A column generation approach for examination timetabling

Roel Wijgers

Han Hoogeveen

Department of Information and Computing Sciences,
Utrecht University

Technical Report UU-CS-2007-001

www.cs.uu.nl
ISSN: 0924-3275

A column generation approach for examination timetabling

R. Wijgers J.A. Hoogeveen *
Department of Information and Computing Sciences
Utrecht University
P.O. Box 80089, 3508 TB Utrecht, The Netherlands.
rwijgers@cs.uu.nl, slam@cs.uu.nl

December 22, 2006

Abstract

In examination timetabling we have to assign the exams of a given set of courses to time slots, such that conflicting exams are assigned to different time slots. We study the problem that occurs at our department. The characteristics of this problem are as follows. The exams have to take place in 5 days, and per day there are 2 time slots. Teachers can express their preferences for the time slots. The exams preferably take place in the small rooms, but there are some bigger rooms available as well; since these big rooms are used by other departments as well, using these is not encouraged. Finally, we try to avoid students having two exams on one day as much as possible.

Many authors have presented a number of approaches for solving a myriad of variants of the examination timetabling problem, but remarkably enough column generation seems to have been ignored. We apply column generation to solve several variants of our problem. The pricing problem boils down to a complicated weighted independent set problem, which we solve by a branch-and-bound algorithm. In this way we find very good solutions in a matter of seconds. We further indicate situations in which column generation is applicable.

1980 Mathematics Subject Classification (Revision 1991): 90B35.
Keywords and Phrases: examination timetabling, column generation.

*Supported by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society)

1 Introduction

In examination timetabling, we have to schedule a given set of exams, such that no student has two exams at the same time. We assume that the set of available time slots is given, but there are many applications in which the optimal set of time slots have to be found. Some further (soft) constraints concern the availability of rooms, time slot preferences, and student comfort.

Examination timetabling is an important problem in any educational institution. The quality of a solution is of great importance to a number of parties including lecturers, students, and administrators. Many variants of the problem have been studied since the seventies; we refer to Qu, Burke, McCollum, Merlot, and Lee (2006) for a detailed survey of the automated approaches for examination timetabling presented in the past decade. Strangely enough, column generation has not been used, except for the variant of the problem in which the number of time slots has to be minimized. This problem boils down to a graph coloring problem, for which Mehrotra and Trick (1997) present a column generation approach to find an optimal solution.

In this paper we look at the potential of column generation. We use it to find near-optimal solutions for several variants of the examination timetabling problem that we encountered at our department. We describe the problem in Section 2. In Section 3, we look at the problem of minimizing the number of time slots that we need to allow a feasible solution. We show how it can be solved by applying column generation. In Section 4, we show how we can use the knowledge obtained in Section 3 to deal with the problem of our department, in which the time slots are given and in which we face some additional constraints. In Section 5, we present our results, and in Section 6 we present our conclusions and point out directions for future research.

2 Problem description

We are given a set of exams $E = \{e_1, e_2, ..., e_n\}$, where we know for each exam the students who have to follow it. Hence, we can compute for each pair of exams the overlap. There are two major variants in exam timetabling, depending on whether the available time slots are given. In our case, the time slots are given: the exams take place on five consecutive days, and there are two time slots per day. Nonetheless, we first consider the variant in which we minimize the number of time slots that are needed such that a conflict-free solution is possible, since this gives us insight into the difficulty of the problem. As this minimum turns out to be smaller than the number of available time slots, we require in the remainder that no two conflicting exams are assigned to the same time slot. We then take the rooms into account. Each exam has to take place in one room, but it is possible to have several exams in the same room. For each time slot we have a given set of small rooms available, which we can use as we wish. Furthermore, there are three big rooms available, which we want to use as few times as possible. Hence, we put a term in the objective function to minimize the number of times we use a big room; this term can be refined, for instance, to

model the effect that we prefer not to use all three big rooms at the same time. Next, there are some lecturer's preferences concerning the time slot. Since these can be met without a large additional cost, we consider these as hard constraints. Finally, we want to avoid as much as possible that students have two exams at one day. It follows from the outcome of the problem of minimizing the number of time slots that there is no solution possible in which no student has two exams per day. Hence, we treat it as a soft constraint, and we include the number of students with two exams on one day in the objective function.

Summarizing, our final problem is that we are looking for a conflict-free solution in which all lecturers' requests are met and in which a weigthed combination of the usage of the big rooms and the number of students with two exams per day is minimized.

3 Minimizing the number of time slots

In this section, we assume that we have to find a conflict-free solution that uses as few time slots as possible; there are no constraints concerning the rooms and lecturers' availability. The problem boils down to partitioning the exams in a minimum number of subsets, where each subset contains non-conflicting exams only. From now on, we call such a subset a *time slot schedule*. Suppose that the set S containing all conflict-free time slot schedules is known. We use decision variables $x_j \in \{0,1\}$ $(j \in S)$ to indicate whether time slot schedule s_j is used $(x_j = 1)$ or not $(x_j = 0)$. To describe time slot schedule s_j, we use

$$a_{ij} = \begin{cases} 1 & \text{if exam } e_i \ (i = 1, \ldots, n) \text{ is contained in } s_j, \\ 0 & \text{otherwise} \end{cases}$$

We can then formulate our problem as an ILP as follows

$$\min \sum_{j=1}^{S} x_j \qquad \text{subject to} \tag{1}$$

$$\sum_{j=1}^{S} a_{ij} x_j \geq 1 \qquad i = 1, ..., n \tag{2}$$

$$x_j \in \{0,1\} \quad j = 1, ..., S \tag{3}$$

We use a $\geq$ in constraint (2) instead of an $=$ sign for computational reasons; an exam that occurs twice in a solution can be assigned to one of the time slot schedules arbitrarily. As it is impossible to enumerate all feasible time slot schedules, we use a column generation approach to generate a subset of S that hopefully contains the ones that are needed to find an optimal solution.

3.1 Column generation

We solve the LP-relaxation by column generation and store the columns that are generated. The LP-relaxation is obtained by relaxing constraint (3) to $x_j \geq 0$ $(j \in S)$; we do not need to enforce $x_j \leq 1$, since a solution with $x_j > 1$ is clearly non-optimal.

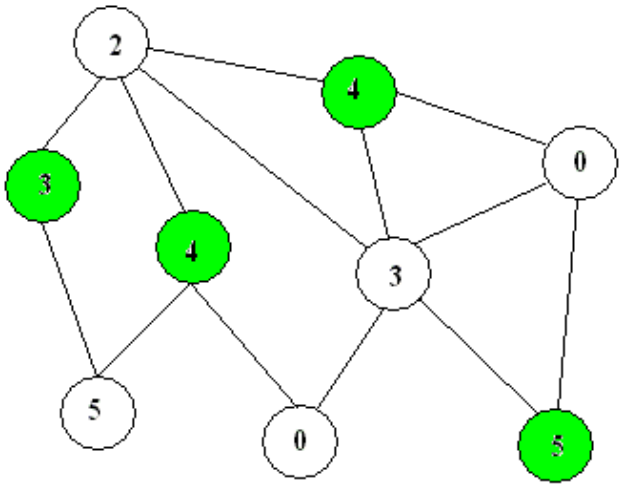

Figure 1: Student constraint graph

As a start, we need to find a valid solution. This is easy: simply generate a time slot schedule for each exam. After we have solved the LP-relaxation for a given set of columns, we find a dual multiplier π_i for the constraint (2) corresponding to exam i. Therefore, the reduced cost for a time slot schedule j is equal to

$$1 - \sum_{i=1}^{n} a_{ij}\pi_i.$$

It is well-known from the theory of linear programming that we have solved the LP-relaxation to optimality if the reduced cost of each time slot schedule is greater than or equal to zero. Hence, we must solve the *pricing problem*, which in this case boils down to computing a feasible time slot schedule with minimum reduced cost. If this reduced cost is non-negative, then we are done, and we add the time slot schedule to the LP-relaxation, otherwise. Hence, we have to maximize $\sum_{i=1}^{n} a_{ij}\pi_i$ over all time slot schedules $j \in S$.

3.2 Solving the pricing problem

Since a time slot schedule is feasible if there is no overlap between any pair of the included exams, we can solve this problem as the problem of finding an independent set of maximum weight. Hereto, we construct the graph $G = (V, E)$, which we call the student constraint graph. We have a node v_i for each exam e_i $(i = 1, \ldots, n)$, and we put an edge between two nodes v_i and v_j whenever exams e_i and e_j have students in common. If we include vertex v_i in the independent set, then we get a reward of π_i $(i = 1, \ldots, n)$. An example is depicted in Figure 1. Here the weights π_i are put inside the vertices. The maximum weighted independent set is marked in gray. The maximal weighted independent set problem is $\mathcal{NP}$-hard in the strong sense, and therefore, we solve it through a branch-and-bound algorithm. We build the branch-and-bound tree by branching on the decision of including an exam in

the current independent set or leaving it out. In each node in the branch-and-bound tree we maintain the following information:

- A partial column containing the exams included so far.
- The current weight of the partial column.
- A list of possible exams to add.

Whenever an exam is included, the information in a node is updated. When the list of possible exams is empty, the weight of the column is compared to the currently best found time slot schedule and kept if it is better. The currently best solution gives a lower bound LB to the optimal solution. In each node we calculate an upper bound UB by summing the current weight and the π_i values of the list of possible exams to add. Whenever $UB \leq LB$ we fathom the node.

To improve the efficiency, we add the exams to the branch-and-bound tree in the order of non-increasing π_i values. Moreover, we use a depth first search in our branch-and-bound tree. As a result, we obtain a good lower bound very fast. Since the first (few) complete column(s) will contain mostly exams with relative large π_i values, the cost of this column(s) probably will not differ much from the optimal column.

After we have solved the LP-relaxation, we solve the ILP-formulation for this set of columns. In this way, we found a solution with 6 time slots, whereas the LP-relaxation needed more than 5 time slots. Therefore, we could conclude that

- finding a conflict-free solution for 10 time slots with the additional constraints was likely to exist;
- no conflict-free solution could exist in which all students have at most one exam per day.

Therefore, from now on we consider the constraint of having a conflict-free solution as a hard constraint. Finally, remark that we do not need to solve the pricing problem to optimality each time, as finding a column with negative reduced cost is sufficient in each iteration. Since the instance that we considered was quite small, solving it to optimality each iteration did not make the problem computationally infeasible.

4 Solving the actual problem

In this section we look at the actual problem faced by our department. From now on, we look for a conflict-free solution in which all exams are assigned to the 10 given time slots. We add the remaining complications one by one and indicate how we should adjust the ILP formulation and the column generation approach that we use to solve the LP-relaxation. In each subsection, we build on on the model of the previous subsection.

4.1 Only a fixed number of rooms is available

In practical applications of examination timetabling there are only a number of rooms available. In our department, there are a number of smaller rooms at our disposal. Moreover, there are three big rooms available, which we should use as little as possible. We model this by using as our objective function that we should minimize the number of times that we use the big rooms. Given a time slot schedule s_j, we compute its cost coefficient c_j as the number of big rooms that are needed in s_j. Hence, the ILP model then becomes that we should find binary values x_j that minimize $\sum_{j \in S} c_j x_j$, such that each exam is part of at least one selected column.

We again solve the LP-relaxation using column generation. The reduced cost of s_j is now equal to

$$c_j - \sum_{i=1}^{n} a_{ij} \pi_i.$$

We solve the pricing problem by determining an independent set that maximizes the opposite of the reduced cost. We have to adjust the column generation procedure slightly to deal with the rooms constraint. Before an exam is included in a partial column, we will have to check whether there is a room available. We will maintain a list of rooms in each node which represents the rooms that are assigned to exams from the partial column. Whenever an exam is included, the cost of the partial column is updated by adding the cost of the assigned room. It is allowed to schedule multiple exams in the same room whenever the capacity of the involved room is sufficient, but we are not allowed to split an exam over two or more rooms. When an exam is scheduled in a room which was used before in the partial column, no cost is added, since this cost was already taken into account earlier.

4.2 Availability of lecturers

In many cases, the lecturer needs to be present at the exam. Hence, the exam can only be planned in the time slots during which the lecturer is available. To deal with this constraint we need to adjust the ILP model. We work now with time slot schedules that are feasible for a specific time slot k; we use S_k to denote the set of feasible time slot schedules for time slot k ($k = 1, \ldots, t$), where t denotes the number of time slots (in our application $t = 10$). A solution to the examination timetabling problem will contain one time slot schedule from each subset S_k ($k = 1, \ldots, t$). For each time slot, we add a constraint that at most one time slot schedule can be selected. The ILP model then becomes

$$\min \sum_{k=1}^{t} \sum_{j \in S_k} c_j x_j \qquad \text{subject to} \tag{4}$$

$$\sum_{k=1}^{t} \sum_{j \in S_k} a_{ij} x_j \geq 1 \qquad i = 1, ..., n \tag{5}$$

$$\sum_{j \in S_k} x_j \leq 1 \qquad k = 1, ..., t \tag{6}$$

$$x_j \in \{0,1\} \quad j \in \cup_{k=1}^{t} S_k \tag{7}$$

The column generation approach changes slightly. Instead of generating one column after optimizing the current LP-model, we generate one column for each time slot. A column will be added to the LP, whenever its reduced cost is negative, where the reduced cost of time slot schedule s_j for time slot k is defined as

$$c_j - \sum_{i=1}^{n} a_{ij}\pi_i - \lambda_k,$$

where λ_k is the dual multiplier for the constraint (6) corresponding to time slot k. We can apply the branch-and-bound algorithm of the previous subsection, where we only include exams that can be taken in time slot k.

The decision to consider the lecturers' preferences as hard constraints simplifies the problem tremendously, as it reduces the size of the pricing problem per time slot. If these preferences were soft constraints, then we can use the same ILP-formulation. When we solve the pricing problem for time slot k, then we need all exams in the student constraint graph, and we include a penalty for adding an exam that is not preferred in this time slot.

Working with designated time slot schedules gives us the opportunity to refine the cost structure for the usage of the big rooms as well. We can easily incorporate issues like

- limited availability of big rooms in specific time slots
- non-linear cost functions for using the big rooms

in our model to avoid using too many big rooms at once.

4.3 Avoiding two exams per day

To improve the student examination results, it helps if students only have to take one exam per day. Since there is no conflict-free solution that uses only 5 time slots for the situation that we relax all other constraints, there does not exist a conflict-free solution in which no student has two exams per day. Therefore, we consider it a soft constraint and include the number of violations in the objective function.

Again, we have to adapt the ILP model. We cannot work with time slot schedules anymore, because of the relation between the selected time slot schedules per day. Therefore, we now work with examination day schedules. We denote the number of examination days by d, which is 5 in our case. Note that we have only two time slots per day, but our approach can be easily extended to deal with more than two time slots per day. The ILP model then becomes

$$\min \sum_{k=1}^{d} \sum_{j \in D_k} c_j x_j \qquad \text{subject to} \tag{8}$$

$$\sum_{k=1}^{d} \sum_{j \in D_k} a_{ij} x_j \geq 1 \qquad i = 1, ..., n \tag{9}$$

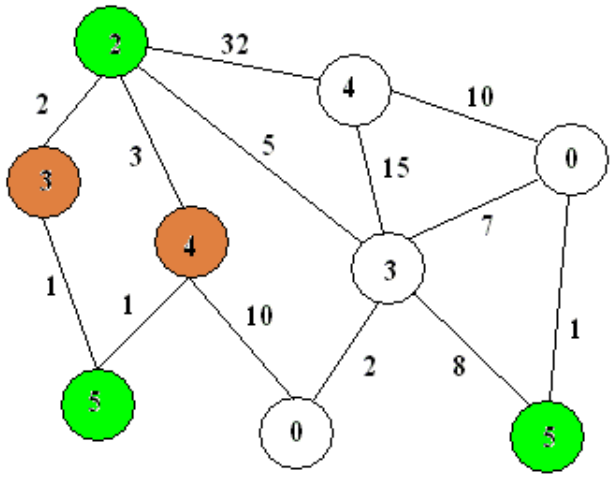

Figure 2: Two optimal independent sets

$$\sum_{j \in D_k} x_j \leq 1 \qquad k = 1, ..., d \tag{10}$$

$$x_j \in \{0, 1\} \quad j \in \cup_{k=1}^{d} D_k \tag{11}$$

Here d_j denotes examination day schedule j, and its cost is denoted by c_j; D_k denotes the subset of feasible examination day schedules for day k; and a_{ij} indicates whether exam i is included in examination day schedule j. The cost coefficient c_j consists of a term measuring the number of students with two exams per day in this examination day schedule and of a term measuring the number of big rooms used.

We again select good columns by solving the LP-relaxation through column generation. To solve the pricing problem, we no longer have to find an optimal independent set, but we have to find two independent sets that together minimize the reduced cost, which is equal to

$$c_j - \sum_{i=1}^{n} a_{ij}\pi_i - \lambda_k.$$

We maximize the opposite, where we ignore the constant term λ_k. We solve this using the student constraint graph. The edges in the student constraint graph get a weight denoting the number of students that have to take both exams. We use a branch and bound algorithm to solve the pricing problem for each day to optimality. We have to adjust the branching rule slightly, since we now have three instead of two options for an exam: An exam is scheduled in the first time slot, in the second time slot, or it does not get scheduled at all. We administrate in each node the current independent sets, the lists of the available exams per time slot, and the lists of the used rooms. An example student constraint graph with two maximal independent sets with as little overlap as possible can be found in Figure 2. The dual multipliers are depicted in the vertices.

5 Results

All tests have been performed on a Pentium(4) 2.8 (Ghz) computer with 512 MB RAM. The used programming language is Java, where CPLEX 9.120 was used as a subroutine to solve the various linear programming problems.
All of the explained techniques have been tested on a real life problem instance from the department of Computing Science, Utrecht University. This real-life problem instance only contains 31 exams, so the search space of possible examination rosters is not very huge. The examination rosters that need to be generated in this department consider a five day examination week with two time slots per day. It turned out that there exists a conflict-free solution that needs only 6 time slots. Then we took the rooms into account. The examination scheduler of the department indicated that the huge examination rooms were a scarce resource and should be used as few as possible. Other, smaller rooms were considered to be available anytime. This real life situation was reflected in the room cost by giving a cost of one to the huge examination rooms and a cost of 0 to the smaller rooms. The optimal solution that was found contained 11 uses of the huge examination rooms, a little more than one per time slot. Including the lecturers' preferences did not yield any difficulty. Finally, we included the objective of minimizing the number of students with more than one exam per day. Here we considered the usage of a big room to be five or ten times more important than a student with two exams per day. In both cases the same solution was found, in which 5 students had two exams on one day and in which 11 times one of the big rooms was used. All of the above problems were solved within a few seconds computation time.

The examination scheduler of the department of Computing Science of Utrecht University is now using our algorithm to full satisfaction. Before this it was already hard to construct a feasible schedule without taking additional criteria into account. Now, he can generate feasible examination rosters within a few seconds that are optimal with respect to the usage of the scarce rooms and the number of students with more than one exam per day.

5.1 Performance on a benchmark dataset

We have applied our algorithm on some benchmark problems for examination timetabling that have appeared in the literature. Unfortunately, there are no data available stating the room availability and stating the amount of overlap. Therefore, we could only apply our approach on the problem of minimizing the number of time slots. We have tested it on the well-known and well-studied University of Toronto benchmark dataset for examination timetabling. The results have appeared in the table below. The density resembles the average number of edges for one exam, where a density of 1 means that all exams are conflicting with each other.

Instance	Number of exams	Density	Best known solution	Our solution	Runtime in mins
UTE92	184	0.08	10	12	267
EAR83	190	0.27	22	25	371
LSE91	381	0.06	17	20	442
KFU93	461	0.06	19	23	460

At first (and second) sight, our column generation approach does not work very well. The major problem was the size of the problem instance. Therefore the pricing problem could not be solved to optimality each time. Since solving it to optimality each time is not necessary until the column generation converges to the optimum, we just stopped the branch-and-bound as soon as we had found a column that was 'improving enough'. A disadvantage is that this does not give a lower bound, since the LP-relaxation is not solved to optimality. To counter this, we could compute a lower bound as

$$\frac{\sum_{i=1}^{n} \pi_i}{\hat{c}},$$

where $\hat{c}$ is equal to 1 minus the minimum reduced cost, which is equal to the outcome of our pricing problem (see for example Van den Akker, Hoogeveen, and Van Kempen (2006)). Once-in-a-while, we need to solve the pricing problem to optimality then.

Another problem is that presumably we need more columns when solving the ILP then just the ones that were generated in the solution process of the LP-relaxation.

6 Further research

We have shown that column generation is able to solve our practical problem easily and that the additional constraints pose no problem at all. Solving the basic problems from the University of Toronto benchmark dataset was still a bridge-too-far, but we want to stress that our application was merely intended to solve the problem of our university, and many features can be added. In this way, the smaller instances of the dataset should be solvable.

A well known criterion in examination timetabling is maximizing the so called paper spread. This criterion means you want the exams of individual students to be as far away from each other as possible. This criterion can not be taken into account in a linear programming approach. The main problem is the fact that the cost of a column is influenced by the other columns in a solution and therefore is not a constant. Column generation might be useful here in a two-phase approach, in which in the first phase good time slot or day schedules are generated, which are then assigned to time slots or days to maximize the paper spread.

References

[1] J.M. van den Akker, J.A. Hoogeveen, and J.W. van Kempen (2006). Parallel machine scheduling through column generation: minimax objective functions (ex-

tended abstract). Y. Azar and T. Erlebach (Eds.) *ESA 2006*. LNCS 4168, Springer, 648–659.

[2] A. Mehrotra and M.A. Trick (1997). A column generation approach for graph coloring. *INFORMS Journal on Computing 8*, 344–354.

[3] R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot, and S.Y. Lee (2006). *A survey of search methodologies and automated approaches for examination timetabling*, Technical Report NOTT-CSTR-2006-04, School of Computer Science & IT, University of Nottingham.