ABSTRACT

Algorithms for timetabling solutions typically involve sequential allocation of students to courses and resources as the algorithm unfolds. Most current solutions, to this end, commonly use some form of stochastic optimization. In this paper, we propose a novel paradigm for optimal timetabling that comprises two distinct phases. First, we enumerate all feasible course schedules, along with their costs, using a modified implementation of Knuth's Dancing Links technique for the exact cover problem. To our knowledge, the only prior use of this implementation has been to solve games such as Sudoku and N-Queens. Once the list of all solutions that satisfy prerequisite and time-clash constraints is generated, the second phase applies a standard deterministic optimization to allocate students to these feasible schedules. This method has been applied to a real complex timetabling problem in the Royal Australian Navy helicopter aircrew training program. The results are compared, in terms of computational time, to an exhaustive best practice backtracking algorithm for generating a complete set of feasible schedules, as well as to a pure integer linear programming solution for generating schedules and allocating students to schedules.

INTRODUCTION

Solutions to timetabling problems essentially involve allocation of human and material resources to available time slots to satisfy constraints for a given domain. A particularly challenging example of this occurs in a training continuum when the supply of students is fluctuating, pass rates are highly variable, and class size is a hard constraint imposed by resource requirements. This occurs, for example, in the training of surgeons for advanced robotic surgery and, in our case, the training of helicopter pilots. In these situations, constraints are clearly defined and schedules allocated to all students at the start of the program with the knowledge that only a small fraction of them will graduate. Another key feature of this type of problem is that the number of students is relatively small. In the case of the Royal Australian Navy (RAN), numbers are around 50, though the per-student costs are very high. Besides expensive infrastructure (principally helicopters and simulators) and instructors, a significant part of the cost is due to delays—when students are simply waiting between courses. This is a cost, not only in financial terms, but in morale.

Optimal scheduling or timetabling in this context involves the satisfaction of hard constraints and minimization of costs of "soft" constraints in an efficient way. Metaheuristic methods, being a combination of constraint satisfaction along with, for example, stochastic optimization, or integer programming that, to retain computability, relax to linear programming with the potential loss of optimality, are currently widely used.

Optimal timetabling is generally described in terms of combinatorial optimization where events are allocated to time periods and places, subject to constraints (Wren, 1996; Burke et al., 1997; de Werra, 1997) and a given cost function. Such problems have been shown to be NP-hard, with computation time increasing exponentially with the number of events to be scheduled and the number of constraints imposed (Karp, 1972; Even et al., 1975; Johnson and Pilcher, 1988). To combat this computational complexity, various stochastic search algorithms have been proposed. Tabu search (Glover and Laguna, 1997; Battiti and Tecchiolli, 1994), has been a popular choice in various scheduling problems; for example, job shop scheduling and timetabling (Dell'Amico and Trubian, 1993; Hertz and Widmer, 1996; Nowicki and Smutnicki, 1996). A defining feature of Tabu search is the use of "hard" and "soft" constraints to provide, on the one hand, the delineation of the feasible space of candidate solutions and, on the other, an objective function to rank the "better" timetables from an optimization viewpoint. Ultimately, the entire search is concerned with minimization of the objective function, and, therefore, the satisfaction of the constraints.

In the case of RAN aircrew training, given that adequately resourcing of helicopter platforms is synonymous with achieving capability, even small fluctuations in supply are extremely costly. Achievement of an optimal timetabling solution in this situation becomes even more important. We propose a two-phase approach for optimal timetabling. First,

Dancing Links for Optimal Timetabling

Vivian Nguyen

Department of Defence, Australia
Vivian.Nguyen@dst.defence.gov.au

Bill Moran

University of Melbourne

Ana Novak

Department of Defence, Australia

Vicky Mak-Hau and Terry Caelli

Deakin University

Brendan Hill and David Kirszenblat

University of Melbourne

APPLICATION AREAS: Readiness and Training (Force Readiness), Advances in Military OR (Methods)

OR METHODOLOGIES: Deterministic Operations Research (Linear Programming), Other (Dancing Links)

we enumerate all feasible course schedules along with their costs. Then integer linear programming (ILP) is applied to optimally allocate students to these feasible schedules.

The key features of the aircrew timetabling problem that determine our approach are:

- It has a long and complex list of courses with a complicated prerequisite structure.
- There are a significant number of course repetitions with time clashes.
- Each student must complete all courses in the syllabus.
- Each student does only one course at a time.
- The number of students is relatively small.

The problem of finding feasible schedules can be posed in terms of *exact cover problems*, and so solved using known covering algorithms. In such a problem, a finite set Ω is given, together with a collection of subsets $\mathcal{Y} = \{A_n \subset \Omega : n = 1; 2; \ldots ; N\}$ with union equal to all of Ω: $\cup\, \mathcal{Y} = \cup_{n=1}^{N} A_n = \Omega$. The problem is solved when a subcollection, $\mathcal{Y}^*$ of $\mathcal{Y}$, $\mathcal{Y}^* = \{A_{n_k} : k = 1, 2, \ldots, K\}$ $\mathcal{Y}^* = \{A_{n_k} : k = 1, 2, \ldots, K\}$, of *disjoint sets* ($A_{n_k} \cap A_{n_k^a} = \emptyset$) is found, also with union equal to Ω: $\cup_{k=1}^{K} A_{n_k}$; that is, each element of Ω is contained in exactly one of the A_{n_k}s. Of course, such a subcollection may not exist. Knuth's Dancing Links implementation has been used successfully and efficiently to solve some of the well-known exact cover problems such as the N-queens and Sudoku with significantly more than the familiar nine rows, columns, and digits (Knuth, 2000). Literature searches indicate that it has never previously been applied to timetabling or scheduling.

In this paper, we present an application of Dancing Links to a complex aircrew timetabling problem in the RAN. The problem is first translated into a generalization of an exact cover problem, and then a Dancing Links implementation enumerates all feasible solutions along with their costs. Once the list of all solutions that satisfy the given constraints is generated, we apply fairly standard ILP to find the solution with the minimum cost. An important advantage of the enormous speed-up provided by Dancing Links is the possibility of using it as part of an optimization algorithm for start dates of courses, enabling a larger range of less costly schedules. Although this is a topic that will be dealt with more comprehensively in a later publication, we give some early results here.

PAST TIMETABLING SOLUTIONS AND CONTEXT

Timetabling problems are encountered in various domains, such as nurse rostering in healthcare (Bellanti et al., 2004) and train timetabling in transportation (Barrena et al., 2014). In the educational domain, in particular, numerous surveys have been published on solutions to automated school and university timetabling (Burke et al., 1997; Schaerf, 1999; Burke and Petrovic, 2002; Lewis, 2008; MirHassani and Habibi, 2013; Babaei et al., 2015).

Early attempts to model and solve timetabling problems were based on graphs and networks, with timetabling reduced to a graph coloring problem (de Werra, 1985). Mathematical programming models such as integer programming have also been used to formulate the timetabling problem (Carter, 1989). Graph coloring and integer programming, though yielding optimal solutions, are computationally expensive given the large search space of real-world timetabling problems (Carter, 1989; Costa, 1994; Burke et al., 2005). Consequently, research has shifted to metaheuristic and hybrid approaches that, though not guaranteed to yield the optimal solution, produce satisfactory solutions with respect to constraints, without computational time becoming prohibitive (Costa, 1994; Burke et al., 2005). Metaheuristic algorithms, such as Tabu search, simulated annealing, and evolutionary algorithms, their hybrids and hyper-heuristics provide state-of-the-art solutions to timetabling problems in various domains (Teoh et al., 2015; Pillay, 2016).

In the educational domain, timetabling problems are typically grouped into examination and course timetabling, with course timetabling further categorized into post-enrollment and curriculum-based timetabling (Rudová et al., 2011; Bettinelli et al., 2015; Müller and Rudová, 2016). Examination timetabling is concerned with scheduling nonrecurring events (examinations), whereas course timetabling is focused on scheduling lectures, tutorials, and so on, which recur, usually on a weekly basis.

In post-enrollment course timetabling, classes are scheduled after students have enrolled whereas, in curriculum-based course timetabling, the schedule of classes to be followed is defined before enrollment.

Military and Other Aircrew Training Applications

There are a number of reported studies on automation of military training timetabling. For example, Lee et al. (2009) described their work as ''first attempt to solve a real timetabling problem in a military training center'' (Lee et al., 2009, p. 298). In such cases, integer programming, metaheuristic algorithms and hybrids have been explored. Citing the performance limitation of integer programming alone, the United States Military Academy recently employed hybrid heuristic-integer programming to schedule examination timetables for their cadets (Wang et al., 2010), with the inclusion of examination repeats or ''makeups'' to cater for their overly constrained problem. This yielded not optimal but ''satisfactory'' answers by using variable bilinearization and decomposition to improve the initial solution followed by integer programming to obtain a locally optimal solution. The Korea Army Training Center has used an edge-coloring-based heuristic to generate ''good'' schedules in a reasonably short time (Lee et al., 2009). Scott (2005) applies ILP to produce multiyear schedules at the Defence Language Institute, maximizing the number of ''pep'' sections scheduled in a single year along with a number of secondary goals. Significant improvements (in one case by 45 percent) in the number of students accommodated in courses were achieved, although the ILP ran in under 1 minute on a regular laptop.

Much of the emphasis in military training operations research (OR) has been on training pipeline simulation and not on timetabling, per se. For example, OR has been used to explore policy effects, bottlenecks, and resource utilization (Davenport et al., 2007; Séguin and Hunter, 2013; Novak et al., 2015; Nguyen et al., 2016). A review of system dynamics simulations in workforce planning is given in Wang (2005). One possible reason for not addressing timetabling is that, given the independent and disconnected nature of course operation, delivery and involvement of numerous stakeholders, identifying and addressing specific problems (for example, bottlenecks) through simulation of the training pipeline may be considered easier than mathematical optimization of the entire training pipeline.

Branch and bound along with a family of heuristics is applied to the scheduling of the retraining of Continental Airlines pilots is described in Qi et al. (2004). A related piece of work that provides a measure of readiness for training in a tank battalion is Raffensperger and Schrage (2008); a combination of dynamic programming and column generation is used.

In military training-related scheduling problems, Yang and Ignizio (1987) discussed heuristic algorithms for a peacetime scheduling of training activities of battalions where decisions must be made regarding what kind of training tasks should be performed, by which battalion, and at what time, with the objective of minimizing the makespan. The heuristic algorithm comprises two phases: in Phase 1, a greedy heuristic is employed to generate a solution, then in Phase 2, an exchange heuristic is used to improve the solution.

McGinnis and Fernandez-Gaucherand (1994) presented a resource scheduling problem for the US Army's basic combat training program. The problem was formulated as a dynamic programming problem, however, in solving problems of real-world size, a 3-phase heuristic approach was proposed: deriving an initial feasible training resource schedule in Phase 1, if necessary, changing of level of resources in Phase 2, and an improvement heuristic in Phase 3.

Brown et al. (2013) discussed the optimization of an intratheater military airlift problem that has some strong similarity to the problem under study in this paper. The authors presented a very interesting approach where an ILP model is solved, with variables representing paths that can be described on a directed graph, and the ILP is to allocate passengers to these path variables. The variables are generated by an interesting stack iterative algorithm. In other military-related recent ILP-based work, Squires and Hoffman (2015) solve a military maintenance planning and scheduling problem,

wherein the ILPs are solved using Benders decomposition.

The military aircrew training domain presented in this paper is similar to the curriculum-based university domain in some ways. Unlike university course timetabling, where the primary concern is the allocation of courses to time periods and classrooms, here the course time periods and locations are already defined, and it is difficult to make changes to them. In aircrew training, courses require expensive resources (e.g., helicopters and instructors), and the problem to be solved is to produce a schedule for each individual student, where a schedule is an allocation of the student to precisely one of the available multiple sessions of a course, for each course in the prescribed curriculum. This has to be done so that the total time to graduate for all students is minimized. Significantly, allocation of students to courses, a core consideration in our military training, is not generally considered in the wider timetabling literature.

For aircrew training, a significant issue is the length of the overall training process. Student morale can be affected if students spend significant time waiting between courses. In any case, such occurrences are wasteful in other ways, including financial, as students in Australian Defence are paid salaries throughout these waiting periods. Accordingly, in this work we adopt the simple view that the cost of a schedule for an individual student is the total time (*makespan*) from the start date of the first course to the end date of the last course. Other costs are possible; in particular, a mixture of financial and time costs, or a cost that penalizes very high costs for individuals (compensated by lower costs for the rest of the cohort). Such a penalization might involve a quadratic function of the total time extent of a student's studies. This penalization would require only a minor change to our approach, and is not discussed here.

DANCING LINKS

Dancing Links is a technique popularized by Donald Knuth to efficiently implement his Algorithm X (Knuth, 2000). The latter is a backtracking algorithm to solve exact cover problems. In his paper (Knuth, 2000), he refers to the algorithm as the ''most obvious trial-and-error approach'' for finding all solutions to an exact cover problem. More technically, Algorithm X is a recursive, nondeterministic, depth-first, backtracking algorithm. The key innovation of Knuth is not the algorithm itself but the way this algorithm is implemented and we will describe that next. It is important to keep in mind that what is being implemented is just fairly standard backtracking. The computations are as rapid as they are because of its implementation.

The generic problem that Knuth addresses is an exact cover problem as described earlier. He used a 0–1 matrix to describe it, where, in the notation introduced earlier, the columns are indexed by the elements of Ω and each row represents one of the given subsets of Ω, so that the rows are indexed by $\mathcal{Y}$ (Figure 1). Then a solution is a set of rows (corresponding to $\mathcal{Y}^*$) containing exactly one 1 in each column. Such a solution may, or may not, exist. This kind of matrix formulation will be useful for our problem.

The basic exact cover problem can further be generalized where the constraints are divided into two sorts: *primary* and *secondary*. Primary constraints must be satisfied by *exactly one* selection as in the basic case, while the secondary constraints can be satisfied by *at most one* selection. In the matrix formulation, the solution set of rows $\mathcal{Y}^*$ has exactly one 1 in each of the

0	0	1	0	1	1	0
1	0	0	1	0	0	1
0	1	1	0	0	1	0
1	0	0	1	0	0	0
0	1	0	0	0	0	1
0	0	0	1	1	0	1

Figure 1. 0–1 matrix.

primary columns, and at most one 1 in each of the secondary columns.

The standard backtracking implementation of Knuth's Algorithm X spends a significant amount of time searching for 1s in the matrix. To improve this search time from complexity $O(n)$ to $O(1)$, Knuth represented the 0–1 matrix as a sparse matrix, where only 1s are stored in the circular doubly linked list data structure. Each Node in the linked list points to the adjacent Node to the left and right (1s in the same row), above and below (1s in the same column). Each column also includes a ColumnNode that acts as a ''header'' for that column, and may optionally track relevant information pertaining to its column. Each Node has a reference to its ColumnNode. All ColumnNodes are linked into a circular list. This data structure is the key to extremely fast implementation of Algorithm X.

Using Dancing Links, called DLX by Knuth, the algorithm quickly selects sets of rows as possible solutions and efficiently performs backtracking steps. In this doubly linked list implementation, moving down the tree is provided by deletion of all links in rows and columns associated with a specific 1 in the matrix, and backtracking is provided by reinstating these links. This can be done very quickly, both in terms of computing speed and code length. Knuth (2000) describes this algorithm in detail.

PROBLEM FORMULATION

Figure 2 shows one instantiation of the syllabus-course-prerequisite structure captured by a directed graph with nodes representing the individual courses. The solid arrows between nodes indicate prerequisite order, whereas the dashed arrows indicate mutually exclusive paths to courses that belong to different syllabuses. This graph represents two syllabuses: $\{C_1, C_2, C_3, C_4, C_5, C_7,\}$ and $\{C_1, C_2, C_3, C_4, C_6, C_8,\}$. These might, for instance, correspond to training of pilots with different specializations in later courses.

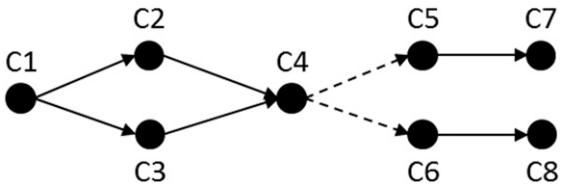

Figure 2. Simplified course prerequisite structure.

To describe the generic problem, we first define a *syllabus* to be a consistent set of courses, where by *consistent* we mean that it includes all prerequisites of every course in the set. For each course, there is a set of *sessions* that are parallel instantiations of the course at specified times and dates. Successful completion of any one of the course sessions corresponding to a course means that that course itself has been passed. A *schedule* for a syllabus is a sequence of course sessions that is consistent in the following sense:

- Con-1) It contains exactly one session for each required course in the syllabus.
- Con-2) Sessions in the sequence occur in prerequisite order for courses.
- Con-3) There are no time clashes.

Ultimately, our aim is to assign students to schedules in such a way as to minimize the summed makespan over all students. As discussed earlier, we divide our overall problem into one of first finding all schedules and then optimally assigning the students to those schedules in ways to be described later. The optimization has some nuances that are not relevant to the enumeration of schedules. Given that, it will be convenient to focus in this section entirely on the problem of finding all schedules. Later we will discuss the optimization problem.

We introduce the following definitions and notation:

- C is the set of all courses.
- Ψ is the set of all syllabuses of interest, and we use Y with sub/superscripts as notation for members of Ψ.
- We write $c \prec c'$ $(c, c' \in C)$ to mean that course c is a prerequisite to c'. We remark that, in our case, this prerequisite structure is syllabus independent.
- For a given course $c \in C$,
  - $T(c)$ is the set of all sessions of c, and
  - $\sigma(c)$ is the set of all prerequites of c, courses (either as a direct prerequisite or implied prerequisite).

- For a session e, t_e^s and t_e^f are its start and finish times.
- Given two sessions $e \in T(c)$ and $e' \in T(c')$, where c and c' may or may not be the same, $e < e'$ means that e is completed before e' starts ($t_e^f < t_{e'}^s$), and $e \perp e'$ means that the time extents of these two sessions do not overlap (that is, either $e < e'$ or $e' < e$).
- A schedule for a syllabus $S \in \Phi$ (regarding S as a subset of C) is a function $f : S \rightarrow \cup_{c \in C} T(c)$ such that $f(c) \in T(c)$, $f(c) \perp f(c')$ for all $c \neq c'$ in C, and if $c \prec c'$ then $f(c) < f(c')$.
- S^Y is the set of all possible schedules for Syllabus Y ($Y \in \Psi$).

DLX-ILP TIMETABLING

DLX-ILP described here consists of two distinct phases. First, a list of all feasible schedules, along with their costs, is computed using a Dancing Links implementation. Second, ILP is used to assign students optimally to the feasible schedules. First, we describe how we convert the timetabling problem into a generalized cover problem.

Translating Timetabling into a Generalized Cover Problem

We illustrate how this is done with an example: Table 1 is converted into a generalized cover problem so as to capture all constraints. This is done in terms of a 0–1 matrix (Figure 3).

Rows represent courses and their sessions. The first constraint (Con-1)—that each course in a syllabus must occur exactly once in the schedule—is represented by the first set of columns, labeled *Constraint 1* in the matrix. For instance, given the course structure shown in Figure 2 and the matrix in Figure 3, choice of session c_{51} will not only exclude other c_5 sessions from the solution, it will also automatically remove all c_6 and c_8 sessions from the search space. Conversely, if c_6 is included in a solution, then both c_5 and c_7 will be excluded from that solution.

The second constraint (Con-2) is enforced by the second set of columns labeled Constraint 2. For any pair c_{ij} and c_{pl}, where c_{pl} is a prerequisite of c_{ij} ($c_{pl} \prec c_{ij}$), we consider three cases:

1. $c_{ij} > c_{pl}$
2. $c_{ij} < c_{pl}$
3. $c_{ij} \not\perp c_{pl}$

No issue ensues when $c_{pl} < c_{ij}$ (Case 1). When $c_{ij} < c_{pl}$ (Case 2) or when part or whole of $c_{ij} \not\perp c_{pl}$ (Case 3), c_{ij} and c_{pl} must not be included in the same schedule. Observe that, in fact, Case 3 is handled by Constraint 3 (Con-3), whereas Constraint 2 is only concerned with Case 2. For instance, $c_1 \prec c_2$, but $c_{21} < c_{12}$ and $c_{21} < c_{13}$, entailing that sessions c_{21} and c_{12} are mutually exclusive, as are sessions $c_{21}$and c_{13}. These are represented by columns c_{12} and c_{13} in Figure 3.

Lastly, Con-3) ensures that no two or more course sessions included in a solution have overlapping time slots. For example, $c_{21} \not\perp c_{11}$, and so only one of these sessions will be included in a schedule.

Constraint 1 is governed by $n - k + 1$ columns, where n is the number of all courses and k is the number of branches sharing the same prerequisite. In the case of parallel branches, the first course in each branch is aggregated into one column, and labeled accordingly. The columns in Constraint 2 are labeled so as to allow at most one course session to be chosen from a set of prerequisite-incompatible sessions. Constraint 3 is governed by p columns, where p is the number of time slots during which a session is taking place. The primary columns for the generalized cover problem are those governing Constraint 1, whereas Constraints 2 and 3 appear as secondary columns.

One way to deal with a secondary (non-exact) constraint is to convert it into a standard exact cover constraint by appending an extra row, for each secondary column, that has a single 1 in the secondary column and 0 in every other column. Thus, if a candidate solution does not cover a secondary column in the original rows, for instance, there is a time break between selected course sessions, then the columns for those time slots would be empty, it will cover the column in the extra row created.

However, Knuth (2000) suggests that it would be more efficient to work with the generalized

cover problem directly by slightly modifying the data structure. All the headers for primary columns are linked into a circular list, as in the exact case, but all the headers for secondary columns just link back to themselves. This approach is particularly beneficial in our case, as adding one more row for each secondary column, of which there are many, significantly increases the size of the search space.

Algorithm 1 automatically converts the scheduling problem described above into the generalized cover problem, by directly creating the doubly linked list data structure that represents the 0–1 matrix in Figure 3. We fix a syllabus Y. For convenience, here we write c_{ij} for a session corresponding to course c_i.

For readability, this algorithm omits statements about linking Nodes and ColumnNodes to enforce circular rows and columns where appropriate.

Generation of All Feasible Solutions

We have made a slight modification to Knuth's algorithm to speed-up computation in our case. Knuth's approach considers a solution to be found when no more primary columns remain to be covered. Instead, we specify that a solution is found when all end courses in a syllabus are included in the solution: the prerequisite structure forces all other courses to be included.

Another modification to Knuth's algorithm we have made is choosing the right pivot. A careful choice of a pivot can significantly affect running time. In our case, a pivot is a primary column and choosing a pivot contained in as few sets as possible will lead to a smaller subproblem in the next iterations, and thus a smaller subsearch space.

Once a doubly linked list data structure is created as described previously, the modified DLX generates all feasible solutions. A solution contains a set of rows, where there would be exactly one 1 per primary column, and at most one 1 per secondary column. A complete set of all solutions is the set of all feasible schedules that satisfy all the constraints (the selection of rows, i.e., course session sequences that strictly follow the set-out constraints). In our simplified case, depicted in Figure 3, there are nine feasible schedules, each with its cost (makespan):

Table 1. Input data.

c_{ij}	Prerequisites	t_s	t_e
c_{11}	—	1	5
c_{12}	—	5	9
c_{13}	—	8	12
c_{21}	1	4	4
c_{22}	1	7	7
c_{23}	1	10	10
c_{31}	1	6	7
c_{32}	1	8	9
c_{33}	1	12	13
c_{41}	2, 3	7	7
c_{42}	2, 3	10	10
c_{43}	2, 3	11	11
c_{51}	4	5	9
c_{52}	4	11	15
c_{53}	4	13	17
c_{61}	4	7	9
c_{62}	4	10	12
c_{63}	4	14	16
c_{71}	5	15	15
c_{72}	5	18	18
c_{81}	6	13	13
c_{82}	6	17	17

$$\{c_{11}\, c_{22}\, c_{32}\, c_{42}\, c_{52}\, c_{72}\},\ M_1 = 18$$
$$\{c_{11}\, c_{22}\, c_{32}\, c_{42}\, c_{53}\, c_{72}\},\ M_2 = 18$$
$$\{c_{11}\, c_{22}\, c_{32}\, c_{42}\, c_{63}\, c_{82}\},\ M_3 = 17$$
$$\{c_{11}\, c_{22}\, c_{32}\, c_{43}\, c_{53}\, c_{72}\},\ M_4 = 18$$
$$\{c_{11}\, c_{22}\, c_{32}\, c_{43}\, c_{63}\, c_{82}\},\ M_5 = 17$$
$$\{c_{11}\, c_{23}\, c_{31}\, c_{43}\, c_{53}\, c_{72}\},\ M_6 = 18$$
$$\{c_{11}\, c_{23}\, c_{31}\, c_{43}\, c_{63}\, c_{82}\},\ M_7 = 17$$
$$\{c_{11}\, c_{23}\, c_{32}\, c_{43}\, c_{53}\, c_{72}\},\ M_8 = 18$$
$$\{c_{11}\, c_{23}\, c_{32}\, c_{43}\, c_{63}\, c_{82}\},\ M_9 = 17 \qquad (1)$$

Small example. We consider a small example with four courses, where $c_1 \prec c_2$, $c_1 \prec c_3$, $c_2 \prec c_4$ and $c_3 \prec c_4$. A 0–1 matrix for this example is the starting state in Figure 4.

	Constraint 1							Constraint 2									Constraint 3															
	C_1	C_2	C_3	C_4	C_5C_6	C_7	C_8	C_{12}	C_{13}	C_{13}	C_{23}	C_{32}	C_{33}	C_{42}	C_{43}	C_{63}	w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17	w18
C_{11}	1																1	1	1	1	1													
C_{12}	1							1													1	1	1	1	1								
C_{13}	1								1	1														1	1	1	1	1						
C_{21}		1						1	1											1														
C_{22}		1							1														1											
C_{23}		1									1															1								
C_{31}			1							1												1	1											
C_{32}			1									1												1	1									
C_{33}			1										1														1	1						
C_{41}				1							1	1	1										1											
C_{42}				1									1	1												1								
C_{43}				1									1		1												1							
C_{51}					1		1							1	1						1	1	1	1	1									
C_{52}					1		1																				1	1	1	1	1			
C_{53}					1		1																						1	1	1	1	1	
C_{61}					1	1								1	1								1	1	1									
C_{62}					1	1																				1	1	1						
C_{63}					1	1										1														1	1	1		
C_{71}						1																									1			
C_{72}						1																												1
C_{81}							1									1												1						
C_{82}							1																									1		

Figure 3. A 0–1 matrix for a simplified course structure. The shaded cells denote mutually exclusive paths and courses.

DLX is initiated by choosing the primary column c_3 (least number of 1s) and puts c_{31} in a partial solution while eliminating (covering) row c_{13} because of a time clash (Constraint 3). Next, it has two options, c_{11} or c_{12}. Choice of c_{11} eliminates rows c_{12} (Constraint 1) and c_{21}(Constraint 2). Now it has only one option and picks c_{22}. Then c_{41} is chosen to complete a solution. The DLX then backtracks and chooses c_{42} as a second solution. The algorithm then backtracks further to when c_{31} was chosen and puts c_{12} into the partial solution, while covering c_{11} (constraint 1), c_{21} (Constraint 2), and c_{22} (Constraint 3). As primary column c_2 is now empty, the algorithm returns with no solution for this branch. By backtracking in this way along all possible branches of the tree, DLX enumerates all feasible solutions.

Costing Solutions

The cost of each schedule is its makespan as mentioned earlier. Because each generated schedule is a collection of course sessions that are unordered, extra steps are required to sort the sessions into a chronological order using a standard merge-sort approach with time complexity $O(n\,log(n))$ and space complexity $O(n)$. It is important to perform this sorting prior to applying the ILP algorithm, described in the next section, which takes into account the pass rates and maximum number of students per session. We note that the number of solutions provided by DLX may be very large; even so, we have been able to solve reasonably large problems satisfactorily.

Integer Linear Programming Formulation

Having obtained the costed list of schedules, we now turn to optimization. This will require a little more notation in addition to that given earlier. We present our ILP below using the *define before use* format of Teter et al. (2016).

Algorithm 1. Algorithm for converting the described scheduling problems into the generalized cover problem.

for each course $c_i \in Y$ **do**
  for each session j **do** ▷ Start with Constraint 1
    if c_i has a mutually exclusive $c_{i'}$ that shares the same prerequisite **then**
      if $c_{i'}$ is not created **and** $j = 0$ **then**
        create a `ColumnNode` $c_{ii'}$
      else
        retrieve `ColumnNode` $c_{ii'}$
    else if $j = 0$ **then**
      create a `ColumnNode` c_i
    else
      retrieve `ColumnNode` c_i
    create a `Node` for c_{ij} under `ColumnNode` c_i or $c_{ii'}$
    if c_i has a prerequisite $c_r \in Y$, with a mutually exclusive $c_{r'}$ sharing the same prerequisite **then**
      create a `Node` for each session of $c_{r'}$ under `ColumnNode` c_i, if not already created
    for each prerequisite $c_p \prec c_i$ **do** ▷ Constraint 2
      for each session ℓ of c_p **do**
        if $c_{ij} < c_{p\ell}$ **then**
          if `ColumnNode` $c_{p\ell}$ is not created for c_i **then**
            create `ColumnNode` $c_{p\ell}$
          else
            retrieve `ColumnNode` $c_{p\ell}$ for c_i
          create a `Node` for c_{ij} **and** a `Node` for $c_{p\ell}$ under `ColumnNode` c_ℓ
    for $t_x \leftarrow t_s \ldots t_e$ **do** ▷ Constraint 3
      if `ColumnNode` t_x is not created **then**
        create `ColumnNode` t_x
      else
        retrieve `ColumnNode` t_x
      create a `Node` for c_{ij} under `ColumnNode` t_x

- S^Y is the set of all schedules for syllabus Y.
- T^Y is the set of all students required to undertake syllabus Y.
- p_c is the pass rate of Course c (independently of sessions).
- N is the total number of students.
- $a_{c,e}^{Y,s} = \begin{cases} 1, & \text{if Session } e \text{ of Course } c \text{ is selected for Schedule } s \in S^Y \text{ of Syllabus Y} \\ 0, & \text{otherwise.} \end{cases}$
- M_{ce} is the number of places available for Session e of Course c.
- z_s^Y is a decision variable (non-negative integer) indicating the number of students assigned to Schedule $s \in S^Y$.
- m_s^Y is the cost (makespan) of Schedule s of Syllabus Y.

We define the probability of a student surviving from recruitment until course c in Syllabus Y as

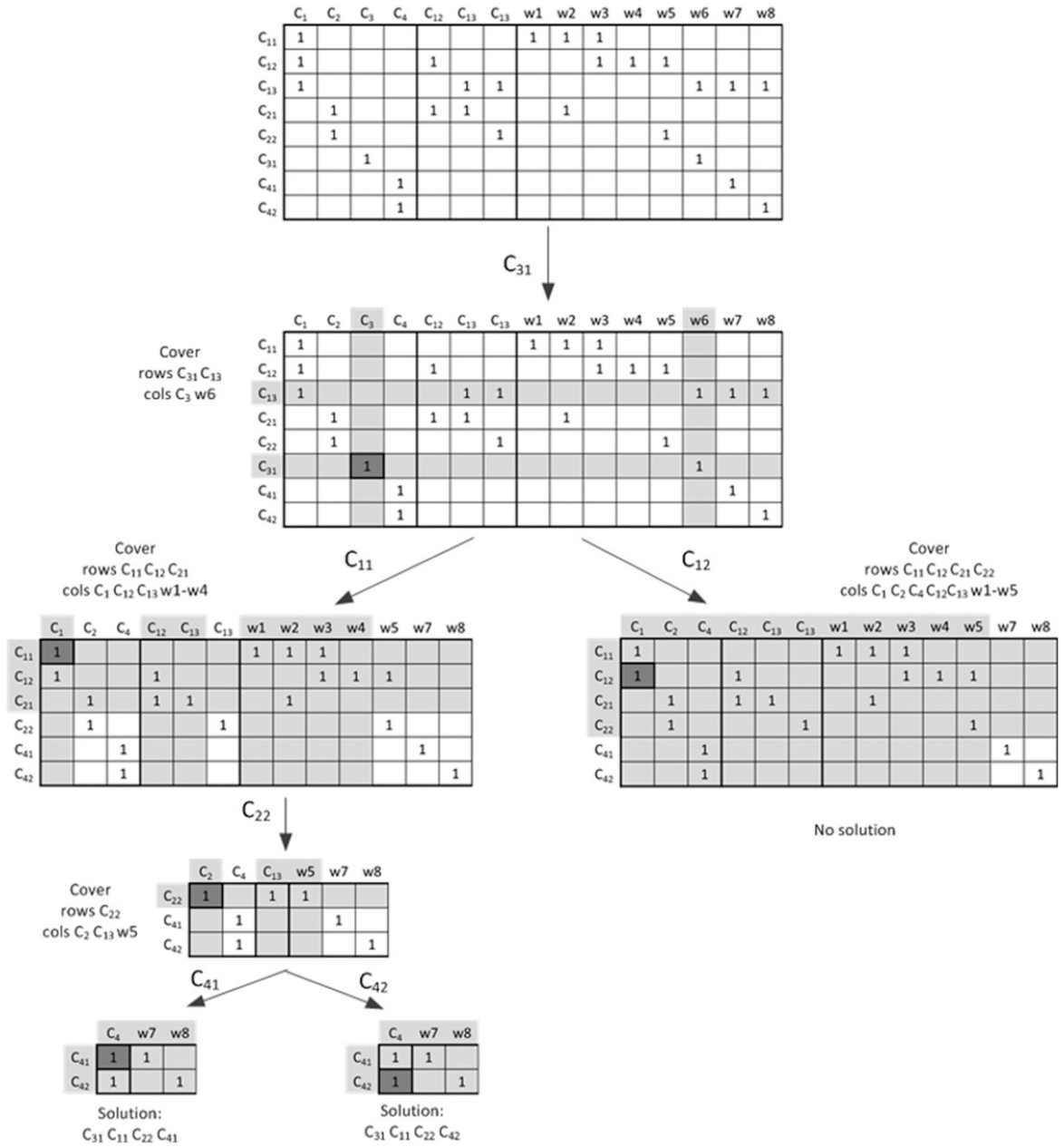

Figure 4. Walk-through of the small example. Greyed out cells denote the rows and columns that were covered at each particular step of DLX.

$$\mathbb{P}(Y, c) = \prod_{c' \prec c} p_{c'}$$

where, as is standard, an empty product is equal to 1.

Assume Course $c \in Y$. Then, admission of z_s^Y students into Schedule $s \in S^Y$ of Syllabus Y, yields an expected number of students remaining when Course c commences equal to the probability of a student passing all precedence courses $\mathbb{P}(Y, c)$ multiplied by z_s^Y.

The optimization is given below.

Model 1

$$\min \sum_{Y\in\Psi}\sum_{s\in S^Y} m_s^Y y_s^Y \tag{2}$$

$$\text{s.t.} \sum_{Y\in\Psi}\sum_{s\in S^Y} a_{e,c}^{Y,s}\mathbb{P}(Y,c)z_s^Y \le M_{c,e},\ \forall j\in T_c, c\in C \tag{3}$$

$$\sum_{Y\in\Psi}\sum_{s\in S^Y} y_s^Y = N \tag{4}$$

$$\delta(c_{ij},S^K)=\begin{cases}1, & \text{if } c_{ij} \text{ is a course session in}\\ & \text{the course schedule } S^K\\ 0, & \text{otherwise}\end{cases} \tag{5}$$

The cost function is predicated on the assumption that even failing students incur the same cost as those who are retained for the full extent of the schedule. This is not ideal, and later work will include costs for partial completion of courses. In fact, it is not hard to modify the approach here to include such partial costs.

SIMULATIONS

Application in Royal Australian Navy

The Training Authority Aviation (TA-Avn) is an organization within RAN responsible for managing aviation-specific training for all RAN personnel who are to be employed in an aviation-related job category. TA-Avn maintains responsibility for individual student aircrew, engineers, and technicians throughout their journey through the relevant training continuum until their posting to an operational squadron.

In a temporal sense, the bulk of aircrew training consists of a sequence of major, structured courses that constitute the backbone of the RAN training continuum. Students also have to complete a number of mandatory short courses that have a limited prerequisite structure. Figure 5 represents a course structure for one RAN aircrew type. Both short and long courses are run repeatedly throughout a year with a fixed number of repetitions (course sessions). The short courses are often run by other military or external organizations where TA-Avn has little influence over the scheduling of dates and allocation of places within the course. Dates for some course sessions, therefore, may not be known more than 12 months in advance or may be changed with short notice. In the case of military agencies, this occurs because of continually changing and competing demands on limited infrastructure resources as well as uncertainty in the availability of required and suitably qualified instructors. It is critically important to be able to quickly and easily regenerate a new timetable at short notice.

The difficulties in managing the aviation training continuum, particularly for officer aircrew, are further exacerbated by the high and extremely variable course pass rates, making it difficult to construct reliable predictions of numbers of students feeding into operational squadrons. Given that adequate resourcing of helicopter platforms is synonymous with achieving capability, even small fluctuations in supply are extremely costly.

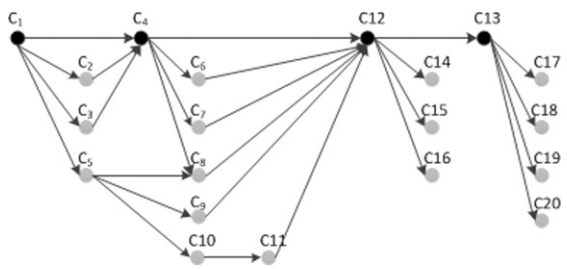

Figure 5. RAN course structure for one aircrew type. Black nodes denote major structured courses. Gray nodes are short courses that have more relaxed prerequisite structure.

It is even more important, in this context, to produce timely optimal timetabling solutions.

Currently, the comprehensive scheduling and planning of an individual specific training continuum is undertaken manually using Microsoft Excel (2010) spreadsheets. Biennial rotations of the staff involved (in keeping with the RAN's posting cycle), together with the manual nature of this detailed training continuum planning, makes it particularly vulnerable through the handover and early succession period. In the Bayliss et al. (2016) paper, we used Tabu search to mitigate such problems, by automating the timetabling solution. Here, the objective is to go further and avoid heuristic optimization techniques which are (without guarantees) close to optimal, but rather ask the question ''can we provide an optimal solution in a reasonable time?''

The answer is ''yes'' if we use a Dancing Links ILP implementation.

Comparison with a Backtracking Algorithm for Generating All Feasible Schedules

This section compares the speed of generating all feasible schedules using DLX with the standard depth-first backtracking algorithm (Tarjan, 1972). Both programs were implemented in Java (Oracle, 2018) and runs were performed single-threaded on the same computer (Intel Core i7 processor with 16 Gbytes RAM) for comparison of executing times. Each batch was run 10 times and the average runtime is reported here. Table 2 provides the result of comparing the runs using the DLX and backtracking approaches to generate all schedules for datasets with different number of courses and number of sessions per course. As seen, for a small number of courses the DLX and backtracking approaches are quite comparable in producing a relatively small number of all solutions (lines 1–4). However, for datasets with a larger number of courses, leading to millions of feasible schedules (lines 5–8), DLX is shown to be much faster (between 7 and 181 times) than the backtracking approach.

For dataset 9 with 25 courses and 12 sessions each, no solution can be generated but DLX took 253 ms while backtracking took 3.5 min to terminate the search. The last two datasets particularly show the strength of DLX. DLX produced solutions in both cases in less than a minute, while standard backtracking failed to produce solutions after one hour of running.

The start and end times of course sessions can significantly affect the total number of feasible schedules, given the constraints. Shifting them around slightly can produce a much higher number and, therefore, could prove to be beneficial when scheduling courses. Although standard backtracking would require prohibitive computational time, DLX is fast enough to enable a search over adjustments of the time extents

Table 2. Comparison of DLX against backtracking runs for varying number of courses and number of sessions of courses. Computation is limited to one hour.

				Solution time	
No.	# courses	# sessions/course	# schedules	DLX(ms)	Backtracking (ms)
1	5	12	2,941	3	2
2	5	12–18	6,133	2	6
3	5	12–24	11,881	3	5
4	6	10–15	549	1	1
5	15	12	2,084,733	929	168,254
6	15	15–22	38,056,034	7,004	52,553
7	20	8–10	3,229,134	289	3,094
8	24	8–10	31,368,744	2,756	90,920
9	25	12	0	253	207,723
10	25	12–24	1,728,875	24,144	Incomplete result
11	15	12–18	132,609,631	40,181	Incomplete result

to reproduce a complete set of schedules. Table 3 shows exactly that. Our datasets here consist of 20 courses and between eight and 11 sessions per course. The dates for major courses, most of which are real dates scheduled by the training authorities, were fixed. The dates for minor courses were slightly adjusted. The number of possible schedules increased from 600 thousand to 3 million, all of which were generated in less than one second by the DLX. Backtracking took at least 10 times longer. In this way, the speed of DLX makes it a useful tool as part of a larger search algorithm for course time adjustment.

The advantage of formulating our timetabling problem as a generalized cover problem to generate all feasible schedules is twofold. First, to lay out all courses and their sessions in such a way so that those sessions that clash with each other due to constraints, such as prerequisites and time order and/or clashes, become mutually exclusive. This naturally leads to the next advantage of this approach, with each step of finding the next candidate c_{ij}, the number of branches that are automatically removed can be quite large, significantly reducing the subsearch space.

Comparison with an ILP Solution for Generating and Allocating Optimal Schedules

This section compares the efficiency of the DLX-ILP method with an equivalent formulation relying exclusively on ILP for both the generation and allocation phases.

The ILP approach differs from the DLX-ILP method by solving schedule generation and allocation simultaneously within a single ILP. This is achieved by fixing the number of students, and defining decision variables representing the allocation of each individual student to each course session. Further variables represent the flow of individual students through the training program, along with the cumulative pass rates at each stage. A solution to this ILP defines a feasible schedule for each individual student within the overall constraints of the training program, where multiple students may receive identical schedules. As in DLX-ILP, the cost function is engineered to minimize the sum of the students' schedule makespans.

This formulation circumnavigates the need to enumerate and represent all feasible schedules; instead, the optimal set of schedules (along with their allocations) can be extracted from the solution. However, this formulation also incurs a high degree of symmetry in the set of optimal solutions: when many students receive distinct schedules, a large number of permutations will exist; hence a large number of optimal solutions to the ILP may be encountered.

Both programs were implemented in Java (Oracle, 2018) and CPLEX (ILOG, 2018). All evaluations were run in the ''Spartan'' high-performance computing environment. Each job was allocated eight cores, 20 Gbytes RAM and a maximum run time of one hour. In total, 13 distinct scenarios with varying numbers of courses and sessions were evaluated, and their individual runtimes (or timeouts) are reported in Table 4.

Table 3. Comparison of DLX against backtracking runs for the same number of courses (20) and sessions per course (8-11) but with varying time extents.

# schedules	DLX(ms)	Backtracking (ms)
607,194	58	620
739,094	64	675
948,840	5	964
1,057,472	110	1,016
1,270,434	152	1,400
1,686,102	203	1,859
1,700,080	173	4,408
2,159,660	217	2,129
3,229,134	289	3,094

Table 4. Comparison of DLX-ILP against ILP runs for varying number of courses and sessions per courses.

No.	# courses	# sessions/course	# students	Solution time	
				DLX-ILP (s)	ILP (s)
1	5	12	24	0.15	91.9
2	15	12	24	302.9	Timeout
3	25	12	24	Infeasible	Infeasible
4	5	12–18	24	0.46	Timeout
5	15	12–18	24	Timeout	Timeout
6	25	12–18	24	Infeasible	Infeasible
7	5	12–24	24	0.6	88.7
8	15	12–24	24	Timeout	Timeout
9	25	12–24	24	Timeout	Timeout
10	6	10–17	24	0.09	0.509
11	20	5–11	24	8.89	Timeout
12	20	5–11	24	11.35	142.1
13	20	6–11	24	14.44	Timeout

In scenarios where both methods completed, the DLX-ILP method had a speed advantage of around 5 to 613 times faster. Among the results in which ILP timed out, the advantage was at least 11 times faster. In three scenarios, both methods reached the above imposed time limit, whereas the ILP method timed out in over half the scenarios overall.

A more detailed sweep over the numbers of courses and number of students within the ILP method revealed a dramatic increase in runtime when both parameters exceeded 15. Although the ILP method requires substantially fewer variables (linear in the number of sessions and students, and at worst quadratic in the number of courses), the constraints required to represent the flow of each individual student and cumulative pass rates introduce significant complexity. On the other hand, whereas DLX-ILP requires a large number of variables (linear in the number of feasible schedules generated by DLX), the computation for student flows and pass rates occurs in preprocessing.

To understand the difference in performance, we considered the gaps between the integral solutions and linear programming (LP) relaxation solutions of each formulation. In case of DLX-ILP, most of the constraints are handled in the DLX preprocessing phase, hence the LP relaxation allows nonintegral values only for numbers of students allocated to each schedule. However, the LP relaxation of ILP solution permits nonintegral values of all variables including allocations to individual course sessions and movements. This in turn means that fractional students are floating not just through the allocation part of the problem, but also generation of feasible schedules. In most instances, the LP relaxation of the DLX-ILP yielded the integral solution immediately, and otherwise yielded a solution with a gap of less than 1 percent from the optimal solution. By contrast, the LP relaxation of ILP typically yielded solutions with gaps from the integral solution of greater than 80 percent. Hence, the DLX-ILP formulation provides a far tighter representation of the integral problem, whereas ILP spends substantially more time closing the gap via branch and bound.

DISCUSSION

The *key* proposals in this paper are:

- to separate the generation of feasible timetables and their costs from the problem of allocating students to such timetables, and
- to use Knuth's Dancing Links algorithm to generate the feasible timetables.

From that point on, our approach solves the allocation problem by using ILP for obtaining exact solutions. Each process has its own complexity and limitations. If the complexity of the timetable and the repetitions of courses results in a very

large number of schedules, the problem may just be too big for the ILP to be computationally tractable. A mitigating factor in our situation is that the number of students, N, is relatively small. As a result, since we are only interested in integer solutions, the solution lies in a linear subspace with small dimension. This may enable more efficient ILP approaches. Determining the appropriate low dimensional subspace forms part of our future work. It might be thought that we only need to consider the N lowest cost schedules in a greedy manner, but it quickly becomes apparent that this may not be optimal as a higher-cost schedule for some students may permit other students to have lower-cost schedules. However, an approach to limit the computational complexity is to only choose the M least costly schedules, where M is much larger than N but still feasible from a computational viewpoint.

We believe that the use of Dancing Links opens up the possibility of attacking quite large scheduling problems of this kind, and we propose future work in several directions to explore these possibilities. Clearly the complexity will become computationally infeasible at some point, and stochastic models will be even harder to solve. Therefore we will explore metaheuristics or hybrid heuristic and branch-and-bound type methods. We are also considering an on-the-fly use of the Dancing Links phase to combine interactively with the ILP phase, allowing the latter to tackle a sequence of lower dimensional problems.

A formal complexity analysis will be conducted and further applications of the approach will be performed to timetabling problems with additional constraints that require an alternative, problem specific cost function.

CONCLUSION

We have shown how Knuth's DLX can be applied effectively to generate feasible course schedules incorporating multiple repetitions of courses, time clashes, prerequisites, and rigid course requirements. DLX enables extremely fast computation of millions of such schedules, thus enabling searches over adjustments of time extents of course repetitions to increase the number of feasible schedules. A standard ILP approach obtains minimal overall cost assignments of students to course schedules and it works well for a realistic Navy aircrew sample datasets.

Future work will develop these ideas in several directions including further improvements in the ILP phase, deployment of focused metaheuristic methods as a replacement for ILP, and more efficient integration of the DLX and ILP phases.

ACKNOWLEDGEMENTS

The authors would like to thank Mina Shokr for the implementation of the backtracking algorithm.

REFERENCES

Babaei, H., Karimpour, J., and Hadidi, A. 2015. A Survey of Approaches for University Course Timetabling Problem, *Computers and Industrial Engineering*, Vol 86, 43–59.

Barrena, E., Canca, D., Coelho, L. C., and Laporte, G. 2014. Exact Formulations and Algorithm for the Train Timetabling Problem with Dynamic Demand, *Computers and Operations Research*, Vol 44, 66–74.

Battiti, R., and Tecchiolli, G. 1994. The Reactive Tabu Search, *ORSA Journal on Computing*, Vol 6, No 2, 126–140.

Bayliss, C., Novak, A., Nguyen, V., Moran, B., Caelli, T., Harrison, S., and Tracey, L. 2016. Optimising Aircrew Training Schedules Using Tabu Search, *Australian Simulation Congress* (SimTecT).

Bellanti, F., Carello, G., Croce, F. D., and Tadei, R. 2004. A Greedy-Based Neighborhood Search Approach to a Nurse Rostering Problem, *European Journal of Operational Research*, Vol 153, No 1, 28–40.

Bettinelli, A., Cacchiani, V., Roberti, R., and Toth, P. 2015. An Overview of Curriculum-Based Course Timetabling, *TOP*, Vol 23, No 2, 313–349.

Brown, G. G., Carlyle, W. M., Dell, R. F., and Brau, J. W. 2013. Optimizing Intratheater Military Airlift in Iraq and Afghanistan, *Military Operations Research*, Vol 18, No 3, 35–52.

https://calhoun.nps.edu/handle/10945/38129.

Burke, E., Dror, M., Petrovic, S., and Qu, R. 2005. Hybrid Graph Heuristics within a Hyper-Heuristic Approach to Exam Timetabling Problems. *The Next Wave in Computing, Optimization, and Decision Technologies.* Springer, 79–91.

Burke, E., Jackson, K., Kingston, J., and Weare, R. 1997. Automated University Timetabling: The State of the Art, *The Computer Journal*, Vol 40, 565–571.

Burke, E. K., and Petrovic, S. 2002. Recent Research Directions in Automated Timetabling, *European Journal of Operational Research*, Vol 140, No 2, 266–280.

Carter, M. W. 1989. A Lagrangian Relaxation Approach to the Classroom Assignment Problem, *INFOR: Information Systems and Operational Research*, Vol 27, No 2, 230–246.

Costa, D. 1994. A Tabu Search Algorithm for Computing an Operational Timetable. *European Journal of Operational Research*, Vol 76, No 1, 98–110.

Davenport, J., Neu, C., Smith, W., and Heath, S. 2007. Using Discrete Event Simulation to Examine Marine Training at the Marine Corps Communication-Electronics School. *2007 Winter Simulation Conference*, 1387–1394.

de Werra, D. 1985. An Introduction to Timetabling. *European Journal of Operational Research*, Vol 19, No 2, 151–162.

de Werra, D. 1997. The Combinatorics of Timetabling, *European Journal of Operational Research*, Vol 96, No 3, 504–513.

Dell'Amico, M., and Trubian M. 1993. Applying Tabu Search to the Job-Shop Scheduling Problem, *Annals of Operations Research*, Vol 41, Nos 1–4, 231–252.

Even, S., Itai, A., and Shamir, A. 1975. On the Complexity of Time Table and Multi-commodity Flow Problems. *Proceedings of the 16th Annual Symposium on Foundations of Computer Science* (SFCS 75). IEEE Computer Society, 184–193.

Glover, F., and Laguna, M. 1997. *Tabu Search.* Kluwer Academic Publishers.

Hertz, A., and Widmer, M. 1996. An Improved Tabu Search Approach for Solving the Job Shop Scheduling Problem with Tooling Constraints, *Discrete Applied Mathematics*, Vol 65, No 1, 319–345.

ILOG. 2018. IBM ILOG CPLEX V12.8: User's manual for CPLEX.

Johnson, R., and Pilcher, M. G. 1988. *The Traveling Salesman Problem*, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, eds. John Wiley & Sons, 1985. Book review. *Networks*, Vol 18, No 3, 253–254.

Karp, R. M. 1972. *Reducibility among Combinatorial Problems.* Springer, 85–103.

Knuth, D. E. 2000. Dancing Links, *Millennial Perspectives in Computer Science*, 187–214.

Lee, S.-Y., Kim, Y.-D., and Lee, H.-J. 2009. Heuristic Algorithm for a Military Training Timetabling Problem, *Asia Pacific Management Review*, Vol 14, No 3, 289–299.

Lewis, R. 2008. A Survey of Metaheuristic-Based Techniques for University Timetabling Problems, *OR Spectrum*, Vol 30, No 1, 167–190.

McGinnis, M. L., and Fernandez-Gaucherand, E. 1994. Resource Scheduling for the United States Army's Basic Combat Training Program, *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, Vol 1, 553–558.

MirHassani, S. A., and Habibi, F. 2013. Solution Approaches to the Course Timetabling Problem, *Artificial Intelligence Review*, Vol 39, No 2, 133–149.

Müller, T., and Rudová, H. 2016. Real-Life Curriculum-Based Timetabling with Elective Courses and Course Sections, *Annals of Operations Research*, Vol 239, No 1, 153–170.

Nguyen, V., Shokr, M., Novak, A., and Caelli, T. 2016. A Reconfigurable Agent-Based Discrete Event Simulator for Helicopter Aircrew Training, *International Symposium on Military Operations Research* (ISMOR).

Novak, A., Tracey, L., Nguyen, V., Johnstone, M., Le, V. T., and Creighton, D. C. 2015. Evaluation of Tender Solutions for Aviation Training Using Discrete Event Simulation and Best Performance Criteria, *Winter Simulation Conference.*

Nowicki, E., and Smutnicki, C. 1996. A Fast Taboo Search Algorithm for the Job Shop Problem, *Management Science*, Vol 42, No 6, 797–813.

Oracle. 2018. Java SE at a Glance. http://www.oracle.com/technetwork/java/javase/overview/index.html.

Pillay, N. 2016. A Review of Hyper-Heuristics for Educational Timetabling, *Annals of Operations Research*, Vol 239, No 1, 3–38.

Qi, X., Bard, J. F., and Yu, G. 2004. Class Scheduling for Pilot Training, *Operations Research*, Vol 52, No 1, 148–162.

Raffensperger, J. F., and Schrage, L. E. 2008. Scheduling Training for a Tank Battalion: How to Measure Readiness, *Computers & Operations Research*, Vol 35, No 6, 1844–1864.

Rudová, H., Müller, T., and Murray, K. 2011. Complex University Course Timetabling, *Journal of Scheduling*, Vol 14, No 2, 187–207.

Schaerf, A. 1999. A Survey of Automated Timetabling, *Artificial Intelligence Review*, Vol 13, No 2, 87–127.

Scott, J. 2005. Optimally Scheduling Basic Courses at the Defense Language Institute Using Integer Programming. Master's thesis, Naval Postgraduate School, Monterey, CA.

Séguin, R., and Hunter, C. 2013. 2 Canadian Forces Flying Training School (2CFFTS) Resource Allocation Simulation Tool, *Winter Simulations Conference* (WSC), 2866–2877.

Squires, R. R., and Hoffman, K. L. 2015. A Military Maintenance Planning and Scheduling Problem, *Optimization Letters*, Vol 9, No 8, 1675–1688.

Tarjan, R. 1972. Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing*, Vol 1, No 2, 146–160.

Teoh, C. K., Wibowo, A., and Ngadiman, M. S. 2015. Review of State of the Art for Metaheuristic Techniques in Academic Scheduling Problems, *Artificial Intelligence Review*, Vol 44, No 1, 1–21.

Teter, M. D., Newman, A. M., and Weiss, M. 2016. Consistent Notation for Presenting Complex Optimization Models in Technical Writing, *Surveys in Operations Research and Management Science*, Vol 21, No 1, 1–17.

Wang, J., Feb 2005. A review of operations research applications in workforce planning and potential modeling of military training. Tech. rep., DSTO, Salisbury, South Australia.

Wang, S., Bussieck, M., Guignard, M., Meeraus, A., and O'Brien, F. 2010. Term-End Exam Scheduling at United States Military Academy/West Point, *Journal of Scheduling*, Vol 13, No 4, 375–391.

Wren, A. 1996. *Scheduling, Timetabling and Rostering—A Special Relationship?* Springer, 46–75.

Yang, T., and Ignizio, J. P. 1987. An Algorithm for the Scheduling of Army Battalion Training Exercises, *Computers & Operations Research*, Vol 14, No 6, 479–491.