us suppose that we are solving a subproblem in the branch-and-bound tree for which $(K-1)$ variables have been set to 0. If we compute the first K minima in (7), we are sure that among those minima there is the minimum of (7) with the additional requirement of excluding the $(K-1)$ columns associated to the variables set to 0.

However, computing the first K minima in (7) is not a straightforward problem. Let us consider the case when (7) is minimized without constraints. There is an array of values $T := \{t_1, t_2, \ldots, t_{|H|}\}$ (in general unrestricted in sign). Let $J \subset \{1, 2, \ldots, |H|\}$. The value of the subset J is defined as $\sum_{j \in J} t_j$. We ask if there are at least K distinct subsets of $\{1, 2, \ldots, |H|\}$ with value at most $-u_c$. This is a variant of the K-th LARGEST SUBSET problem, which is known to be NP-hard (see for instance [11]). Our problem is a variant with subsets of equal cardinality and values unrestricted in sign. It is not hard to see that the variant is NP-hard as well. However, it may be solved pseudopolynomially, for instance in the following way. Let $T_j := \{t_1, t_2, \ldots, t_j\}$ and $I_j := \{1, 2, \ldots, j\}$. Define $L(i, j)$ to be a list of the values of the K best subsets of I_j with i elements (the list may have less than K values if K subsets do not exist) and $L^*(i, j)$ to be the list of the corresponding subsets. Then the following dynamic programming recursion holds:

$$L(i, j) = \min\{L(i, j-1)\ ;\ t_j + L(i-1, j-1)\} \tag{9}$$

where $t_j + L(i-1, j-1)$ means that the value t_j is added to each value of the list $L(i-1, j-1)$ and the 'min' operation is actually a merge operation extracting the best K values from the two lists. The same merge operation is carried out on $L^*(i, j-1)$ and $L^*(i-1, j-1) \cup j$. This operation has complexity $O(K)$ on sorted lists and produces a sorted list. The meaning of (9) is that $L^*(i, j)$ is obtained by merging the lists which do not contain j and those which do. By

the optimality principle the latter are optimal if the subsets with one element less up to $j-1$ are optimal.

The recursion is initialized as

$$L(0,j) = \{0\}\,, \quad L^*(0,j) = \{\emptyset\}\,, \quad j := 0, \ldots |H|$$

and is computed for increasing values of i and j (note that $L(i,j)$ and $L^*(i,j)$ are undefined for $i > j$). The complexity is $O(K\,|H|\,d)$. In our case due to the small values of d, H and K this computation is quite fast.

The other cases of minimizing (7) can be taken care of in similar ways. For instance if we consider the case of allowing at most one time slot per day, then we denote the time slots as (jk) (j-th time slot of day k) and define $L(i,k)$ to be a list of the values of the K optimal subsets up to the day k with i elements and $L^*(i,k)$ to be the lists of the same subsets. Then the recursion is as follows:

$$L(i,k) = \min\{L(i,k-1)\ ;\ t_{(1k)} + L(i-1,k-1))\ ;$$
$$t_{(2k)} + L(i-1,k-1)\ ; \ldots\ ; t_{(pk)} + L(i-1,k-1)\}$$

(p is the number of time slots per day) with complexity $O(K\,|H|\,d)$ as before ($L^*(i,j)$ is computed accordingly).

7 Computational Results

We have applied the model to the actual data of one teaching period of our faculty. There are 63 courses, 25 time slots (5 days, 5 time slots per day), 4 classroom types and 25 groups of non overlapping courses. This can be considered a medium size model. The starting LP model has 788 rows and 168 columns. We use CPLEX routines to solve the model.

Solving the LP relaxation requires 731 columns to be generated. The fractional optimum has a value of 1443. At this point the CPLEX MIP routine is called to solve the ILP problem with the currently generated columns. The MIP routine returns the first incumbent with value 1411. The computation time up to this point is 225 seconds, of which 173 seconds are spent on the MIP routine.

As expected the LP relaxation provides a strong upper bound (it is a maximization problem). The gap is $(1443 - 1411)/1443 = 0.02217$, i.e. around 2 %. Since the objective function is an artificial one, in the sense that the preferences are numbers which reflect in an imprecise way the real preferences of teachers and students, we might consider that any solution within a certain gap is acceptable. This has the obvious implications that the branch-and-bound tree does not grow too much. Indeed if we consider acceptable a gap within 3 % we might just take the first incumbent and stop the computation.

If on the contrary we continue the computation up to the very end, we need building a branch-and-bound tree with 556 nodes and depth 53 and generating 534 more columns and we eventually find an integer solution of value 1443. This shows that the gap provided by the relaxed model is actually zero for this instance. This is quite remarkable. It raises however the need of finding a better incumbent to prune more efficiently the branch-and-bound tree and also to look

for better strategies to explore the tree in order to find the optimum as soon as possible. This is subject of future research.

We also point out that it is advisable to implement a derived model in which constraint violations are allowed at the price of high penalty values in the objective function. This does not introduce any new feature in the column generation scheme. If there is no feasible solution (which unfortunately may happen) then the artificial variables associated to the penalty values do not vanish and it is possible to detect which contraints are responsible for the infeasibility, thereby suggesting new requirements on the time table.

Once an initial solution is available this can be modified manually by the user either directly or by fixing part of the timetable and running again the model. We are currently experimenting this successive phase of timetable building.

8 Assigning Courses to Classrooms

Once an integer solution is found to the main problem we have a complete assignment for each time slot of each course to a certain classroom type. We know that the constraint on the number of available classrooms for each type is satisfied by the solution. It is therefore a trivial task to assign a definite classroom for each course if we do not ask for more. We mean that, as long as we consider each time slot independent of the others, we may simply assign in a greedy way the classrooms for each time slot. However, we usually would like to have all lectures of the same course in the same classrooms. Although this is not a compulsive requirement, it is a desired property of a time table. Sometimes, it is more than a soft requirement. If there are many students in a class, it may be very annoying and time consuming having them moving around between lectures.

So it is reasonable to require that certain lectures stay in the same classroom as much as possible. Let the solution of the main model be represented by a set of triples (c, h, k). Each triple states that course c is taught in time slot h in classroom of type k. Clearly the problem we face is decomposed into classroom types, so we may just consider pairs (c, h), taking for granted the classroom type. Let Z be the set pairs (c, h). We partition Z into sets $Z_1, Z_2, \ldots, Z_f$ with the idea that pairs in Z_i should be in the same classroom as much as possible. The partition can be specified manually by taking into account for instance courses with the same groups of students.

Let R be the set of classrooms. Let us define variables

$$x_{chr} := \begin{cases} 1 & \text{if the pair } (c,h) \text{ is assigned to classroom } r \\ 0 & \text{otherwise} \end{cases}$$

and variables

$$y_{jr} := \begin{cases} 1 & \text{if a pair in } Z_j \text{ is assigned to classroom } r \\ 0 & \text{otherwise} \end{cases}$$

Then we may write the following constraints:

$$\sum_r x_{chr} = 1 \qquad (c,h) \in Z$$

stating that each pair must be assigned to a classroom,

$$\sum_{c:(ch)\in Z} x_{chr} \leq 1 \qquad r \in R, \quad h \in H$$

stating that for each time slot and each classrooom there can be at most one course, and

$$x_{chr} \leq y_{jr} \qquad r \in R, \quad (ch) \in Z_j$$

to set y variables consistently with x variables. Then we minimize the number of classroom changes within each set Z_j by minimizing either $\sum_{jr} y_{jr}$ or $\max_{jr} y_{jr}$. Due to the symmetry of the problem and also for practical reasons it is advisable to introduce preferences q_{jr} for the assignments between sets Z_j and classroom r so that we actually minimize $\sum_{jr} q_{jr}\, y_{jr}$.

References

1. de Werra D., "An introduction to timetabling", *European J. of Operational Research,* **19** (1985) 151-162.
2. de Werra D., "The combinatorics of timetabling", *European J. of Operational Research,* **96** (1997) 504-513.
3. Schaerf A., "A survey of automated timetabling", *Artificial Intelligence Review,* **13** (1999) 87-127.
4. Hultberg T.H. and D.M. Cardoso, "The teacher assignment problem:a specila case of the fixed charge transportation problem", *European J. of Operational Research,* **101** (1997) 463-473.
5. Daskalaki S., T. Birbas and E. Housos, "An integer programming formulation for a case study in university timetabling", *European J. of Operational Research,* **153** (2004) 117-135.
6. Ferland J.A. and C. Fleurent, "SAPHIR : A Decision Support System for Course Scheduling", *Interfaces,* **24** (1994) 105-115 .
7. Aubin J. and J.A. Ferland, "A Large Scale Timetabling Problem", *Computers and Operations Research,* **16** (1989) 67-77 .
8. Barnhart C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P. Vance, "Branch-and-price: Column generation for solving huge integer problems", *Operations Research,* **46** (1998) 316-329.
9. Maculan N., M. de Mendonça Passini, J.A. de Moura Brito and I. Loiseau, "Column-generation in integer linear programming", *RAIRO Oper. Res.,* **37** (2003) 67-83.
10. Papoutsis K., C. Valouxis and E. Housos, "A column generation approach for the timetabling problem of Greek high schools", *J. of the Operational Research Society,* **54** (2003) 230-238.
11. Garey, M.R., D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness,* W.H. Freeman and Company, San Francisco, CA, USA (1979).