22nd International Congress on Modelling and Simulation, Hobart, Tasmania, Australia, 3 to 8 December 2017
mssanz.org.au/modsim2017

Using column generation to solve an aircrew training timetabling problem

D. Kirszenblat [a], **B. Hill,** [a] **V. Mak-Hau** [b], **B. Moran** [c], **V. Nguyen** [d], **A. Novak** [d]

[a] *School of Mathematics and Statistics, University of Melbourne, Parkville, VIC 3010, Australia*
[b] *School of Information Technology, Deakin University, Waurn Ponds, VIC 3216, Australia*
[c] *Royal Melbourne Institute of Technology, 124 La Trobe St, Melbourne VIC 3000, Australia*
[d] *Defence Science and Technology Group, Department of Defence, 506 Lorimer St, Fishermans Bend, VIC 3207, Australia*
Email: d.kirszenblat@student.unimelb.edu.au

Abstract: The Training Authority Aviation (TA-Avn) is an organisation within the Royal Australian Navy (RAN) responsible for managing aviation-specific training for all RAN personnel, who are to be employed in an aviation-related job category. In a temporal sense, the bulk of aircrew training consists of a sequence of major, structured courses and a number of mandatory short courses for which the prerequisite requirements are less strict. Both short and long courses are run repeatedly throughout a year with a fixed number of repetitions and are subject to high and extremely variable course pass rates. It is important to have an ability to quickly and easily regenerate a new timetable at short notice, potentially on a weekly basis depending on whether students have to repeat failed short courses.

In previous work we explored a number of approaches including a stochastic approach to optimisation. In this paper, we adopt a different methodology, using more conventional integer linear programming techniques, specifically, column generation. The problem of designing feasible schedules is formulated as a network flow problem that encompasses covering and prerequisite constraints. Then column generation is applied in order to improve the tractability of this large scale integer linear program. Here, the original problem is decomposed into a master and subproblem. The master problem is initialised with a set of dummy schedules to which we allocate the aircrew student population, whilst respecting class capacity limitations. The master problem then requests solutions from the subproblem that offer some promise of minimising the overall time spent in training. This process iterates between the master and subproblems until the solution of the master problem cannot be further improved and we have thus reached an optimal allocation of students to feasible schedules. Experimental results are compared with those of an ILP approach that assigns feasible schedules to labelled students.

Keywords: *Optimal timetabling, integer linear programming, network flow, column generation*

1 INTRODUCTION

This paper is concerned with the problem of optimally assigning trainee helicopter pilots in the Royal Australian Navy to a least-cost set of training schedules. The objective is to minimise the total time to graduate all students, as the total time to graduate is a proxy for both financial and morale costs. The key features of this problem are as follows. Students are required to pass a number of courses in order to graduate. Each course has multiple instances held at different times throughout the year and referred to as course sessions. Students must follow feasible schedules of course sessions, where a schedule is deemed feasible if it covers each course exactly once and satisfies certain prerequisite requirements. Some courses are run by external organisations. As such, TA-Avn does not have control over the times and capacities of all courses sessions. In (Bayliss et al., 2016), a stochastic tabu search approach is used to automate the timetabling process. However, in this paper, we adopt the technique of column generation as a first step toward the goal of obtaining an exact solution.

The approach adopted in this paper is to decompose the problem into two parts. The *subproblem* of designing a feasible schedule that covers all required courses and satisfies prerequisite constraints is formulated as a network flow problem. On the other hand, the *master problem* of optimally assigning students to feasible schedules whilst respecting course session capacity constraints is solved using column generation as a heuristic method. The idea is to iterate back and forth between the two problems. The solution of the master problem guides the search for an optimal solution of the subproblem and vice versa. The remainder of this paper is structured as follows. Section 2, which concerns the subproblem, provides a detailed explanation of the constraints to be satisfied by a feasible schedule. Section 3 examines the master problem. In Section 4 we compare the results of column generation to those obtained by an ILP. We conclude by offering suggestions for improvements to the current approach.

2 SUBPROBLEM FORMULATION

This section presents a set of constraints to be satisfied by a feasible schedule. The problem of designing a feasible schedule that minimises the value of some linear objective function is referred to as the column generation subproblem.

2.1 Representing the course prerequisite structure

The prerequisite structure can be represented by a digraph $G(V, A)$. Refer to Figure 1 for an illustration of the digraph associated with the prerequisites listed in Table 1. Let n_C denote the number of courses excluding the entry course. For $i = 0, \ldots, n_C$, the digraph $G(V, A)$ includes a vertex v_i corresponding to the ith course. For each pair (c_i, c_j) consisting of a course c_j and its prerequisite c_i (if it exists), the digraph $G(V, A)$ includes an arc a_{ij} pointing from vertex v_i to vertex v_j. Observe that the red arcs in Figure 1 may be omitted, because a flow along such arcs would bypass certain other vertices corresponding to prerequisite courses. In the example, course 2 is a prerequisite for courses 3 and 4. As course 3 is also a prerequisite for course 4, one cannot take course 4 immediately after taking course 2. Hence, a dashed arc is drawn between vertices v_2 and v_4. In general, an arc a_{ij} may be removed from the digraph $G(V, A)$ if its removal leaves a directed path from vertex v_i to vertex v_j. Introduce a sink v_F. If the ith course is not a prerequisite for any other course, then the vertex v_i is connected by an arc a_{iF} to the sink v_F. Finally, note that there is a partial order $<$ on the vertex set V such that $v_i < v_j$ if the vertex v_i precedes the vertex v_j in a directed path from the source v_0, corresponding to the entry course, to the sink v_F. If neither $v_i < v_j$ nor $v_j < v_i$, as is the case with the vertices v_4 and v_6 in Figure 1, then introduce the directed arcs a_{ij} and a_{ji}. Such extra arcs are illustrated in orange in Figure 1.

Table 1. Courses and prerequisites

Course	Prerequisite
1	0
2	1
3	2
4	2, 3
5	2, 4
6	2, 3

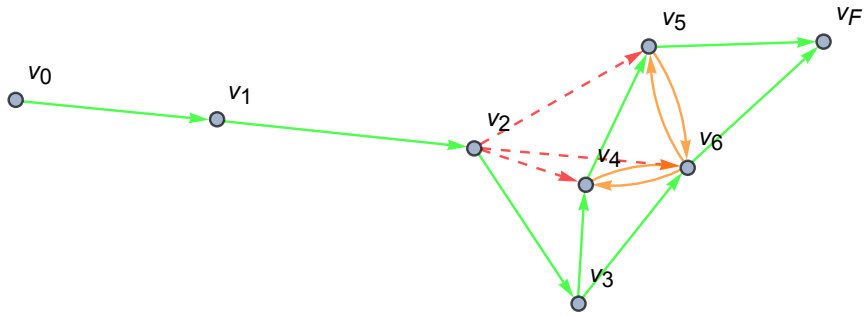

Figure 1. The digraph $G(V, A)$ represents the prerequisite structure

2.2 Exact cover constraints

Let c_{ij} denote the jth session of the ith course. (Refer to Table 2 for an example of course session input data.) Let z_{ij} denote a binary decision variable that is equal to 1 if course session c_{ij} is included in the schedule and 0 otherwise. The covering constraints ensure that each course is covered by exactly one session in a schedule.

$$\sum_{j} z_{ij} = 1 \qquad i = 0, 1, 2, \ldots, n_C \qquad \text{Exact cover} \tag{1}$$

Table 2. An example of course session input data

c_{ij}	Start Day	End Day	Capacity
c_{01}	53	53	9
c_{02}	115	115	15
c_{11}	77	217	30
c_{12}	140	279	18
c_{21}	84	226	27
c_{22}	182	325	26

2.3 Flow conservation constraints

Let y_{ij} denote a binary decision variable that is equal to 1 if the schedule flows directly from course i to course j and 0 otherwise. A feasible schedule must satisfy the following flow conservation constraints:

$$\sum_{a_{ij} \in A} y_{ij} = 1 \qquad i = 0, 1, \ldots, n_C \qquad \text{Unitary outflow} \tag{2}$$

$$\sum_{a_{ij} \in A} y_{ij} = 1 \qquad i = 1, 2, \ldots, F \qquad \text{Unitary inflow} \tag{3}$$

The first set of constraints says that the schedule flows out from every course. The second set of constraints says that the schedule flows into every course except the entry course and that the schedule terminates in exactly one course.

2.4 Prerequisite constraints

Let $\tau(i)$ denote the index set corresponding to the sessions of course i. Let S_{ij} denote the start day of course session c_{ij}. Similarly, let F_{ij} denote the end day of course session c_{ij} . Denote by $\mathcal{P}_i$ the set of prerequisites

for course i. We impose the following set of constraints:

$$\sum_{j \in \tau(i)} F_{ij} \times z_{ij} + 1 \leq \sum_{l \in \tau(k)} S_{kl} \times z_{kl} \qquad \text{for all } k \text{ and } i \in \mathcal{P}_k \tag{4}$$

This constraint ensures that the session of course i in the schedule must finish at least a day prior to the commencement of the session of course k in the schedule.

2.5 Time constraints

We require additional constraints to reflect the temporal order of course sessions. These constraints are similar in structure to the prerequisite constraints. Let ES_i denote the earliest start day of any session of course i. On the other hand, let LF_i denote the latest end day of any session of course i. Define the constant M_{ij} as the difference $LF_i - ES_j$. We define a *terminal arc* as an arc that is connected to the sink, and denote by $\tilde{A}$ the set of nonterminal arcs. For each nonterminal arc a_{ij} in $\tilde{A}$ such that M_{ij} is nonnegative, we impose the following constraint on the start and end times of its associated course sessions:

$$\sum_{j \in \tau(i)} F_{ij} \times z_{ij} + 1 - M_{ik}(1 - y_{ik}) \leq \sum_{l \in \tau(k)} S_{kl} \times z_{kl} \qquad \text{for all } a_{ik} \in \tilde{A} \text{ and } M_{ik} \geq 0 \tag{5}$$

To see how the time constraints work, suppose that the binary decision variable y_{ik} is turned on. That is, the schedule flows directly from course i to course k. Then the associated constraint ensures that session of course i included in the schedule ends at least one day prior to the commencement of the session of course k included in the schedule. If y_{ik} is turned off, then the associated constraint is inactive. Note that these constraints imply that a schedule corresponds to a Hamiltonian path through the vertices of the digraph $G(V, A)$. That is, subtours of the vertices of the digraph $G(V, A)$ cannot arise, as they do not permit a temporal ordering of course sessions.

2.6 Computing the makespan of a schedule

The *makespan* or duration of a schedule is a common measure of cost in scheduling problems. In order to compute the makespan of the schedule, we introduce a linear cost function. Let LS_0 denote the latest start time of any entry course session. (Note that there is only one entry course but possibly several entry course sessions.) Let t_0 denote the difference of the start time of the schedule and the latest start time LS_0 of any entry course session. Let t_F denote the end time of the schedule. Define the vector $\mathbf{v}$ to be $\mathbf{v} = (1, \mathbf{y}, \mathbf{z}, t_0, t_F)$. The first entry of the vector $\mathbf{v}$ is equal to 1 and the remaining entries are the decision variables of the subproblem. Define the vector $\mathbf{q}$ to be $\mathbf{q} = (LS_0, 0, 0, \ldots, -1, 1)$. That is,

$$\mathbf{q} \cdot \mathbf{v} = LS_0 - t_0 + t_F$$

Let τ_0 denote the index set corresponding to the entry course sessions. Let τ_F denote the index set corresponding to the terminal course sessions, i.e., the course sessions in which a schedule can terminate. We compute the makespan of the schedule by solving the problem to

$$\min \mathbf{q} \cdot \mathbf{v} \tag{6}$$

$$\text{subject to } t_0 \leq (S_{0j} - LS_0) z_{0j} \qquad \text{for all } c_{0j} \in \tau_0 \tag{7}$$

$$t_F \geq F_{ij} z_{ij} \qquad \text{for all } c_{ij} \in \tau_F \tag{8}$$

The constraints are defined so that the variable t_0 picks out the difference of the start time of the entry course session appearing in the schedule and the constant LS_0, whereas the variable t_F picks out the latest finish time of any terminal course session appearing in the schedule.

3 MASTER PROBLEM FORMULATION

The objective of the master problem is to assign a fixed number, say N, of students to a set of feasible schedules so that the total time spent in training is minimised. The key idea of column generation stems from the observation that the simplex method does not require all columns of the constraint matrix in order to find an optimal solution. Rather, columns that offer some promise of improving the value of the objective function can be generated as needed. In our master problem, the columns of the constraint matrix correspond to feasible

schedules. Column generation can therefore be used to identify feasible schedules that are likely to aid in finding an optimal solution as opposed to exhaustively enumerating all feasible schedules before optimising.

Suppose that there is a total of J feasible schedules satisfying the constraints of the subproblem. Once again, we emphasise that the feasible schedules do not need to be known in advance. For $j = 1, 2, \ldots, J$, let f_j denote the makespan of the jth feasible schedule. Let $\mathbf{f} = (f_1, f_2, \ldots, f_J)$ denote the cost vector. Similarly, for $j = 1, 2, \ldots, J$, let x_j denote the number of students assigned to the jth feasible schedule. Let $\mathbf{x} = (x_1, x_2, \ldots, x_J)$ denote the vector of integer decision variables. Let $\mathbf{z}_j$ denote the column of binary decision variables from the subproblem indicating which of the course sessions are included in the jth feasible schedule. The entries of the column $\mathbf{z}_j$ are treated as constants in the master problem. Similarly, let $\mathbf{b}$ denote the vector whose entries are the capacities of the respective course sessions. Now, the master problem is to

$$\min \quad \mathbf{f} \cdot \mathbf{x} \tag{9}$$

$$\text{subject to} \sum_{j=1}^{J} x_j = N \qquad \text{Student assignments} \tag{10}$$

$$\sum_{j=1}^{k} \mathbf{z}_j x_j \leq \mathbf{b} \qquad \text{Course session capacities} \tag{11}$$

$$\mathbf{x} \geq 0 \tag{12}$$

We choose to initialise the master problem using a single dummy column, interpreted as a dummy schedule to which we assign N students without violating the session capacity constraints. The dummy column need not be a feasible solution of the subproblem, provided we assign its master variable a sufficiently large cost. Assuming the master problem is feasible, when the algorithm terminates the dummy column will have either been driven out of the basis and replaced by some feasible solutions of the subproblem or its corresponding master variable will have been assigned the value of 0. We set the cost of the dummy schedule equal to M, where

$$M = \max\{LF_j : j = 1, 2, \ldots, n_c\} - \min\{ES_j : j = 1, 2, \ldots, n_c\} + 1$$

That is, M is the longest possible makespan of a schedule, and so the dummy schedule is at least as costly as any other schedule. Let $\mathbf{p}$ denote the vector of dual costs. (See, for example, (Bertsimas et al., 1997) for details of how to obtain the dual costs.) In order to identify a feasible schedule offering some promise of reducing the total time spent in training, we introduce the objective function $(\mathbf{q} - \mathbf{p})$ to the subproblem. That is, the subproblem is to

$$\min \quad (\mathbf{q} - \mathbf{p}) \cdot \mathbf{v} \tag{13}$$

subject to the exact cover, flow conservation, prerequisite and time constraints. The iterative method is carried out as follows. We turn our attention to the subproblem with the updated objective function. That is, we seek to minimise $(\mathbf{q} - \mathbf{p})$ over all schedules. If we find a schedule $\mathbf{v}_k$ for which $(\mathbf{q} - \mathbf{p}) \cdot \mathbf{v}_k$ is negative, then the corresponding column is said to have *negative reduced cost* and enters the basis. At the first iteration, at least one such schedule must exist if the subproblem is feasible. We then re-solve the master problem to find an optimal assignment of students to the updated set of schedules. The algorithm terminates when we are unable to find a column with negative reduced cost. At this stage we solve the master problem as an integer program in order to obtain an integral allocation of students to schedules. The reader is referred to [2] for a more detailed explanation of column generation.

Note that column generation is used as a heuristic method for solving this particular problem. That is, as many columns as needed are generated in order to solve the LP relaxation of the master problem, whereby fractional numbers of students are assigned to schedules. Then an integer solution is obtained by solving a restricted IP using only those schedules that have been obtained via column generation. As will be seen in the next section, column generation produces optimal solutions for the datasets that have been tested. However, in principle the optimal integer solution to the original problem and the optimal solution to the LP relaxation may not share the same set of variables, in which case the restricted master IP may produce a suboptimal answer or be infeasible. In the case where only the entry course sessions are at capacity, column generation will produce an optimal solution. To see this, note that for each entry course session, column generation will fill a shortest schedule with the required number of students. On the other hand, we have not obtained performance bounds for the

case where course sessions other than the entry course sessions are at capacity. However, an exact solution could be obtained by incorporating the column generation formulation presented here into a branch and price formulation.

4 EXPERIMENTAL RESULTS

In this section, we compare the results of column generation with those of an ILP according to which N labeled students are each assigned exactly one schedule. Both models were implemented using CPLEX and runs were performed on the same computer (Intel Core i5 processor with 8GB RAM) for comparison of execution times. The ILP model was implemented in Java, whereas column generation was implemented in MATLAB. As can be seen from the data in Table 3, column generation obtains the same results as the ILP, albeit more quickly for larger datasets. The respective values of the objective function are identical for both models and are not included in the table. Note that the last four rows of Table 3 pertain to data sets for which only the start and end times of intermediate course sessions were altered; the start and end times of entry and terminal course sessions were fixed. The difference in computational efficiency may be in part attributed to the high degree of symmetry associated with the ILP. There are two ways in which symmetry slows down the performance of the ILP. First, many optimal combinations of schedules may be identical after permuting the labels associated with the students. Second, an optimal combination of schedules may involve assigning the same schedule to different students. And as the number of students increases, so does the number of times that the same schedule has to be computed. This is very wasteful given that the problem is NP hard and the computation time is expected to grow exponentially with the size of the problem. On the other hand, column generation works in reverse by assigning a number of unlabelled students to schedules and is therefore insensitive to changes in the number of students.

Table 3. Experimental results

# Courses	# Sessions	ILP time (s)	CG time (s)
5	62	2.92	1.16
5	86	3.62	1.08
6	77	0.37	0.77
20	166	0.27	1.46
20	167	0.26	1.72
20	166	0.28	1.47
20	168	0.27	1.37
20	162	1.55	1.93
20	168	1.95	1.88
20	168	0.26	0.95
20	168	1.83	1.80
20	168	1.97	1.62

5 FUTURE DIRECTIONS

In future work we will modify the network flow formulation to account for pass rates associated with the courses and will implement branch and price in order to ensure that an exact solution is obtained. Another idea worth exploring is the possibility of warm starting the ILP solver by providing it with feasible solutions obtained by some other fast method.

6 CONCLUSIONS

This paper presents a column generation approach to optimally assigning students to training schedules. A comparison in terms of speed is made between the column generation approach and an ILP approach involving the assignment of feasible schedules to labelled students. Experimental evidence and symmetry arguments suggest that the column generation approach performs significantly faster as the size of the problem increases.

REFERENCES

Bayliss, C., A. Novak, A. Nguyen, A. Moran, A. Caelli, S. Harrison, and S. Tracey (2016). Optimising aircrew training schedules using tabu search. *Australian Simulation Congress (SimTecT).*

Bertsimas, D., and J. N. Tsitsiklis (1997). Introduction to Linear Optimization. Athena Scientific, Massachusetts