Computers & Industrial Engineering 127 (2019) 263–273

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Exam timetabling with allowable conflicts within a time window

Omar Abou Kasm[a], Baraa Mohandes[b], Ali Diabat[a,c,*], Sameh El Khatib[b]

[a] Department of Civil and Urban Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, United States
[b] Department of Engineering Systems and Management, Masdar Institute of Science and Technology, Abu Dhabi, United Arab Emirates
[c] Division of Engineering, New York University Abu Dhabi, Saadiyat Island, 129188 Abu Dhabi, United Arab Emirates

ARTICLE INFO

Keywords:
Exam timetabling
NP-complete problem
Integer programming
Largest box algorithm
Graph theory
Case studies

ABSTRACT

The exam timetabling problem is considered an NP-complete problem and its complexity depends on the different constraints and policies set by an institution's administration. The goal of this work is to facilitate exam timetabling for Masdar Institute (MI), which is a graduate level institution. Besides the renowned constraint on conflicts for students, MI's timetabling case includes the incorporation of venues' limited capacities, special instructor requests, and the number of exams scheduled for one student within a preset window of days. To the best knowledge of the authors, the latter constraint is new to the literature. Moreover, it increases the problem's complexity since it requires cross-validation on both student and course levels. This contrasts with conventional exam timetabling which only deals with checks on a course level. We introduce an integer programming (IP) formulation that captures all the studied constraints. The proposed formulation can solve small problems using commercial software; however, this formulation's performance deteriorates as the problem size increases. Therefore, the paper proposes heuristics to solve medium and large sized problems in a timely manner. This study employs graph coloring algorithms that include a new approach, within the steps of the proposed exam timetabling heuristics. Four real-case studies from MI are solved to illustrate the feasibility and competitiveness of the proposed heuristic. Finally, a computational study is presented to benchmark the proposed heuristics against the IP formulation. The results show that the proposed heuristics are capable of obtaining optimal and near-optimal solutions in smaller computational time.

1. Introduction

Resource scheduling problems have numerous forms and variations in real life. Such problems arise during allocation of scarce resources such as human power, during process planning in production lines, as well as exam and lecture scheduling in educational facilities. The core challenge in the scheduling problem is preventing concurrent assignment of limited resources. For example, a company running several projects in parallel has only a few employees who are qualified to execute a critical job. At the same time, such a job is required at different phases of each project. Another example is machines that carry out special tasks which are required in multiple process lines.

One application for the scheduling problem is exam timetabling (Malkawi, Hassan, & Hassan, 2008). The problem's complexity stems from resource limitations in terms of exam venues and time slots. Furthermore, educational institutions apply additional rules such as limiting the number of exams in one day for each student. Small institutions tackle the problem manually with trial and error. Various techniques for mitigating the problem are studied in the literature. The existing algorithms are based on heuristics, optimization, or graph theory analysis. Graph theory and heuristics tend to converge faster than the iterative optimization methods, but the optimality of the solution is not guaranteed.

Graph coloring is a renowned problem in graph theory. Some aspects in the exam timetabling problem are similar to the graph coloring problem. Essentially, if courses are used to represent vertices, the graph coloring problem requires grouping vertices together, such that adjacent vertices are not assigned to the same color group. Two adjacent vertices represent two courses that have at least one student in common, and the two courses may not be scheduled for the same time slot. The general approach of existing literature on the problem is minimizing the number of groups.

This paper aims to solve the exam timetabling problem, and it incorporates real considerations adopted from a graduate institution, Masdar Institute (MI), and it generalizes the proposed approaches. The institution's policy aligns with the general requirement of not scheduling two consecutive or concurrent exams for one student. Moreover, a student may not have more than two exams in a two-day window.

* Corresponding author at: Division of Engineering, New York University Abu Dhabi, Saadiyat Island, 129188 Abu Dhabi, United Arab Emirates.
E-mail address: diabat@nyu.edu (A. Diabat).

https://doi.org/10.1016/j.cie.2018.11.037
Received 20 March 2018; Received in revised form 20 August 2018; Accepted 19 November 2018
Available online 24 November 2018
0360-8352/

Available work in the literature usually restricts the number of exams per a single day only (Akbulut & Yilmaz, 2013; Hassan & Hassan, 2016; Mohamed, Mushi, & Mujuni, 2013). The problem's complexity is significantly increased by restricting the number of exams to two within a window of two days. Variations of this rule are also addressed; namely, 1 exam in 2 days, 2 exams in 1 day, and 1 exam in 1 day. The institution also accounts for special instructor requests, such as allocating a specific exam on a particular day. Finally, exam venue capacities are also considered. In the cases tackled by this paper, available exam venues vary in capacity from only 5 students to over 25 students. Hence, considering these capacities is essential.

We first propose an integer program (IP) to solve the proposed problem. However, the problem is tractable in commercial software with small problems only. Thus, we employ heuristics to solve medium and large sized problems. Heuristics incorporate graph coloring algorithms within their steps. In more detail, two graph coloring algorithms are used as subroutines within the larger exam timetabling heuristic. The first finds the minimum number of groups for a given graph coloring problem, while the second is designed and introduced in this work to find the largest possible group. The latter serves the purpose of providing more flexibility when allocating exams in classrooms while satisfying capacity constraints. Nevertheless, we then illustrate the use of the proposed exam timetabling heuristics on four real case studies acquired from MI. Finally, we compare the performance of the proposed heuristics against the IP formulation using the MI case studies as well as renowned benchmark cases.

This paper is organized as follows. Section 2 reviews the literature on the subject. Section 3 lists the problem's considerations and introduces the IP formulation. Section 4 introduces the color graphing and exam timetabling heuristics. Section 5 discusses four case studies adapted from MI. Section 6 provides a computational analysis and, finally, the conclusions of this paper are presented in Section 7.

2. Literature review

A simple graph defined as $\mathcal{G}$: (V, E) consists of n vertices and m edges. The graph can be represented by two sets: the set of vertices $V(G) = \{v_1, v_2, v_3, v_n\}$ and the set of edges $E(G) = \{e_1, e_2, e_3, \ldots, e_m\}$. Each edge $e(k)$ uniquely connects an unordered pair of vertices v_i and v_j such that $(v_i, v_j) = e_k \in E(G)$. $A(G)$ is the adjacency matrix of $\mathcal{G}$. $A(G)$ is an $n \times n$ symmetric binary matrix where $A_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E(G)$; and $A_{ij} = 0$, otherwise. The main constraint in the graph-coloring problem is: $x, y \in E(G) \Rightarrow C(x) \neq C(y)$. This mathematical expression simply says that if the vertices x and y are connected by an edge, then the color of x cannot be the same as the color of y. The minimum number of colors required to group the vertices for a graph, while satisfying the constraint above, is integer denoted as the chromatic number: $\chi(G)$. Finding the value for $\chi(G)$ is called a Graph Coloring Problem (GCP). The GCP is considered a NP-complete problem (Garey, Johnson, & Stockmeyer, 1974).

Nguyen and Bui and University of Nottingham provide real sized instances that can be useful for benchmark testing of new algorithms. In fact, we employ these tests in this paper's computational study. Several techniques from different research fields have been used to tackle this problem. Most of these algorithms revolve around genetic algorithms and heuristic combinations of genetic algorithms (Fleurent & Ferland, 1996; Hertz & de Werra, 1987; Marappan & Sethumadhavan, 2013; Mumford, 2006; Han & Han, 2010) and others move towards obtaining exact or near optimal solutions such as column generation or branch and cut (Mehrotra & Trick, 1996; Méndez-Díaz & Zabala, 2006). Algorithms employed to solve the exam timetabling problem can be classified into four types. Those types include graph based sequential techniques, constraint based techniques, local search based techniques, and, finally, population based algorithms (Qu, Burke, McCollum, Merlot, & Lee, 2009). The most recent methods that target the resource scheduling problem are mainly hybridizations of methods from the different categories. Sabar, Ayob, Qu, and Kendall (2012) discussed hyper-heuristic algorithms that consist of two levels. The first, upper level, heuristic selects the most suitable solution algorithm. This choice is based on the problem constraints and size. The second, lower level, heuristic solves the actual graph coloring problem.

Techniques based on graph theory represent the system as a graph and are mainly heuristic approaches. Sabar et al. (2012) classify graph coloring algorithms into two types: constructive algorithms and improvement-based algorithms. Constructive algorithms build the solution step by step. On the other hand, improvement-based algorithms start with a low-quality feasible solution, and carry out modifications on the solution to improve the objective, iteratively. The earliest graph theory based heuristics are constructive algorithms. Exams are scheduled one by one, based on a difficulty rating. Different difficulty indices were proposed in the literature. Sabar et al. (2012) developed a hybrid difficulty index that combines several indices together. Since exams are scheduled one by one, constructive techniques are sometimes denoted as sequential techniques.

The Recursive Largest First (RLF) algorithm is among the most common algorithms used for exam timetabling (Leighton, 1979). This algorithm schedules exams based on their centrality degree, i.e. the number of edges connected to the vertex (Newman, 2010). Thus, if represented in an adjacency matrix, the vertex degree is the sum of elements in its respective column. In the exam timetabling problem, the centrality of a particular course would be the number of other courses with which it conflicts, regardless of the number of students causing the conflict. RLF assigns the courses with the highest centrality first. Such courses are usually the most difficult to schedule. When these courses are removed from the graph, a relatively sparser sub-graph remains which is easier to schedule. Moreover, Mohamed et al. (2013) explain that a course with many conflicts usually contains the largest number of students, so scheduling it first allows the instructors more time to grade their exams. When a course is scheduled, a set of possible non-conflicting vertices are added to its group. From the set of available (non-conflicting) options, courses with the highest centrality are added to the color group. The set of non-conflicting vertices needs to be modified to consider the conflicts of the last scheduled course. The process is repeated until no more vertices are available to schedule in the given group. This procedure continues until no more exams are left for scheduling.

Constructive algorithms, or sequential assignment of exams, can result in situations where some courses are left unscheduled, and no more time slots are left. For this reason, new graph based techniques are developed. Courses that were already scheduled, but happen to have numerous conflicts, are revoked, leaving space for courses that were left unscheduled. In essence, this helps climbing out of an infeasible solution. These techniques are also called back-tracking techniques. There is not a particular graph based technique that has systematic and consistent advantage over other techniques (Qu et al., 2009). This aligns with the *No-free-lunch* theorem in optimization. In contrast with graph theory based techniques, constraint based techniques handle the problem as a mathematical program consisting of an objective function and constraints. One measure of the goodness of a solution is the spread of exams over the exam period for each particular student. An evaluation criterion was developed by Laporte and Desroches (1984) particularly for this purpose. Constraint based techniques are usually computationally intensive. Large problems require substantial time to solve. Hence, constraint based algorithms are usually combined with heuristic methods to obtain solution algorithms with better time complexity (Qu et al., 2009). Mathematical formulations of the exam time tabling problem are generally Integer Programming (IP) formulations and thus IP solution approaches can be used. Hence, Woumans, De Boeck, Beliën, and Creemers (2016) proposed a column generation approach for solving the exam timetabling problem.

Improvement based algorithms are also known in the literature as local search algorithms. These algorithms start from an arbitrary

feasible solution and search for better solutions in the vicinity of the current one. Candidate solutions are generated by making modifications on the current solution. To compare between different feasible solutions, a cost function is evaluated for each candidate solution. If a candidate solution has the same cost function, or better, this particular candidate solution replaces the current solution. New solutions are also sought from that point onward through random modifications. Different local search algorithms employ different techniques to improve the algorithm's performance. This is the core difference among local search algorithms (Bykov & Petrovic, 2016). Some examples of local search algorithms are the Tabu Search algorithm, Simulated Annealing, and the latent decision algorithm. Roulette-wheel selection was proposed by Sabar, Ayob, Kendall, and Qu (2009) as an application of introducing randomness to the solution algorithm.

Considerations and constraints dictated by the educational institutions' policies create various versions of the exam timetabling problem. Akbulut and Yilmaz (2013) involved exam-venue as another factor in the scheduling problem and added venue capacity as a constraint. The authors also suggested scheduling two exams in the same hall at the same time to utilize the full capacity of a venue. Other work by Rachmawati, Armay, and Purnomo (2012) requires the lecturer of a course to be present in the exam venue, which means that two exams of separate courses by the same lecturer cannot be scheduled concurrently. However, this constraint cannot be met in universities where the same instructor teaches courses with their sections segregated by gender. Scheduling two such sections at different time slots compromises the confidentiality of the exam questions. Another example of real case studies is demonstrated by Leighton (1979), where certain exams may not be scheduled at specific time periods. To overcome this constraint, the author created a dummy vertex scheduled at the concerned time slot, and connected the dummy vertex to the courses that are not allowed to be scheduled at the concerned slot. We account for this consideration in our proposed algorithm.

Malkawi et al. (2008) approached the exam timetabling problem also by using graph theory and discussed an algorithm that accounts for a custom set of constraints. They discuss the idea of user identified exam slots and exam rooms; however, they do not consider the exam room capacities. Moreover, they allow each student to have up to "y" exams per day where the user can identify this value. It is a fair assumption to limit the number of exams per day for a single student to 1 exam. If this condition is not sensible, it can be relaxed to two exams per day. Exceeding two exams per day for a student is considered unfair. This idea is incorporated in our work. We also introduce a two-day window policy for the exams, which significantly increases the complexity of the problem. In the next section, all conditions and constraints proposed by our work are presented.

3. Problem description and mathematical formulation

Given a list of courses and student enrollments, it is required to schedule the respective exams within the least amount of days. A number of scheduling policies and constraints must also be respected. All such considerations listed below are followed by an Integer Program (IP) formulation that captures the problem's objective and all constraints.

1. user identified exam days
2. user identified exam slots per day
3. no back-to-back exams for each student
4. four scheduling policies that the user can choose from
   a. no more than two exams in two days for each student
   b. no more than one exam per day for each student
   c. no more than two exams per day for each student
   d. no more than one exam in two days for each student
5. special requests by faculty
   a. specific time slot
   b. specific day
   c. not on a specific day
   d. exam duration exceeds the typical duration of one slot
6. user identified exam rooms and capacities
7. two exam room policies that the user can choose from
   a. only one exam per exam room
   b. more than one exam can be assigned in the same exam room as long as the capacity is not exceeded
8. courses with more than one section should be scheduled in the same time slot

The notation is defined as follows:

Sets

$\mathcal{I}$ set of exams to schedule, $i', i \in \{1, \ldots, I\}$
$\mathcal{J}$ set of days, $j \in \{1, \ldots, J\}$
$\mathcal{K}$ set of slots in a day, $k \in \{1, \ldots, K\}$
$\mathcal{E}$ set of exam rooms, $e \in \{1, \ldots, E\}$
$\mathcal{L}$ set of students, $l \in \{1, \ldots, L\}$

Parameters

$$A'_{ii} = \begin{cases} 1 & \text{if conflict exists between exams } i \text{ and } i'. \ \forall i, i' \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{li} = \begin{cases} 1 & \text{if student } l \text{ is enrolled in exam } i. \ \forall l \in \mathcal{L}, i \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$$

N_i number of students enrolled in exam i. $\forall i \in \mathcal{I}$
C_e capacity of exam room e. $\forall e \in \mathcal{E}$

Decision variables

$$x_{ijke} = \begin{cases} 1 & \text{if exam } i \text{ is scheduled on day } j, \text{ slot } k, \text{ venue } e. \ \forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}, e \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{lijk} = \begin{cases} 1 & \text{if student } l \text{ has exam } i \text{ on day } j, \text{ slot } k. \ \forall l \in \mathcal{L}, i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K} \\ 0 & \text{otherwise} \end{cases}$$

Using this notation, the model can be stated as follows:

$$\min T \tag{1}$$

$$s.\ t.$$

$$T - \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} \sum_{e \in \mathcal{E}} j.\ x_{ijke} \geqslant 0 \quad \forall i \in \mathcal{I} \tag{2}$$

$$\sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} \sum_{e \in \mathcal{E}} x_{ijke} = 1 \quad \forall i \in \mathcal{I} \tag{3}$$

$$A_{ii'}. \sum_{e \in \mathcal{E}} (x_{ijke} + x_{i'jke}) \leqslant 1 \quad \forall i, i' \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}, i \neq i' \tag{4}$$

$$A_{ii'}. \sum_{e \in \mathcal{E}} (x_{ijke} + x_{i'jk+1e}) \leqslant 1 \quad \forall i, i' \in \mathcal{I}, j \in \mathcal{J}, k < K, i \neq i' \tag{5}$$

$$y_{lijk} = B_{li}. \sum_{e \in \mathcal{E}} x_{ijke} \quad \forall l \in \mathcal{L}, i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K} \tag{6}$$

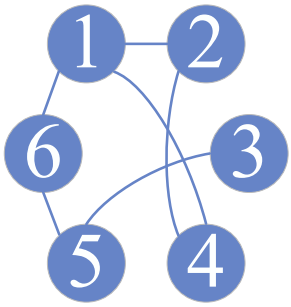
$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} (y_{lijk} + y_{lij+1k}) \leqslant 2 \quad \forall l \in \mathcal{L}, j < J \tag{7}$$

$$\sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} N_i.\ x_{ijke} \leqslant C_e \quad \forall i \in \mathcal{I}, e \in \mathcal{E} \tag{8}$$

$$\sum_{i \in \mathcal{I}} x_{ijke} \leqslant 1 \quad \forall j \in \mathcal{J}, k \in \mathcal{K}, e \in \mathcal{E} \tag{9}$$

$$x_{ijke} \in \{0, 1\} \quad \forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}, e \in \mathcal{E} \tag{10}$$

$$y_{lijk} \in \{0, 1\} \quad \forall l \in \mathcal{L}, i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K} \tag{11}$$



(a) Graph Coloring Network Example

	1	2	3	4	5	6
1		1		1		1
2	1			1		
3					1	
4	1	1				
5			1			1
6	1				1	

(b) Adjacency Matrix Example

Fig. 1. Example of a network.

The objective function, in (1), minimizes the exam's period T which is restricted by constraint (2). The constraint, with the support of the objective function, is the result of the linearization for Eq. (12) which sets T equal to the last day in the schedule.

$$T = \max_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} \sum_{e \in \mathcal{E}} j.\, x_{ijke} \tag{12}$$

Constraint (3) ensures that every exam is scheduled. Constraints (4) and (5) ensure that there are no conflicts in the same time slot and no back-to-back exams for each student, respectively. Constraint (6) links students to their respective exams. Specifically, given that student l is enrolled in exam i, the constraint ensures that respective student variable y_{lijk} takes the value of 1 if exam i is scheduled in day j and slot k ($\forall\, l \in \mathcal{L}, i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}, B_{li} = 1$). Constraint (7) constrains the number of exams scheduled in two days for each student. Constraint (8) enforces the capacity limit of each exam venue. Constraint (9) prevents scheduling multiple concurrent exams in the same venue. Constraints (10) and (11) declare the decision variables as binary.

If scheduling multiple exams in the same venue is permitted by the institution's policies, then constraint (13) replaces constraints (8) and (9).

$$\sum_{i \in \mathcal{I}} N_i.\, x_{ijke} \leqslant C_e \qquad \forall\, j \in \mathcal{J}, k \in \mathcal{K}, e \in \mathcal{E} \tag{13}$$

The formulation represents the policy in 4.a. In order to account for policy 4.b, constraints Eqs. (4)–(7) must be replaced by constraint (14) which ensures no conflicts on each day. Incorporating policies 4.c and 4.d requires replacing constraint (7) by constraints (15) and (16), respectively. Finally, all special requests by faculty (policy 5) and courses with multiple sections (policy 8) can be easily accounted for by adding simple specific constraints to meet the required condition. For example, if exam#3 is required to be scheduled on (day#4, slot#2), then the following constraint: $\sum_{e \in \mathcal{E}} x_{3,4,2,e} = 1$ can be added. The constraint ensures that the exam is scheduled in the requested slot and time.

$$A_{ii'}.\sum_{e \in \mathcal{E}} (x_{ijke} + x_{i'jke}) \leqslant 1 \quad \forall\, i, i \in \mathcal{I}, j \in \mathcal{J}, i \neq i' \tag{14}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} y_{lijk} \leqslant 2 \quad \forall\, l \in \mathcal{L}, j \in \mathcal{J} \tag{15}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} (y_{lijk} + y_{lij+1k}) \leqslant 1 \quad \forall\, l \in \mathcal{L}, j < J \tag{16}$$

4. Graph coloring and exam timetabling heuristics

The IP formulation's performance deteriorates with the increase of the problem's size as demonstrated in the computational study (Section 6). Henceforth, we proceed to design exam timetabling heuristics which are time efficient and capable of giving near-optimal results. Graph coloring algorithms are the underlying technique of the proposed heuristics. The graph-coloring problem is concerned only with one constraint, which is analogous to the no-conflict constraint in the exam timetabling problem. The bulk of the literature on graph-coloring algorithms minimizes the number of color groups. In this paper, we introduce the Largest Box (LB) algorithm, which obtains the largest group of non-conflicting vertices. This algorithm is then used as part of a larger algorithm that captures realistic general constraints in exam timetabling other than the basic non-conflict constraint.

The reason for the proposed modification in the algorithm design is to have flexibility in scheduling the exams with limited venues and limited venue capacity. When an approach similar to the RLF is used, the courses which are hardest to schedule (vertices with the highest centrality) are scheduled first. As a result, the first color groups have fewer exams which constrains the problem of allocating venues to the scheduled courses considering room capacities. The proposed algorithm behaves differently, where the first color group has the most courses (vertices). This grants some flexibility in selecting the courses based on room capacities. Moreover, the algorithm can be set to ensure that exams with high conflicts are among the courses scheduled in the beginning of the exam's period. Such courses usually contain the largest number of students, as hypothesized by Mohamed et al. (2013), and are better scheduled in the beginning of the exams period to provide instructors with ample time for grading papers. These courses may be divided across different slots in one day rather than scheduling as many as possible in the same single slot. In the next section, the LB algorithm is introduced and then incorporated as part of the main exam timetabling heuristics discussed afterwards.

4.1. Largest Box Algorithm: a new graph coloring approach

A given network, such as Fig. 1, can be represented by the adjacency matrix shown in Fig. 1. We can then identify n vertices which can be grouped together by finding a single body of $n \times n$ empty boxes around the diagonal. For example, the matrix entries a_{23} and a_{32} satisfy this criterion. The boxes corresponding to these entries are highlighted in Fig. 2. Similarly, the entries a_{45} and a_{54} form an empty box, which indicates that

	1	2	3	4	5	6
1		1		1		1
2	1			1		
3					1	
4	1	1				
5			1			1
6	1				1	

Fig. 2. Example of a box for grouping vertices.

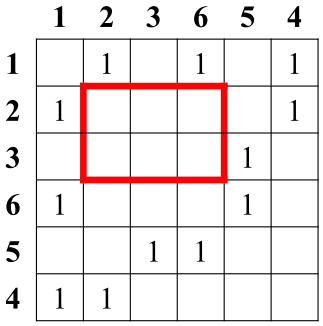


Fig. 3. Obtaining a larger box.

vertices 4 and 5 can be scheduled together. Changing the order of vertices in the matrix can generate larger empty boxes and a larger color group. For example, swapping vertices 4 and 6 in the matrix yields the matrix illustrated in Fig. 3. It is possible then to mark a larger box of dimensions 3×3 as compared to the previous 2×2 box. Hence, vertices 2, 3, and 6 can be grouped together. Note that a larger box is achieved by placing more vertices that can be grouped together consecutively.

The algorithm aims to find the largest box possible through re-ordering columns and rows and goes as follows:

Definitions:
A: adjacency matrix
N: ordered set showing vertex arrangement in *A*
C: array showing the location of the first non-zero element of each row in *A*
G_i: array to store vertices that can be scheduled together
R: number of rows in *A*

1. r = 1, i = 1
2. while $r \leqslant R$
   a. If $C(r) = \max(C(r), C(r+1), \ldots, C(n)) \Rightarrow r = r + 1$
   b. Else, find $max(C(r), C(r+1), \ldots, C(n))$ and swap its respective row and column with row r and column r in A, swap the respective elements in N, update *C*
3. Find the largest square (symmetrical along the diagonal) having an element sum = zero where its first element is the first element in *A*
4. Assign all respective vertices to G_i, $i = i + 1$
5. Delete the respective vertices from *A* and *N*
6. Repeat from step 2 until there are no more vertices to schedule.

For example, consider the adjacency matrix *A* given in Fig. 1. Let vector *N* represent the vertices in their order of appearance, and thus $N = [1, 2, 3, 4, 5, 6]$. We set $r = 1$ (i.e., to study the first row) and $i = 1$ (i.e., to fill the first group). In the beginning, we need to identify the vertex that has the farthest non-zero element in its row. Vector $C = [2, 1, 5, 1, 3, 1]$ identifies the location of the first non-zero element for each vertex in their appearance in *N*. Vertex 3 now has the maximum value and, therefore, we swap it with the vertex in location *r* (vertex 1 in this case). To do that, we must swap rows 1 and 3, and swap columns 1 and 3. The obtained matrix is shown in the top left of Fig. 4. Vector *N* which represents the order of vertices is now $N = [3, 2, 1, 4, 5, 6]$.

Now we find vector *C* for the new arrangement: $C = [5, 3, 2, 2, 1, 3]$. Since the largest element is in the position of $r = 1$, we set $r = r + 1 = 2$ to study the second row. The vertex of this row also has the maximum value. Note that inspection of rows starts from the row of *r*. Thus the value 5 of the first vertex is discarded. The vertex with the farthest non-zero element is vertex 6, and thus we swap it with the vertex in position $r = 3$ (i.e., vertex 1). The matrix after the swap is shown in the top right of Fig. 4. Vector *N* describing the order of vertices is now $N = [3, 2, 6, 4, 5, 1]$.

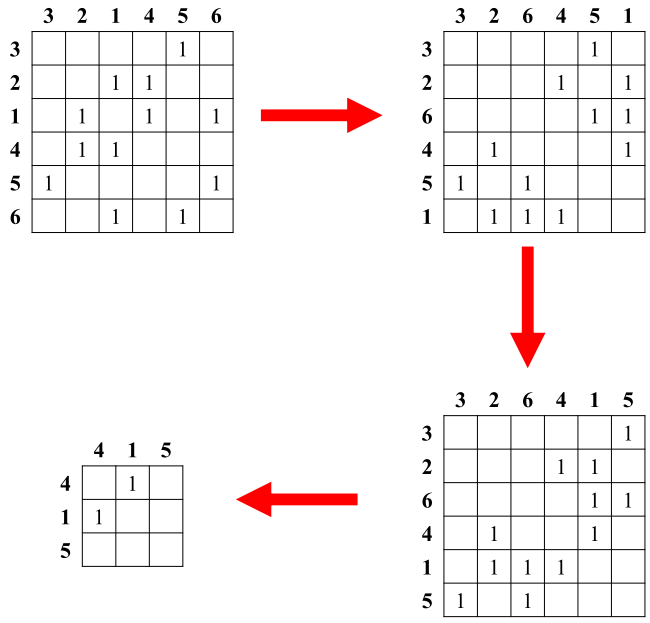


Fig. 4. Example steps.

Vector *C* for the new arrangement is $C = [5, 4, 5, 2, 1, 2]$. The vertex in the position $r = 3$ has the highest value, so we proceed to carry out the same row and column operations for $r = 4$ and $r = 5$, respectively. Thereupon, it appears that vertex 1 can be swapped with vertex 5, to obtain the matrix shown in the bottom right of Fig. 4. Vector *N* now is $N = [3, 2, 6, 4, 1, 5]$. At the same time, vector *C* is $[6, 4, 5, 2, 2, 1]$ indicating that the current value of $r = 6$ is equal to the number of vertices of the graph, and no more swaps are required. Marking out the largest box is possible now, where the box includes the first entry in the matrix (vertex 3 in this example). The largest empty box has dimensions 3×3, and it contains the vertices $\{3, 2, 6\}$. These three vertices can be grouped together into one color group $G_1 = \{3, 2, 6\}$.

When the vertices which compose the largest box have been allocated to a color group, they can be removed from the adjacency matrix to start the process of finding the next largest empty box. The result is the matrix shown in the bottom left of Fig. 4, with vector $N = [4, 1, 5]$. We set $i = 2$, $r = 1$, and repeat the same procedure until all vertices are grouped. We obtain $G_2 = \{5, 1\}$ and $G_3 = \{4\}$.

The proposed algorithm terminates when every vertex has been assigned to a color group. However, as will be discussed in the next section, only the first (and largest) color group is required to initialize the exam timetabling algorithm. When a vertex is required to belong to a certain color group, the vertex is placed as the first vertex in the matrix. The LB algorithm starts from the subject vertex. For example, assume that vertex 4 from the previous problem is to be included in the largest color group. We place vertex 4 as the first vertex; this is depicted

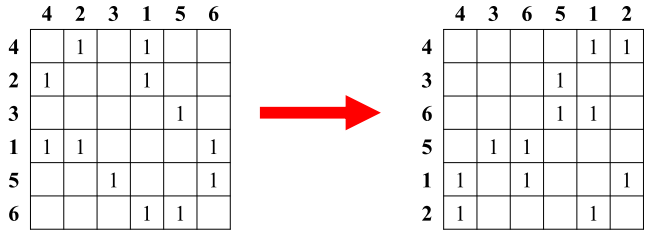


Fig. 5. Matrix with ensuring the inclusion of vertex 4.

as the left matrix in Fig. 5. The end result after all necessary vertices swaps is shown as the right matrix in Fig. 5, in which the largest box includes vertices: $G = \{4, 3, 6\}$. We can follow the same approach to assure that a small group of non-conflicting vertices are scheduled together. Finally, it is important to note that swapping with the vertex with lowest degree centrality facilitates finding larger boxes. This is because the vertex with lowest degree centrality has the least conflicts and makes it easier to include more courses in the group.

4.2. Exam timetabling heuristics

In this section, we introduce exam timetabling heuristics that can account for all the considerations in Section 3. The notation used in pseudo-codes are defined as follows:

Parameters:

D	number of days
S	number of slots per day
N	number of exam rooms
ND	number of high degree nodes

Sets:

S_{ij}	courses scheduled on day i at time slot j
C	full list of courses
T_1	courses forbidden temporarily from being scheduled
T_2	courses forbidden temporarily from being scheduled
A	successfully scheduled courses
B	remaining courses to be scheduled (not scheduled yet)
G	courses considered temporarily for scheduling

Algorithm A accounts for considerations 1, 2, 3, 4.a, 6, and 7. The logic behind the algorithm is to start with day 1, and attempt to fill slot 1 with as many courses as possible before moving to slot 2. To avoid assigning two consecutive exams for one student, the courses in slot 2 must not conflict with the courses in slot 1. The same applies afterwards, where the courses allocated to slot 3 should not conflict with both slots 1 and 2. If this is not possible, then we schedule exams that only do not conflict with slot 2. If the latter case is applicable, then there is a conflict between slots 1 and 3, and some students will take two exams on this day. In this case, it is important to keep record of these students and their other courses in a temporary list of courses. Allocating these courses on any slot of the next day's schedule will be prohibited. When the exam schedule of the next day is complete, these temporarily forbidden courses are considered for the following days. This ensures that the students will have a maximum of two exams in a window of two days. In general, consider that the algorithm is currently handling slot j, with the goal of scheduling exams that do not conflict with slots 1 to $j-1$. If this goal is not attainable, the algorithm attempts to schedule courses which do not conflict with slots 2 to $j-1$, and so on.

Algorithm A. Main pseudo-code

Init: $B = C; i = 1; j = 0;$ (i.e., S_{10}); $r \longleftarrow 2; T_1 \& T_2 = \emptyset$
$j \longleftarrow j+1$: Schedule up to N courses[1] from B and move them from B to A
  if $A = C$ **then End**
  else Go to: Line 4
$j \longleftarrow j+1$: Schedule up to N courses not conflicting with $S_{i,j-1}$ from B
  if *Possible* **then**
    Move Scheduled courses from B to A
    if $A = C$ **then End**
    if j=S **then** $i \longleftarrow i+1$, $j = 0$, Move courses from T_2 to B, **Go to:** Line 1
    else Go to: Line 13
  else
    if j=S **then** Skip (this slot will be empty), move courses from T_2 to B, $i \longleftarrow i+1$, **Go to:** Line 1
    else Skip and **Go to:** Line 20
  end
$j \longleftarrow j+1$: Schedule up to N courses non-conflicting with S_{ik} for $k = 1 : j-1$ from B
  if *Possible* **then**
    Move Scheduled courses from B to A
    if $A = C$ **then End**
    if $j = S$ **then** $i = i+1, j = 0$, move courses from T_2 to B, **Go to:** Line 1
    else Go to: Line 13
  else
    Move courses that contain students having exams on both $S_{i-1,j}$ and S_{ik} for $k = 1 : j-2$ from B to T_2, Schedule up to N courses non-conflicting with S_{ik} for $k = r : j-1$ from B
      if *Possible* **then** Move contents of B to A
        if $j = S$ **then** $i = i+1, j = 0$, Move conflicting courses in S_{ij} & S_{ik} for $k = 1 : r-1$ from B to T_1, Move contents of T_2 to B, Move contents of T_1 to T_2, **Go to:** Line 1
        else Move conflicting courses in S_{ij} & S_{ik} for $k = 1 : r-1$ from B to T_1, $j \longleftarrow j+1$, **Go to:** Line 20
      else if $r = j-1$
        if $j = S$ **then** $r = 2, i = i+1, j = 0$, Move contents of T_2 to B, **Go to:** Line 1
        else $j \longleftarrow j+1$, Schedule up to Ncourses from B, Move Scheduled Courses from B to A
          if $A = C$ **then** End
          if $j = S$ **then** $r = 2, i = i+1, j = 0$, Move conflicting courses in S_{ij} & S_{ik} for $k = 1 : r$ from B to T_1, move contents of T_2 to B, move contents of T_1 to T_2, **Go to:** Line 1
          else $r \longleftarrow r+2$, **Go to:** Line 20
      else $r \longleftarrow r+1$, **Go to:** Line 20
  end

[1]Use Sub-algorithm (i)

The discussion until now covers the concept of a forbidden list of courses for the following day. The same concept needs to be applied for the preceding days. For example, if two exams were scheduled for one student on the current day, it is essential to ensure this particular student has no courses scheduled on the previous day. The algorithm identifies the students who have exams scheduled on the previous day when scheduling two conflicting courses in one day. The algorithm then identifies students who have one exam scheduled so far on the current day, and moves their remaining courses to a temporary list of forbidden courses. This list of forbidden courses is open for scheduling again after allocating exams to the current day.

The LB algorithm is used as the main color graphing technique as it gives larger groups. Hence, the LB algorithm extends more flexibility to schedule the courses while respecting the exam room size constraints. The RLF is used as a set up for the LB algorithm in order to include a number of high centrality courses in the slot we aim to fill. For scheduling courses in a slot, the user can choose between conditions 7.a or 7.b depending on the required policy. Their steps are shown in Sub-Algorithm (i).

Sub-Algorithm (i): Steps to Schedule *N* Courses

Condition 7.a: one exam per exam room
1 Run the RLF algorithm to schedule *ND* courses
2 List the *ND* courses as the first nodes in the matrix representing set *B* and run LB with $r = ND + 1$, save result in set *G*
3 Assign one exam to each exam room from *G*: for each room choose the exam with the highest number of students that respects the class capacity. In the case of ties, choose the course with higher degree. Return excess to set *B*.

Condition 7.b: more than one exam per exam room
1 Run the RLF algorithm to schedule *ND* courses
2 List the *ND* courses as the first nodes in the matrix representing set *B* and run LB algorithm with $r = ND + 1$, save result in set *G*
3 Schedule exams from *G*: Choose the class with least capacity, add the exam with the highest number of students that respects the class capacity (In the case of ties, choose the course with higher degree). Repeat until class is full or no more exams can fit.
4 Repeat Line 3 for another class with higher or equal capacity until all exam rooms are occupied. Return excess to set *B*.

To account for condition 4.b (one exam per day) instead of 4.a (two exams in a window of two days), Algorithm B must be used. The latter algorithm uses the RLF for graph coloring since it returns a solution with fewer color groups. This outcome is actually preferable when only one exam per day may be scheduled for any student. Each color group is distributed over a separate day, and the remaining groups, if any, are returned to the matrix to be scheduled in later days. If condition 4.c (no more than two exams per day for each student) is to be enforced instead of 4.a or 4.b, then Algorithm A can be used while discarding the use of temporary sets T_1 and T_2. That is, discard any move that requires moving courses to temporary sets. This allows for allocating two exams per day without a limit on the number of exams within a window of days. Finally, to account for condition 4.d (no more than one exam in two days for each student), Algorithm C can be used. The difference between Algorithms C and B is that whenever a color group is chosen, it is scheduled over two days rather than one day.

Algorithm B. Capturing condition 4.b

Condition 4.b: no more than one exam per day for each student
Init: $B = C, i = 1, j = 1, A = \emptyset, G = \emptyset$
1 Run RLF on B and select the first group, and move selected exams from B to G
2 Schedule up to Ncourses from G in S_{ij} (Line 3 from Sub-Algorithm (i))
3 **if** $A = C$ **then** End
4 **if** $j = S$ **then**
5 Move scheduled courses from G to A
Return remaining courses in G to B
$i \longleftarrow i + 1, j = 1$, **Go to:** Line 1
6 **else**
7 Move scheduled courses from G to A
$j \longleftarrow j + 1$, **Go to:** Line 2
8 **end**

Algorithm C. Capturing condition 4.d

Condition 4.d: no more than one exam in two days for each student
Init: $B = C, i = 1, j = 1, G = \emptyset$
1 Run RLF on B and select the first group, and move selected exams from B to G
2 Schedule up to Ncourses from G in S_{ij} (Line 3 from Sub-Algorithm (i))
3 **if** $A = C$ **then** End
4 **if** $j = S \wedge i$ ***is*** *odd* **then**
5 Move scheduled courses from G to A
$i \longleftarrow i + 1, j = 1$, **Go to:** Line 2
6 **end**
7 **if** $j = S \wedge i$ ***is*** *even* **then**
8 Move scheduled courses from G to A
Return remaining courses in G to B, $i \longleftarrow i + 1, j = 1$, **Go to:** Line 1
9 **else**
10 Move scheduled courses from G to A
$j \longleftarrow j + 1$, **Go to:** Line 2
11 **end**

The problem's complexity correlated with enforcing condition 4.a is much higher than the others (4.b, 4.c, and 4.d). This is because it is necessary to consider days $i - 1$ and $i + 1$ when scheduling day *i*. Moreover, it is necessary to consider individual students. Note that if we do not look at the students and just consider the course conflicts, then we may unnecessarily move courses to the temporary set. This can be explained by a simple example. Consider having five courses and only 3 students; all students are registered in course 1. However, courses 2 and 3 only have student #1 while courses #4 and #5 have student #2 and student #3, respectively. If course #1 and course #2 were scheduled in a certain day, then we have a conflict caused by student 1; hence, we cannot schedule course #3 on the next day, but we do not have a problem with courses #4 and #5. However, if we do not analyze the students' schedules, but only analyze the course conflicts, then we can see that course #1 has conflicts with courses #3, #4, and #5. This leads to a false conclusion that it is not possible to schedule any of the courses on the following day. This demonstrates the importance of analyzing the exam schedules of individual students separately.

Condition 5.a (special requests for a certain day) depends on the policy chosen in condition 4. However, regardless of the chosen policy, the concerned courses should be removed from the list of remaining courses (to be scheduled) and recorded as the first course in the time slot S_{ij}. When the normal scheduling process reaches slot S_{ij}, the pre-assigned courses are set as the first vertex when the graph coloring algorithm is called. This ensures that the pre-assigned course is included in the color group, and avoids conflicts among courses in the concerned slot. We show below how condition 5.a is treated under each choice of condition 4.

4.a Algorithm E
4.b The requested course is included in day *i*
4.c Algorithm D
4.d If *i* is even, the requested course is included in day *i-1*. If *i* is odd, it is included in day *i*.

Algorithms D and E place all courses that conflict with the special requests in the temporary list of forbidden courses. This is done while taking into consideration the respective policy from condition 4. For example, if two exams can be scheduled for a student in one day, and one exam is already scheduled on the requested course's day, then all the student's other courses are removed to ensure that the requested course will be the student's second exam. Moreover, no student will have exams on two consecutive slots. If the institution observes a policy of having no more than two exams in a window of two days, then the algorithm also inspects the previous day to ensure this policy is respected.

Algorithm D. Special request to schedule an exam in S_{ij} (under condition 4.c)

Condition 4.c: no more than two exams per day for each student
On day i **Init**: k=1
1 **if** $k < j - 1$ **then**
2 Schedule slot S_{ik} **Go to:** Line 7
3 **else if** $k = j - 1$ **then**
4 Remove all courses conflicting with the requested course
Schedule $S_{ik}, k = k + 1$, **Go to:** Line 6
5 **else if** $k = j$ **then**
6 Schedule S_{ij} while ensuring that the requested course is included[1]
7 List students that have a conflict with the requested course Move all their other courses from B to T_2
$k \longleftarrow k + 1$, **Go to:** Line 1

Algorithm E. Special request to schedule an exam in S_{ij} (under condition 4.a)

Condition 4.a: no more than two exams in two days for each student
On day $i - 1$, **Init**: $j = 1$
1 Schedule slot $S_{i-1,j}$
2 List students that have a conflict with the requested course
3 Move all their other courses from B to T_1
4 **if** $j \neq S$ **then**
5 $j \longleftarrow j + 1$, **Go to:** Line 1
6 **end**
On day i, follow Algorithm D.

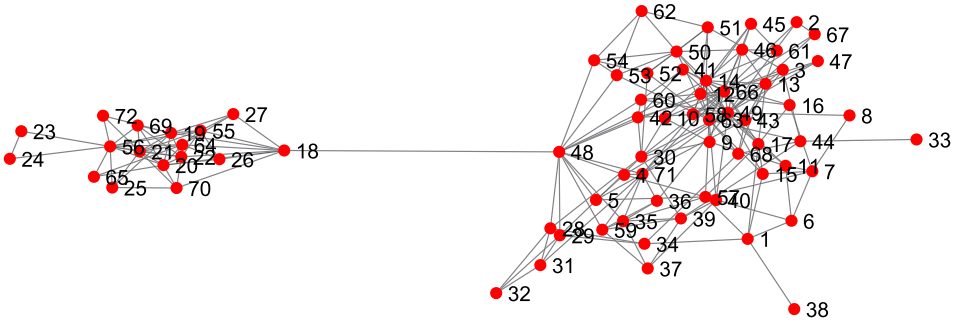


Fig. 6. Network representation–MI case study.

For special requests (condition 5.b), Algorithm E should be used in addition to assuming that the exam is to be scheduled on the last slot of the day. If the course was scheduled in an earlier slot, without special allocation, then we drop the assumption and proceed normally. To meet condition 5.c (special request to avoid a certain day) the concerned course is placed in the temporary list of forbidden courses, but reconsidered for scheduling after this day. Caution is required if the concerned day is the last exam day. In this case, the concerned exam must be scheduled, at the latest, on the last slot of the previous day. The concerned course needs not be removed from the list of courses available for scheduling on any day until it has actually been allocated. Accommodating condition 5.d (exam duration is longer than the time frame of one slot) requires the exam venue to be free for the following slot.

Consideration 8 concerns scheduling multiple class-sections of the same course in the same time slot. Note that it is not necessary to have all sections in the same exam room. To account for this consideration, the user must combine the sections together into one course containing all students registered across the sections. When this dummy course is scheduled, all the required sections are scheduled together. When the dummy courses vertex is chosen for scheduling from a color group, the separate sections can be distributed individually on available exam venues. Finally, it is important to note that the proposed algorithms are designed to schedule exams within a continuous period. For example, the exams' period can extend from Monday to Friday. If, however, a recess day is given, such as a weekend, a study day, or a public holiday, then the user must return all exams in the temporary list of forbidden courses (if any) to the set of courses available for scheduling on the last day before the recess. By default, the algorithm assumes that there is no gap between any two days. Hence, this adjustment is necessary.

5. Masdar Institute case studies

MI is a research-oriented institution that runs engineering graduate programs only. Master students usually enroll in two or three courses per semester, while PhD students enroll in one or two courses only. In this section, we discuss four real case studies acquired from the institution.

5.1. Case study M1

The total number of students is 383. The total number of registrations is 865 distributed across 72 courses, noting that one student taking three courses counts as three course registrations. Hence, the adjacency matrix is 72×72. The systems graph has 229 edges (conflicts). The network and matrix representations are shown in Figs. 6 and 7, respectively. Using Masdar's considerations, we deduce that conditions 1, 2, 3, 4.a, 6, and 7.a are necessary. Condition 5 is excluded as there are no special requests for this semester. Thus, Algorithm A with condition 7.a from the "Steps to Schedule up to N courses" will be employed. However, since satisfying condition 4.b (one exam per day) also satisfies condition 4.a (two exams in a window of two days), we can run Algorithm B first. In fact, the latter option is preferred by students. If the output from Algorithm B is a schedule that requires more than 5 days, it is necessary to use Algorithm A which is the case for the given data. Running Algorithm B on the concerned data set returns an exam schedule with 6 exam days. Hence, Algorithm A is required to solve the scheduling problem.

The input parameters used are:

1. number of days: 5
2. number of slots per day: 3
3. number of exam rooms: 8
4. exam room capacities:
   a. classrooms 1–5: 24
   b. classrooms 6 and 7: 5
   c. classroom 8: 65

The inputs to Algorithm A are the parameters shown above, the adjacency matrix, and student lists for each course. Moreover, when using the proposed LB algorithm, a value of $ND = 1$ is used (number of high degree vertices to include), ensuring that at least one high degree vertex is scheduled whenever a group is selected. Note that other ND values can also be tried if $ND = 1$ does not generate the desired results. However, if the parameter ND is too high, the method will act as RLF and will face the risk of having a smaller color group to schedule. In that case, adhering to the exam room capacity limits would become more difficult. Algorithm A is run showing successful results and respecting all the policies and constraints set by MI.

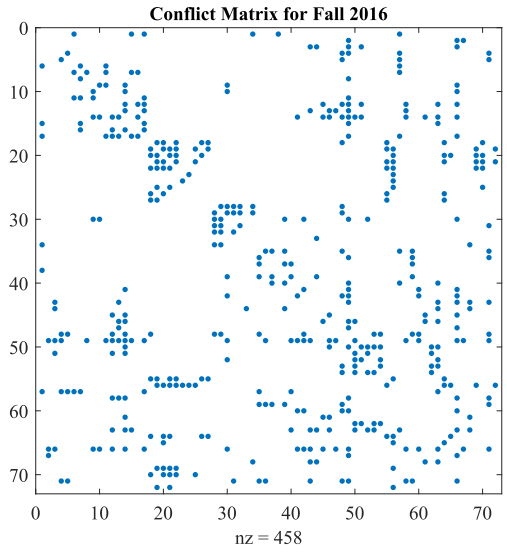


Fig. 7. Matrix representation–MI case study.

5.2. Case studies with special requests: M2, M3, M4

Three case studies are tackled in this section. The data sets correspond to three separate semesters ($M2$, $M3$, and $M4$). The list below names the special conditions each semester has. These conditions were not incorporated in the previous case study from the methodology section.

Semester *M2*:

1. Course 18 cannot be scheduled on the third day
2. Course 130 requires extra time (4 h)

Semester *M3*:

1. Course 130 requires extra time (4 h)
2. Course 156 must be scheduled within the first two days

Semester *M4*:

1. Course 52 must be scheduled on the first day
2. Course 102 must be scheduled on the last day

Algorithm B could schedule the courses of semester $M2$, and meet the requirements, accounting for the policy of having a maximum of one exam in one day for each student, which is preferable by students and automatically satisfies the institute's first policy of two exams in a period of two days. Semesters $M3$ and $M4$ are both scheduled using Algorithm A with $ND = 2$. Some course exams require more time than the period of a single slot. In such cases, no exam is scheduled in the same venue in the following time slot. This was accounted for as shown in Tables 1 and 2. For example, course 130 in semester $M2$ is scheduled in slot 1 of day 4, in class 8. Thus, class 8 is empty for the following slot. Note that the tables list only the days that contain courses with a special request.

Some instructors request their course not be scheduled on a certain day. On the concerned day, the course is placed in the temporary list of forbidden courses and returned to the set of remaining courses. Caution must be exercised if the concerned day is the last exam's day. In this case, it is necessary for the exam to be scheduled on the last slot on the second last day at the latest. It can be observed that course 18 of semester $M2$ is required to not be scheduled on the third day. Hence, it is scheduled in Slot 2 of day 4. For the special request of scheduling course 156 of Semester $M3$ within the first two days, the course is treated as if it cannot be scheduled on any of the last three days. This is equivalent to having only three exam days and assuming that the course cannot be scheduled on the last day. The course was successfully scheduled in slot 3 of day 1.

To accommodate a special request of scheduling a course on a certain day, Algorithm E is used while considering that the exam is required in the final slot of that day. However, the course should not be removed from the set of remaining courses to schedule during that day. This ensures the course will be scheduled at the latest by the end of the concerned day. Moreover, if it was scheduled in an earlier possible slot, then the assumption can be dropped and the algorithm proceeds normally. As a result, course 52 and course 102 of semester $M4$ were scheduled in slot 1 of day 1 and slot 1 of day 5, respectively. This satisfies the requirements (Table 3). Finally, to prove the need of the proposed graph coloring algorithm (LB) in Algorithm A, we employed the RLF algorithm while scheduling the exams of semester $M3$, instead of the LB. The algorithm failed to schedule all exams in the given 5 days. This illustrates the advantage in applying the LB algorithm in the proposed exam timetabling heuristic.

Table 1
Semester *M2* - schedule of Day 4.

	Day 4		
Class	Slot 1	Slot 2	Slot 3
1	Course 89	Course 156	Course 77
2	Course 58	**Course 18**	Course 79
3	Course 121	Course 116	
4	Course 154	Course 155	
5	Course 44	Course 56	
6	Course 37	Course 69	
7	Course 49	Course 75	
8	**Course 130**		

Table 3
Semester *M4*- schedule of days 1 and 5.

	Day 1			Day 5
Class	Slot 1	Slot 2	Slot 3	Slot 1
1	Course 138	Course 48	Course 13	Course 143
2	Course 64	Course 5	Course 129	Course 56
3	Course 91	Course 107	Course 122	Course 1
4	Course 63	Course 127	Course 113	**Course 102**
5	**Course 52**	Course 74	Course 144	
6	Course 66	Course 76	Course 20	
7	Course 80	Course 146	Course 135	
8	Course 140	Course 50	Course 41	Course 58

6. Computational study

In this section, we test the exam timetabling heuristics against the use of an IP formulation. There are no other algorithms in the literature that can capture the idea of scheduling a maximum of two exams in two days for a student (Policy 4.a in Section 3) which is one of the main contributions of the paper. Thus, it is essential to use the IP formulation as a benchmark for comparison with the proposed heuristics. We examine both the accuracy of the method and the computational time. All instances and case studies are run on a computer with an X5680 CPU @ 3.33 GHz processor and 12 GB RAM.

Table 4 shows the information of the case studies. The first four are the MI cases acquired from the institute's registrar's office and the rest

Table 2
Semester *M3* - schedule of days 1 and 3.

	Day 1			Day 3		
Class	Slot 1	Slot 2	Slot 3	Slot 1	Slot 2	Slot 3
1	Course 49	Course 50	Course 111	Course 100	Course 69	**Course 130**
2	Course 154	Course 6	Course 121	Course 3	Course 75	Course 94
3	Course 37	Course 88	Course 27	Course 15	Course 79	Course 28
4	Course 119	Course 18	Course 103	Course 67		Course 90
5	Course 101	Course 104	Course 22	Course 73		Course 147
6	Course 108	Course 33	**Course 156**	Course 93		Course 51
7	Course 30	Course 23	Course 10	Course 71		
8	Course 140	Course 62	Course 38	Course 45	Course 123	Course 40

Table 4
Exam timetabling computational study cases.

	Case name	Exams	Students	Enrollments	Density
1	Masdar_Sem_M4	66	371	827	0.10
2	Masdar_Sem_M3	70	377	774	0.09
3	Masdar_Sem_M2	72	320	630	0.07
4	Masdar_Sem_M1	72	382	864	0.09
5	HEC92	81	2823	10632	0.42
6	STA83	139	611	5751	0.14
7	YOR83	181	941	6034	0.29
8	UTE92	184	2750	11793	0.08
9	EAR83	190	1125	8109	0.27
10	TRE92	261	4360	14901	0.18
11	LSE91	381	2726	10918	0.06
12	KFU93	461	5349	25113	0.06
13	RYE92	482	11483	45051	0.07
14	CAR92	543	18419	55522	0.14
15	UTA92	622	21266	58979	0.13
16	CAR91	682	16925	56877	0.13

are real cases from the benchmark data sets by University of Nottingham. The table shows the number of exams, students, enrollments, and density for each case. The density is obtained by dividing the number of edges by the possible number of edges in the given problem, as shown by Eq. (17).

$$d = \frac{E}{\binom{V}{2}} \tag{17}$$

where, d is the density, V is the number of vertices (exams), and E is the number of edges (conflicts).

Table 5 shows the results of the computational study. Specifically, the table compares the solutions and computational run-time of the two methods used to solve the problem. Note that the IP formulation is solved using the commercial solver *CPLEX*. The *Days* columns show the number of days the methods require to schedule the exams. And the *Gap (%)* shows the percent difference between the obtained solution and highest lower bound at the time the Algorithm is stopped. For consistency, all cases are assumed to have three slots per day and eight exam rooms per slot where only one exam is scheduled per exam room. This leads to high respective restrictions in cases with a large number of exams which results in the requirement of long exam schedules.

As Table 5 shows, the proposed heuristic generally outperforms solving the IP formulation. In the smaller cases (rows 1–4), the IP formulation reached the optimal solution in reasonable time (< 30 s); however, the heuristic was able to reach an optimal or near-optimal (in one case) solution in a much faster computational time (< 0.2 s). In the larger cases (higher number of exams and/or number of students), the IP formulation was stopped when it passed 1000 s. At that point, the software reports the best integer solution found. However, in most cases the computer ran out of memory and failed to find an integer solution before 1,000 s while the heuristic gave a solution in a reasonable time (see rows 8, 10–16 in the table). This highlights the importance of the proposed heuristic for solving the problem. Moreover, in the cases where the IP formulation found an integer solution within the allocated 1000 s (see rows 5–7), the heuristic found a better solution in a much shorter computational time. Note that the gaps obtained by the optimization model in the latter cases are large; however, this does not necessarily mean that the obtained solution is far from the optimal. Specifically, it may be due to the difficulty of obtaining tighter lower bounds that are generally obtained from the LP relaxation of the model. Nevertheless, the heuristic was able to find better solutions in a shorter time.

Table 5
Exam timetabling computational study.

		IP formulation			Proposed heuristic		
	Case name	Days	Gap (%)	Runtime (s)	ND	Days	Runtime (s)
1	Masdar_Sem_M4	4	0	26.33	3	5	0.11
2	Masdar_Sem_M3	5	0	13.65	2	5	0.12
3	Masdar_Sem_M2	5	0	13.76	3	5	0.12
4	Masdar_Sem_M1	5	0	20.62	3	5	0.15
5	HEC92	19	94.74	> 1000	3	18	0.87
6	STA83	14	60.38	> 1000	4	13	0.84
7	YOR83	25	96.00	> 1000	5	19	1.20
8	UTE92	n/a	n/a	OOM[a]	4	10	3.24
9	EAR83	n/a	n/a	> 1000[b]	5	22	1.55
10	TRE92	n/a	n/a	OOM[a]	6	20	3.56
11	LSE91	n/a	n/a	OOM[a]	5	16	14.80
12	KFU93	n/a	n/a	OOM[a]	5	20	33.58
13	RYE92	n/a	n/a	OOM[a]	4	22	34.98
14	CAR92	n/a	n/a	OOM[a]	6	24	35.56
15	UTA92	n/a	n/a	OOM[a]	5	28	54.87
16	CAR91	n/a	n/a	OOM[a]	6	31	71.73

[a] OOM: Out of Memory.
[b] Runtime exceeded 1000 s and no integer solution was found.

Different *ND* values can yield different solutions deviating in a range of around 2 days, and the best is shown in the table. Note that the *ND* parameter does not affect the computational time. The *ND* parameter controls the number of high degree vertices to be included in the group to schedule. The number of days is observed to decrease with the increase of *ND* and then increases again to reach higher values. As discussed earlier, the LB algorithm's performance tends to mimic the performance of RLF at higher values of *ND* (8 in this case: equal to the number of exam rooms). Values of *ND* within the range [2, 6] are correlated with the best solutions. This demonstrates the advantage of using the LB algorithm.

7. Conclusions and future work

In this paper, we consider the exam timetabling problem under several realistic constraints. A new constraint, not investigated in the literature, is considered in this paper, namely: scheduling a maximum of two courses in two days for each student. This constraint increases the complexity of the problem as it becomes necessary to analyze each student's course enrollments. We formulate the problem using an integer program. Due to the problem's complexity, the mathematical model is computationally inefficient for large problems. Thus, we propose a constructive exam timetabling heuristic that captures all the realistic considerations discussed in the paper. The exam timetabling heuristic includes a novel color graphing algorithm, which finds the color group with the largest number of non-conflicting vertices. This is different from the color graphing algorithms in the literature, whose goal is to find the smallest number of non-conflicting vertex groups. The proposed algorithm's unique perspective and scope allow more flexibility while scheduling exams, in addition of facilitating adherence to exam room capacity constraints. We provide real case studies to discuss the different situations that the algorithm can capture. Finally, we provide a computational study for the proposed heuristic and prove its computational efficiency against the IP formulation showing optimal or near-optimal results.

The proposed heuristic contains an adjustable parameter *ND* which controls the number of high degree vertices to include in the slot under study. A choice of $ND = 0$ does not include any high degree vertices in the schedules. Thus, courses with more students are not given priority to be scheduled earlier. However, as *ND* is increased, higher degree vertices are included at the cost of higher risk of failing to schedule the exams within the given number of days as explained in Section 4. Thus, as future work, the optimal choice of *ND* can be studied and linked to the network characteristics. Moreover, varying *ND* through the scheduling process (dynamic *ND*) can also be studied. The presented algorithms adopt a primitive definition of degree-centrality and

demonstrate acceptable performance. However, other definitions of centrality (i.e., eigenvector centrality) can be studied to observe their effect on the solution and to compare with the definition used in this paper. Finally, the proposed IP formulation was solved using commercial software to benchmark the results of the proposed solution approach. However, advanced optimization techniques can also be designed and used to solve the IP formulation. Such approaches can be investigated and compared with this paper's solution approach as future work.

References

Akbulut, A., & Yilmaz, G. (2013). University exam scheduling system using graphcoloring algorithm and rfid technology. *International Journal of Innovation, Management and Technology, 4*, 66.

Bykov, Y., & Petrovic, S. (2016). A step counting hill climbing algorithm applied to university examination timetabling. *Journal of Scheduling, 19*, 479–492.

Fleurent, C., & Ferland, J. A. (1996). Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research, 63*, 437–461.

Garey, M. R., Johnson, D. S., & Stockmeyer, L. (1974). Some simplified np-complete problems. *Proceedings of the sixth annual ACM symposium on Theory of computing* (pp. 47–63). ACM.

Han, L., & Han, Z. (2010). A novel bi-objective genetic algorithm for the graph coloring problem. *2010 second international conference on computer modeling and simulation* (pp. 3–6). IEEE.

Hassan, M. A. H., & Hassan, O. A. H. (2016). Constraints aware and user friendly exam scheduling system. *International Arab Journal of Information Technology, 13*, 156–162.

Hertz, A., & de Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing, 39*, 345–351.

Laporte, G., & Desroches, S. (1984). Examination timetabling by computer. *Computers & Operations Research, 11*, 351–360.

Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards, 84*, 489–506.

Malkawi, M., Hassan, M. A. H., & Hassan, O. A. H. (2008). A new exam scheduling algorithm using graph coloring. *International Arab Journal of Information Technology (IAJIT), 5*.

Marappan, R., & Sethumadhavan, G. (2013). A new genetic algorithm for graph coloring. *2013 fifth international conference on computational intelligence, modelling and simulation (CIMSim)* (pp. 49–54). IEEE.

Mehrotra, A., & Trick, M. A. (1996). A column generation approach for graph coloring. *Informs Journal on Computing, 8*, 344–354.

Méndez-Díaz, I., & Zabala, P. (2006). A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics, 154*, 826–847.

Mohamed, A., Mushi, A. R., & Mujuni, E. (2013). An examination scheduling algorithm using graph coloring–the case of Sokoine University of Agriculture. *International Journal of Computer Engineering & Applications, 2*, 116–127.

Mumford, C. L. (2006). New order-based crossovers for the graph coloring problem. *Parallel problem solving from nature-PPSN IX* (pp. 880–889). Springer.

Newman, M. (2010). *Networks: An introduction.* Oxford University Press.

Nguyen, T. H. & Bui, T. (n.a.) Graph coloring benchmark instances. [Online]. Available: <https://turing.cs.hbg.psu.edu/txn131/graphcoloring.html> (accessed Oct. 22 2016).

Qu, R., Burke, E. K., McCollum, B., Merlot, L. T., & Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling, 12*, 55–89.

Rachmawati, H., Armay, E. F., & Purnomo, M. H. (2012). Problem solving analysis of course scheduling using graph coloring technique based on bee colony algorithm: Parameter of lecturer priority as soft constraint in Electrical Engineering Department of Sepuluh Nopember Institute of Technology. *2012 7th international conference on telecommunication systems, services, and applications (TSSA)* (pp. 136–141). IEEE.

Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2009). Roulette wheel graph colouring for solving examination timetabling problems. *International conference on combinatorial optimization and applications* (pp. 463–470). Springer.

Sabar, N. R., Ayob, M., Qu, R., & Kendall, G. (2012). A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence, 37*, 1–11.

The University of Nottingham (n.a.) Benchmark data sets in exam time tabling [Online]. Available: <http://www.cs.nott.ac.uk/pszrq/data.htm> (accessed June 21 2017).

Woumans, G., De Boeck, L., Beliën, J., & Creemers, S. (2016). A column generation approach for solving the examination-timetabling problem. *European Journal of Operational Research, 253*, 178–194.