European Journal of Operational Research 34 (1988) 372–383
North-Holland

Theory and Methodology

A Lagrangian relaxation approach to solve the second phase of the exam scheduling problem

Taghi ARANI, Mark KARWAN and Vahid LOTFI
State University of New York at Buffalo, Buffalo, NY 14260, USA

Abstract: The problem of scheduling final exams at a large university can be viewed as a three phase process. The first phase consists of grouping the exams into sets called exam blocks. The second phase deals with the assignment of exam blocks to exam days and the third phase consists of arranging the exam days and also arranging the blocks within days.

In this paper, we present new integer programming formulations for the second phase of the scheduling problem. We present an integer program with a single objective of minimizing the number of students with two or more exams per day. We then present a Lagrangian relaxation based solution procedure to solve this problem. Further, we present a bicriterion integer programming formulation to minimize the number of students with two exams per day and the number of students with three exams per day. Finally, we present some computational experience using randomly generated problems as well as real world data obtained from the State University of New York at Buffalo.

1. Introduction

A review of the literature on scheduling of final examinations reveals that the primary goal has been to develop an examination schedule so that no student is required to take more than one exam in any time period. This, in fact, is such a common objective that the secondary objectives which have serious effects on large number of students have lost importance. Many authors use a graph theoretic approach to generate the overall exam schedule (see for example [3,5,8,13,14,15,16]). Various restrictions introduced to the problem limit the use of the graph theoretic approach. The integer programming formulations of this problem are generally so large in size that existing solution techniques are of little value.

Due to the large scale structure of the exam scheduling problem and its high degree of complexity arising from various objectives, one may resort to a multiphased solution procedure. In this case, the problem can be broken into three separate but interrelated scheduling phases.

The first phase involves the grouping of exams into sets called blocks in such a way that either i) no student is to participate in two exams scheduled in the same block, or ii) the number of students who will take two or more exams in the same block is minimum. Another alternative is to develop an assignment having an upper limit on the number of students taking two exams in the same block. In the past two decades, more than a dozen different heuristics have been developed to solve the first phase of the problem (see for example [4,7,9,10,11,18,21]).

The authors wish to thank the reviewers for their valuable suggestions.

Received October 1986; revised January 1987

The second phase uses the results from the first phase as input. Here the objective is to assign the exam blocks (each containing one or more exams) to exam days such that either (i) no student is required to participate in more than one exam at the same day, or (ii) the number of students participating in two or more exams at the same day is minimum, or (iii) the number of students taking consecutive exams in the overall exam duration is minimum.

The third phase involves the arrangement of exam blocks in each day (i.e. the assignment of blocks to periods) and the arrangement of exam days in such a way that some measure of student comfort is maintained. The primary objective can be (i) to minimize the number of students taking consecutive exams in any day, (ii) to minimize the number of students taking exams in the last period of one day and the first period of the next day.

In achieving any one of the objectives in each scheduling phase, a set of restrictions imposed by the university administration should be taken into consideration. Two major restrictions imposed are (i) the total number of days available for the entire exam schedule, and (ii) number of time periods per day. Other restrictions such as preassignment of some exams to certain time periods should also be considered. In short, the exam schedule should be satisfactory to three groups of people: professors (departments) offering courses, students taking the exams, and the administration in charge should all consider the schedule a desirable one.

In this paper we present a solution technique for the second phase of the multi-phased exam scheduling problem. We utilize a Lagrangian relaxation approach to solve the Integer Programming problem associated with the assignment of exam blocks to exam days.

2. Assignment of exam blocks to time periods

Suppose that a solution to phase one of the exam scheduling problem results in the assignment of all exams to NP exam blocks. One aim of phase two was described as the assignment of these NP blocks (each containing a certain number of exams) to a predetermined number of days (ND) each containing NPD time periods in such a way that the number of students taking consecutive exams is minimized. When the last time period of one day and the first time period of the next day are considered consecutive the problem may be converted into a traveling salesman problem. In this case, each block may be considered a city and the optimum salesman tour constitutes the arrangement of the NP exam blocks with minimum number of students having consecutive exams.

Another possibility is that each day contains exactly two time periods and the second period of one day is not considered adjacent to the first period of the next day. In this case, the assignment of the NP exam blocks to the time periods reduces to a minimum weighted matching problem. The end-nodes of the edges in the matching are the pairs of exam blocks scheduled in the same day.

Colijn [7] uses a traveling salesman approach to assign the exam blocks to time periods. A heuristic procedure is implemented to solve the salesman problem. The approach reduces the number of students with consecutive exams by about 30 percent. White and Chan [21] present a computer program which uses a variation of an algorithm by Peck and Williams [18] to assign the exams to blocks. They proceed by using a traveling salesman approach to assign the blocks to time periods.

In a case where there are three or more exam periods per day, the traveling salesman approach may be used to assign the exam blocks to exam periods. However, this approach suffers from the characteristic that many students may be required to participate in two or more exams in one day. For example, in a three period per day problem, blocks with common students may be assigned to the first and third periods of the same day. This will minimize consecutiveness but may increase the number of students with two or more exams per day. This is of special importance when the available number of blocks is relatively small. Further, in a case where the administration plans to reduce the number of exam days (consequently the number of available blocks), it is important to examine the increase in the number of students with two or more exams per dry. This was the case at the State University of New York at Buffalo (SUNYAB), where the number of blocks was reduced from 21 (7 days with 3 periods each) to 15 (5 days with 3 periods each) for the Spring 1985 exam schedule.

The problem of assigning the exam blocks to exam days, with the objective of minimizing the number of students with two or more exams per day, is neither a traveling salesman problem nor a weighted matching problem. In section three we present a new integer programming formulation for the proposed problem. Section four contains the solution procedure for this integer programming problem. In section five we present a bicriterion integer programming formulation with two objectives: minimizing the number of students with two exams per day and minimizing the number of students with three exams per day. In section six we present some computational experience on randomly generated problems as well as the results of SUNYAB actual data.

3. Problem formulation

Suppose all final examinations are scheduled in NP blocks. Without loss of generality, we assume that the number of periods per day is the same for each day and is equal NPD. Then, the total number of ways the NP blocks can be grouped into ND days is given as

$$N=\binom{\text{NP}}{\text{NPD}}.$$

Note that when $\text{NP} > \text{NPD} * \text{ND}$ the problem does not have a feasible solution. In case $\text{NP} < \text{NPD} * \text{ND}$, introduce extra dummy blocks to make them equal. Then, the NP blocks can be arranged in ND days (NPD periods per day) in N different ways. For example, there are 20 different arrangements of $\text{NP} = 6$ blocks into groups of $\text{NPD} = 3$ (number of periods per day) as follows:

$$\{1,2,3\},\ \{1,2,4\},\ \{1,2,5\},\ \{1,2,6\},$$
$$\{1,3,4\},\ \{1,3,5\},\ \{1,3,6\},\ \{1,4,5\},$$
$$\{1,4,6\},\ \{1,5,6\},\ \{2,3,4\},\ \{2,3,5\},$$
$$\{2,3,6\},\ \{2,4,5\},\ \{2,4,6\},\ \{2,5,6\},$$
$$\{3,4,5\},\ \{3,4,6\},\ \{3,5,6\},\ \{4,5,6\}.$$

Another observation to be made is that each block appears in exactly

$$M=\binom{\text{NP}-1}{\text{NPD}-1}$$

arrangements. In the above example each block appears in exactly 10 different arrangements. Let c_j be the total number of students having two or more exams in the exam blocks that form the j-th arrangement and let $y_j = 1$ if the j-th arrangement is a part of an optimum solution and $y_j = 0$ otherwise. Then, the problem of arranging the NP blocks in ND days, such that the total *number of students participating in two or more exams* in one day is minimized, is given as

$$\text{(P)}\quad \text{Min}\quad Z=\sum_{j=1}^{N} c_j y_j, \tag{1}$$

$$\text{s.t.}\quad \sum_{j=1}^{N} a_{ij} y_j = 1,\quad i=1,2,\ldots,\text{NP}, \tag{2}$$

$$\sum_{j=1}^{N} y_j = \text{ND}, \tag{3}$$

$$y_j = 0, 1,\quad j=1,2,\ldots,N, \tag{4}$$

where $a_{ij} = 1$ if block i is in arrangement j and $a_{ij} = 0$ otherwise. (1) represents the total number of students participating in two or more exams in the same day. (2) forces each block to be assigned to exactly one day. (3) is used to limit the number of days to exactly ND days.

The coefficients of (2) are all zeros and ones. Also, since each block appears in exactly M different arrangements, there are exactly M ones in each constraint of (2). Each arrangement is composed of exactly NPD periods which results in each column having exactly NPD ones. That is, each y_j appears in exactly NPD rows of (2) and then once in (3). From these observations it is clear that (3) is *redundant* since it can be presented as the sum of all NP constraints of (2).

As a result we can remove (3) from (P) and (P) becomes a set partitioning problem with *no possible* row or column dominations (still a hard problem to solve).

In the light of the previous observations on the arrangement characteristics, the equality of sign of (2) may be replaced with greater than or equal to if (3) is reinstated. That is, suppose the set $\text{PS} = \{j \mid y_j = 1\}$ forms a least cost partial solution to (P) with equality replaced by greater than or equal to signs and (3) omitted. Further, suppose that $|\text{PS}| = \text{ND}$ with $\sum_{j\in \text{PS}} a_{ij} y_j \geqslant 1$ for $i = 1, 2, \ldots, \text{NP} - 1$. To satisfy the last constraint, we will select the next least cost variable y_k with $a_{\text{NP},k} = 1$ and set it equal to 1. But this will violate constraint (3) since $|\text{PS}|$ will equal $\text{ND} + 1$. Consequently, (P) may be stated as a set covering problem with an extra side constraint (3).

4. Solution method

As mentioned earlier, (P) can be written as a set covering problem with an additional constraint or a partitioning problem with no row or column reductions possible. Several specialized versions of the implicit enumeration approach of Balas [2] have been proposed for the set covering and partitioning problems. The most relevant of these methods to solve (P) is by Pierce [19] and Pierce and Lasky [20]. Their approach is one of systematically searching the solution space by generating partial solutions and exploring the logical implications of these partial solutions. The special structure of (P) simplifies the logical tests in a branch and bound type algorithm. Although the problem considered by Pierce and Lasky does not contain the extra constraint, (3), they essentially create such a constraint for use in solving their set covering problem.

To enhance the efficiency of the implicit enumeration approach better bounds may be found for the subproblem at each node. While Michaud [14] suggests solving the linear relaxation over all free variables, Pierce and Lasky propose solving the linear programming relaxation after adding all of the constraints (i.e., a surrogate relaxation). In our problem, adding all of the partitioning constraints gives us constraint (3).

We suggest a Lagrangian relaxation approach to obtain better bounds at each node. For problem (P) a natural Lagrangian relaxation is defined below by relaxing constraint set (2).

$$\text{(Pu)} \quad \text{Min} \quad Z = \sum_{j=1}^{N} c_j y_j + \sum_{i=1}^{\text{NP}} u_i \sum_{j=1}^{N} (1 - a_{ij} y_j), \tag{5}$$

$$\text{s.t.} \quad \sum_{j=1}^{N} y_j = \text{ND}, \tag{6}$$

$$y_j = 0, 1, \quad j = 1, 2, \dots, N. \tag{7}$$

Equivalently, (Pu) may be written as

$$\text{(Pu)} \quad \text{Min} \quad Z = \sum_{j=1}^{N} \left(c_j - \sum_{i=1}^{\text{NP}} a_{ij} u_i \right) y_j + \sum_{i=1}^{\text{NP}} u_i, \tag{8}$$

$$\text{s.t.} \quad \sum_{j=1}^{N} y_j = \text{ND}, \tag{9}$$

$$y_j = 0, 1, \quad j = 1, 2, \dots, N. \tag{10}$$

Our Lagrangian relaxation may be viewed in different ways. Considering constraint (3) as redundant, we have relaxed constraint set (2) with dual multipliers but strengthened the bound by enforcing the surrogate constraint (3) similar to Pierce and Lasky. Alternatively, now that constraint set (2) is relaxed, constraint set (3) is no longer redundant and hence strengthens any resultant Lagrangian relaxation.

Define $v(\cdot)$ as the value of an optimal solution for problem $(\cdot)$. (9) is a Generalized Upper Bound (GUB) constraint hence (Pu) has the integrality property. That is, $v(\text{Pu}) = v(\overline{\text{Pu}})$ where $(\overline{\text{Pu}})$ is the linear relaxation of (Pu). Thus using the dual multipliers $(\bar{u})$ obtained by solving $(\bar{\text{P}})$, the linear relaxation of (P), the following holds

$$v(\bar{\text{P}}) = v(\bar{\text{P}}_{\bar{u}}) = v(\text{P}_{\bar{u}}) = \max_{u} v(\text{P}_u) = v(\text{D}_l).$$

Where D_l is the Lagrangian dual. By closely studying the problem structure, bounds obtained from $(\overline{\text{Pu}})$ can be significantly strengthened beyond $v(\bar{\text{P}})$.

To begin, (Pu) is solved by inspection in a very efficient manner. That is, let I be the index set of those variables set at 1 so far. Among the $N - |I|$ remaining variables determine the $k = \text{ND} - |I|$ lowest reduced cost variables and set them all to one. In order to improve the quality of the bound, we impose two side constraints on (Pu) as follows.

Side constraint 1

Suppose at some node of the branch and bound tree $|I|$ variables have been set to 1 and we need to set k more variables to 1 to solve (Pu). Let the first free variable selected be $y_{[i]}$ with a reduced cost of $(c-u)_{[i]}$. The next lowest cost variable $y_{[j]}$ is examined for a conflict with $y_{[i]}$. That is, if there exists at least one exam block in arrangement $y_{[i]}$ which also appears in the arrangement $y_{[j]}$ one of these variables will be zero in an optimum solution to (P). Therefore, $y_{[j]}$ with a reduced cost $(c-u)_{[j]} > (c-u)_{[i]}$ is ignored and $y_{[i]}$ is set to 1 and underlined as part of the solution. In a similar manner, the second lowest cost free variable is selected. This process of variable selection continues until the k required variables are selected. It should be noted that once a variable is utilized in setting another variable to zero it is underlined and can no longer be used in setting other variables. This is required to ensure a valid lower bound. For example, suppose $k = 2$ and that the ordered cost coefficients of the free variables $y_{[i1]}$, $y_{[i2]}$, $y_{[i3]}$, $y_{[i4]}$ and $y_{[i5]}$ are

Variable	$y_{[i1]}$	$y_{[i2]}$	$y_{[i3]}$	$y_{[i4]}$	$y_{[i5]}$
Cost	3	7	9	19	30
Exam	1	2	2	1	4
blocks	2	3	5	5	6
	$\underline{3}$	4	$\underline{6}$	6	7

Without using the side constraint 1, (Pu) is solved by setting $y_{[i1]}$ and $y_{[i2]}$ to 1 and the increase in lower bound will be 10. When using side constraint 1, $y_{[i1]}$ is in conflict with $y_{[i2]}$ (due to the common block 3). The least cost variable $y_{[i1]}$ is set to 1 and $y_{[i2]}$ is set to 0. $y_{[i1]}$ is then underlined so that it will not be used to set another variable to zero. The next least cost variable $y_{[i3]}$ is set to 1 and (Pu) is solved. In this case the increase in lower bound is 12 or the duality gap is further closed by 2 (12 – 10) units. As mentioned above, if $y_{[i1]}$ is allowed to set other conflicting variables (eg. $y_{[i3]}$ and $y_{[i4]}$) to 0, $y_{[i5]}$ would have been set to 1 as the second variable. This would have increased the bound by 33. This bound, however, is invalid since $y_{[i2]}$ and $y_{[i4]}$ are not conflicting either and increase the bound by 26. The relaxation (Pu) with side constraint 1 appended will be called (Pũ). In essence we have inexpensively partially enforced our relaxed constraint set.

Side constraint 2

In relaxation problem (Pũ), side constraint 1 can be said to provide a 1-degree higher bound quality than (Pu). That is, the k selected variables can exclude at most k (one for each) legitimate variables from the solution space. It is evident that by excluding more legitimate variables from the solution space a stronger bound may result. This is because the choice of variables to become a part of the solution will be restricted to free variables with higher costs. Now suppose that the k candidate variables are

$$y_{[1]}, y_{[2]}, \ldots, y_{[i]}, \ldots, y_{[j]}, \ldots, y_{[l]}, \ldots, y_{[k]}$$

in an increasing order of the reduced costs. Suppose that the arrangements $y_{[i]}$, $y_{[j]}$, and $y_{[l]}$ have at least one exam block in common (i.e., they are pairwise conflicting). By excluding $y_{[j]}$ and $y_{[l]}$ (due to their larger reduced costs) we must include two new variables $y_{[k+1]}$ and $y_{[k+2]}$ with reduced costs

$$(c-u)_{[k+1]} + (c-u)_{[k+2]} > (c-u)_{[j]} + (c-u)_{[l]}.$$

To present the idea pictorially, consider a graph in which the nodes represent the variables and the edges represent the variables in conflict. Then a complete subgraph represents a set of variables which are all in conflict. For such a complete subgraph we select the lowest reduced cost node (variable) and discard the rest of the nodes (variables). Next, we include as many variables as discarded, starting with the $(k+1)$st variable among the set of free variables for further consideration. As before, those variables underlined will not be included again to ensure a valid lower bound. This process is repeated until *all* needed variables have been selected.

We now present the overall branch and bound algorithm which employs our enhanced Lagrangian relaxation. First note that the constraint matrix of (P) possesses a 'staircase' structure with the last row containing all ones (see Figure 1). The columns of this matrix are partitioned into t nonempty subsets ('blocks') B_j, $j \in R = \{1, 2, \ldots, t\}$ such that B_j satisfies $a_{jk} = 1$ for all $k \in B_j$ and $a_{jk} = 0$ for $k \in B_l$, $l > j$. Within each block the columns are ordered in increasing order of reduced costs obtained by solving the linear programming relaxation of (P).

Let S denote the index set of a partial solution, S^+ the variables in S that are fixed at 1, and Q be the set of rows (constraints) that are not satisfied by S. Further, let Z^* be the value of the best solution found so far. Then, the algorithm proceeds as follows.

Step 1: Initialization

Set up the initial tableau. Let $S = \emptyset$, $S^+ = \emptyset$, $Q = \{1, 2, \ldots, \mathrm{NP}+1\}$, $k = 0$ and $Z^* = \mathrm{Zu}$, where

$$\begin{array}{ccccc}
1\,1\cdots 1 & & & & \\
 & 1\,1\cdots 1 & & \mathit{All}\ 0 & \\
0\ \mathit{or}\ 1 & & 1\,1\cdots 1 & & \\
 & & & \cdots & \\
1\,1\cdots & \cdots & \cdots & \cdots & 1\,1\,1
\end{array}$$

Figure 1. Structure of problem (P) constraint matrix

Zu may be obtained using any heuristic (see Appendix 2). Solve $(\bar{P})$, the linear relaxation of (P). If the solution is integral stop. Otherwise, record the values of the dual variables u_i and the reduced costs $c_j - z_j$ where $z_j = \sum_{i=1}^{NP+1} a_{ij} u_i$. Sort the reduced costs within each block in an increasing order and continue with Step 2.

Step 2: Branching

(2a) Let r be the index of first block in which all variables are free (i.e., $B_r \cap S = \emptyset$). If there is none, go to Step 2c.

(2b) Find the smallest $z_j - c_j$ in $B_r \cap \bar{S}$ (i.e. among the free variables in B_r), break ties arbitrarily. Let $y_j = 1$, redefine S^+ as $S^+ \cup \{j\}$, and S as $S \cup \{j\}$. Let $y_i = 0$ for all $i \in B_r$, $i \notin S$, and update Q as $Q \setminus \{i \mid a_{ij} = 1\}$. Let $l = 0$ and $k = k + 1$, then go to Step 3.

(2c) Let r be the index of the last block in which there is at least one free variable, i.e.

$$r = \max\{i \mid i \in R,\ B_i \cap S \neq \emptyset\},$$

go to Step 2b.

Step 3: Fathom by bound

Compute a lower bound $v(\text{Pu})$ for Pu. If $v(\text{Pu}) > Z^*$ or if (P) is infeasible then fathom the partial solution and go to Step 5. If $v(\text{Pu}) < Z^*$ go to Step 4.

Step 4: Test for an incumbent solution

If $Q = \emptyset$, a better solution has been found. Record this solution as the incumbent and update Z^*, then go to Step 5. Otherwise, go to step 6.

Step 5: Backtrack

If $S^+ = \emptyset$, terminate with the best solution found. Otherwise, let p be the last index included in S^+. If p is the last entry in S, go to Step 5a, else go to Step 5b.

(5a) Set $x_p = 0$, delete p from S^+ and update Q as $Q \cup \{i \mid a_{ip} = 1\}$. Go to step 3.

(5b) Delete all of the entries to the right of p in S (i.e. the indices of variables set to zero after x_p), go to Step 5a.

Step 6: Updating the multipliers

Let $l = l + 1$. If $l =$ MAXIT, go to Step 2, otherwise update the multipliers u_i as follows:

$$u_i^{l+1} = u_i^l + \frac{\lambda}{\| Ay - b \|^2}(Ay - b)$$

where $\lambda \in (0, 2)$ is a specified step size and MAXIT is the maximum number of iterations (updates) allowed. Then go to Step 3.

5. A bicriterion integer programming formulation

The cost coefficients c_j in the integer programming formulation (P) of Section 4, represent the number of students with *two or more* exams in the jth block arrangement. Consider a situation in which a decision maker prefers to place a different degree of importance on minimizing the number of students with three exams per day than on minimizing the number of students with two exams per day. In this case, the problem may be formulated as a Bicriteria Integer Programming (BIP) problem as follows:

$$\text{(BIP)} \quad \text{Min} \quad z_1 = \sum_{j=1}^{N} c_j' y_j, \tag{11}$$

$$\text{Min} \quad z_2 = \sum_{j=1}^{N} c_j'' y_j, \tag{12}$$

$$\text{s.t.} \quad (2)\text{–}(4),$$

where c_j' and c_j'' are the number of students with exactly two exams and three exams in the jth block combination, respectively.

The single-objective problem consists of optimizing one objective function z_1 or z_2 subject to (2)–(4). On the other hand, in the two-objective problem (BIP), instead of seeking a single optimal solution, a set of 'non-dominated' solutions must be sought. This set of nondominated solutions is a subset of the feasible region formed by constraints (2), (3), and (4). The first step in solving (BIP) consists of identifying a set of non-dominated solutions within the feasible region. There are several methods to generate a set of non-dominated solutions and one of them is the weighting method. The weighting method transforms the two-objective problem (BIP) into a single-objective problem as follows:

$$\text{(WBIP)} \quad \text{Min} \quad z = w_1 \sum_{j=1}^{N} c_j' y_j + w_2 \sum_{j=1}^{N} c_j'' y_j,$$

$$\text{s.t.} \quad (2)\text{–}(4). \tag{13}$$

The weights w_1 and w_2 on the objective functions z_1 and z_2 can be interpreted as 'the relative weight or worth' of the objectives when compared to each other. When the weights are interpreted as such, the solution to (BIP) is equivalent to the best-compromise solution relative to a particular preference structure.

Now, suppose $w_1 = 0$ (or $w_2 = 0$) and $w_2 = 1$ (or $w_1 = 1$), then (WBIP) reduces to a single-objective function covering type problem (P) discussed in Section 4 which can be solved using the branch and bound algorithm presented earlier. For any set of non-zero weights w's, a non-dominated solution for (BIP) will be generated (not all non-dominated solutions can be generated in this manner.) To explore a potentially large set of non-dominated solutions, a sensitivity analysis with respect to different weights at every node of the branch and bound is required. In this research, the branch-and-bound algorithm discussed in Section 4 with different sets of weights is used to locate a subset of non-dominated solutions. Depending on the choice of the administration, a preferred solution is selected. For example, when comparing two solutions, say $\binom{z_1}{z_2} = \binom{100}{10}$ versus $\binom{z_1}{z_2} = \binom{170}{3}$, the second solution is preferable to the first solution if the reduction of 7 students with 3 exams in a day is considered more important than an increase of 70 students with 2 exams in a day.

6. Computational experience

The proposed algorithm of Section 4 was programmed in FORTRAN V on a CDC Cyber 173/730 Dual System. In developing the code, we included several flag variables which enabled us to solve the same set of problems using various options. These options included using the first side constraint, the second side constraint, and adjusting the Lagrangian multipliers at certain nodes of the branch and bound tree.

We began by investigating the relative efficiency of implementing the side constraints in the computation of lower bounds. For this purpose we solved three problem sets, each containing three problems. Table 1 presents the characteristics of these test problems.

Table 1
Problem characteristics

Problem set	NP	NPD	ND	No. of variables	No. of constraints
1 [a]	15	3	5	455	16
2	18	3	6	816	19
3	21	3	7	1330	22

[a] Each set consists of three randomly generated problems with fully dense conflict matrices.

The first option (Option 1) consisted of solving all three problem sets without implementing any side constraints. The second option (Option 2) consisted of using the first side constraint and as a third option (Option 3) we used the second side constraint in the computation of the bound. In all of the above options the Lagrangian multipliers were initially obtained from the solution of the LP-relaxation of (P) with inequalities in constraint set (2) (see Appendix 1). These multipliers were kept fixed throughout the solution process.

The computational results for Options 1–3 are presented in Tables 2–4. In each table, the first column denotes the Option number, column two and three show the execution times (in seconds) for solving the problem (Total time) and computation of the bound (bound time), respectively. Columns four and five represent total number of nodes (Total nodes) and nodes which were fathomed due to the bound (Fathomed nodes).

As seen from Tables 2–4, the inclusion of the side constraints in the computation of lower bounds results in improving the performance of the algorithm. Although these extra steps require additional computation time per node, the improvements in the bound quality and consequently reduction in the number of nodes justifies these means. As an example, when comparing the results of Options 1 and 2 in problem set 3 (see Table 4), the inclusion of the side constraint increases the execution time for bound computation from 43.9 seconds (for 8 151 nodes) to 44.2 seconds (for 5 554 nodes). These figures are equivalent to execution times of 5.4 milliseconds and 8.0 milliseconds per node, or an increase of about 50 percent. However, since by using side constraint one the quality of bound improves significantly, the total number of nodes processed decreases by about 32 percent (from 8 151 to 5 554). Consequently, the total time decreases from an average of 298.3 seconds to 233.7 seconds.

The implementation of the second side constraint has consequences similar to those of side constraint one. The execution time (per node) for

Table 2
Computational results for problem set 1 [a]

Option	Total time [b]	Bound time	Total nodes
1	19.9	2.5	1274
2	16.3	2.6	880
3	13.8	3.1	713

[a] Entries represent averages for the three problems in the set.
[b] Times are in seconds.

Table 3
Computational results for problem set 2 [a]

Option	Total time [b]	Bound time	Total nodes
1	83.2	18.1	3557
2	73.7	18.2	2822
3	63.7	19.7	2268

[a] Entries represent averages for the three problems in the set.
[b] Times are in seconds.

Table 4
Computational results for problem set 3 [a]

Option	Total time [b]	Bound time	Total nodes
1	298.3	43.9	8151
2	233.7	44.2	5554
3	190.2	46.4	4273

[a] Entries represent averages for the three problems in the set.
[b] Times are in seconds.

computation of bound increases but the improvement in the quality of the bound and reduction of processed nodes results in a decrease in the overall execution time. For example, in problem set 3, the execution time for bound (per node) increases to about 10.8 milliseconds (from 8.0 milliseconds in Option 2). However, the total number of nodes processed decreases to 4273 and consequently total execution time decreases to about 200 seconds (averaged over three problems).

Based on the above observations we implemented the second side constraint in bound computation for the remainder of the testing. We decided to further improve the performance of the algorithm by updating the Lagrangian multipliers at nodes whose bounds indicated a potential for fathoming. As before, the first set of multipliers were obtained by solving the LP-relaxation. Then, at each node, where the lower bound was within 95% of the value of the current incumbent, a subgradient search procedure was applied to update the multipliers to try to improve the bound. The number of iterations per subgradient search was limited to 12. However, when the improvement in the bound was less than 25% for every three iterations, the subgradient search was terminated early (fewer than 12 iterations). The value of $\lambda = 2$ was found to perform well.

It is worth mentioning that the changes in multipliers each time a subgradient search is applied does not require the complete resorting of all cost coefficients. Once the coefficients are sorted at node zero, adjusting multipliers does not greatly affect the positions of cost coefficients in the ordered list. Thus, a partial searching for the least cost variables in the early segment of the list is done first. That is, if the least cost variable is located in the first k (ex. $k = 100$) elements then, the second least cost variable is highly probable to be also located in the first k elements. If not, the larger remaining segment of the list is searched once to locate the second least cost (our implementation also divides the remaining segment into three blocks). This process of finding the required number of ordered least cost variables continues until (Pu) is solved. It should be mentioned that a complete sorting procedure would render the subgradient of little value. Sorting 1330 elements using an efficient sorting method takes approximately 0.3 seconds of CPU time. Thus, 12 iterations of subgradient search at every node would cost 3.6 seconds.

The performance of the algorithm of section 4 was evaluated by using two sets of test problems. The first set consisted of the three randomly generated problems of Set 3 and the second set consisted of two real world problems obtained from the scheduling office at SUNYAB. The SUNYAB data contained 981 final exams for Spring 1984 and 766 final exams for Spring 1986. There were an average of 27600 students enrolled in these courses. The SUNYAB final exams were first assigned to exam blocks (Phase I) by using a heuristic procedure due to Arani [1]. Table 5 presents the results of this phase.

The resulting blocks from Phase I were used to determine the various parameters of (P) for SUNYAB data. As mentioned above, we applied the algorithm of Section 4 to solve problems of Set 3 and those of SUNYAB (see Table 6). By selectively performing subgradient search, only at nodes whose bounds were within 5% of being fathomed,

Table 5
SUNYAB direct conflicts

Number of blocks	Spring 1984	Spring 1985
15 [a]	118	50
18	38	6
21	1	0

[a] Entries indicate number of students with more than one exam per block.

Table 6
Computational result of the subgradient search

Problem	LP time [c]	Total time	Bound time	Total nodes	No. of relaxations
Set 3 [a]	6.0	49.7	30.7	714	1.7
SUNYAB [b]	5.9	32.9	31.3	174	2.8

[a] Entries represent averages for the three problems in the set.
[b] Entries represent averages for the two problems.
[c] Times are in seconds.

the execution time was reduced by about 34% and the number of nodes by about 83%.

Results of the bicriteria approach

In order to have a basis for comparison, we solved the SUNYAB problems in several ways (each time using 15, 18, and 21 blocks). First, we used a naive approach (SEQNTL) of assigning the NP exam blocks sequentially to exam periods. That is, exam block 1 was assigned to the first period of the first day and so on. Secondly, we formulated the problem as a traveling salesman problem (TSP) (see Section 2) and solved it using the Held and Karp [12] algorithm. Thirdly, we utilized the heuristic (HURST) procedure (see Appendix 2) which was designed to obtain an upper bound for the algorithm of Section 4. Finally, we let $w_1 = 1$ and $w_2 = 3$ and solved the stated problems using the BIP formulation. This particular set of weights were selected because they would result in the same cost coefficients as the other three options. Tables 7 and 8 show the results for 15 exam blocks.

As seen in Tables 7 and 8, (SEQNTL) consistently performs poorly when compared with other methods. (TSP) performs well on the total consecutiveness dimension but is worse than (HURST) and (BIP) on the 2 Exams and 3 Exams dimensions. Finally, in terms of total consecutiveness, (BIP) outperforms (HURST) but on 2 Exams and 3 Exams the results are not conclusive. These results were consistent when 18 and 21 blocks were used. To further investigate the performance of (BIP), we solved the Spring 1985 problem with 15 Exam blocks. This time we varied w_1 and w_2 in such a way to find the extreme values for 2 Exams and 3 Exams dimensions. Table 9 presents the results of (BIP) with different weights.

Table 7
Computational results for minimizing second degree conflicts (15 exam blocks, spring 1984)

Solution method	2 Exams	3 Exams	Total consecutive	Consecutive within days	Consecutive between days	Total time
SEQNTL	15154	587	13291	10357	2934	0.07
TSP	11394	209	8756	5261	3495	0.76
HURST	10588	181	9542	5819	3723	0.11
BIP	10314	187	9396	5674	3722	2.26

Table 8
Computational results for minimizing second degree conflicts (15 exam blocks, spring 1985)

Solution method	2 Exams	3 Exams	Total consecutive	Consecutive within days	Consecutive between days	Total time
SEQNTL	12825	758	11085	0979	2005	0.075
TSP	9354	239	6922	4263	2659	0.79
HURST	8610	149	7507	4704	2803	0.119
BIP	8399	146	7560	4518	3042	18.23

Table 9
Computational results for minimizing second degree conflicts (15 exam blocks, spring 1985)

2 Exams	3 Exams	Total consecutive	Consecutive within days	Consecutive between days	w_2	w_1	Total time
9134	122	7972	4531	3441	0	1	0.12
8550	128	7513	4606	2907	1	30	37.9
8442	133	7601	4513	3088	1	20	29.02
8442	133	7570	4513	3057	1	10	5.65
8399	149	7798	4518	3280	1	0	9.93
8399	149	7798	4518	3280	10	9	8.83
8399	149	7798	4518	3280	20	9	9.21

Appendix 1. Linear programming relaxation

In the first step of the branch and bound algorithm discussed in Section 4, linear relaxation of the problem (P) is solved using the revised simplex method. We were able to develop an efficient code by utilizing the special structure of the problem considering inequalities in constraint set (2). To begin, we constructed an *initial basis* using ND structural variables y_j's and (NP + 1 − ND) surplus variables. In order for these variables to form a feasible basis two conditions had to be satisfied. First, the structural variables should correspond to nonoverlapping block combinations (i.e., exam block had to appear exactly once among all block combinations (y_j's) in the basis. Secondly, the set of structural variables together with surplus variables should be linearly independent.

One alternative for variable selection is to use the first ND structural variables and complete the set by using the first (NP + 1 − ND) surplus variables. This alternative however will result in some of the rows of the augmented basis matrix being linearly dependent. For example, suppose NP = 9, ND = 3, and NPD = 3. If we use the first three structural variables and complete the set by using the first seven surplus variables, the augmented matrix will be singular. To form a proper initial basis one can start with the first three structured variables. Then, choose two surplus column by choosing an arbitrary row among those for which a surplus variable was not selected yet.

Another proper choice of structural variables correspond to a set in which each configuration consists of the ND + 1 st block. For example, the set $y_i = \{1, 4, 7\}$, $y_j = \{2, 5, 8\}$ and $y_k = \{3, 6, 9\}$ form a proper set. Along with the first seven surplus variables, these structural variables from a proper initial basis. We implemented this latter idea in our algorithm.

Other steps of the simplex method include computation of the reduced costs $z_j - c_j$ and test for a minimum ratio. The latter step requires computation of $B^{-1}a_j$ and $B^{-1}b$. Due to the special structure of the constraint matrix A, these computations may be reduced to just additions of various elements of B^{-1} and $w = c_B B^{-1}$.

As mentioned earlier, the elements of A are all zeros and ones. Further, each column of A consists of exactly NPD 1's in fixed positions and one 1 in the last row. As a result, the A-matrix can be stored in a 2-dimensional array LOCAT(r, s), $r = 1, 2, \ldots, \text{NPD}$, $s = 1, 2, \ldots, N$. Then, the (r, s) element of LOCAT represents the position (row number) of the r-th 1 in the s-th column of A. Note that since the last row of A consists of all 1's it need not be stored.

The computation of $B^{-1}a_j$ and $z_j = wa_j$ now simplifies to the following two equations:

$$B^{-1}a_j(k) = \sum_{r=1}^{\text{NPD}} B^{-1}(k, \text{LOCAT}(r, j)) + B^{-1}(k, \text{NP}+1),$$

$$k = 1, 2, \ldots, \text{NP}+1,$$

and

$$z_j = \sum_{r=1}^{\text{NPD}} w(\text{LOCAT}(r, j)) + w(\text{NP}+1).$$

The above approach resulted in a very efficient solution of the LP-relaxation of problem (P) (see Section 6 for computational results).

Appendix 2. A heuristic to compute the initial upper bound Zu

The branch and bound algorithm presented in Section 4 requires the usual initial upper bound Zu. We developed a heuristic procedure to solve the problem suboptimally and obtain an upper bound.

To begin, NP exam blocks are to be grouped into ND days with NPD exam blocks per day. Suppose that ND(NPD − 1) blocks have been, arbitrarily, assigned to the first (NPD − 1) periods of every day and we are about to assign the last ND blocks to the last period of each day. The optimal arrangement of the last ND blocks among the remaining ND last periods, such that the total additional conflict (beyond those of NPD-1 periods of each day) is minimized, is the usual assignment problem. Once an optimal solution to this assignment problem is found, we may fix the blocks assigned to the last periods. Note that the arbitrary assignment of blocks to the first NPD − 1 periods in conjunction with the optimal assignment of the last periods results in a suboptimal solution for the overall problem and consequently an upper bound.

To continue reducing the total conflict within each day, we backtrack to the NPD-1st periods and try to rearrange the ND blocks (assigned arbitrarily). As with the last periods, we keep the remaining ND (NPD − 1) blocks fixed and solve the associated assignment problem. This backtracking process continues until the assignment problem for the first periods is solved. Then a forward move is attempted by solving an assignment problem for the second period. The process of rearranging ND blocks at a time is repeated, going back and forth between the first to last periods, until no further reduction is possible. At this point the heuristic terminates and the last solution is used as the initial incumbent for the branch and bound method.

Computational results for three sets of problems are presented in Table 10.

Table 10
Computational results for heuristic procedure

Problem set	Problem number	Heuristic soln. value	Optimal soln. value	%Difference
	1	1276	1152	9.7
1	2	1121	902	19.5
	3	1666	1523	8.5
	1	642	436	32
2	2	668	406	39.2
	3	612	386	36.9
	1	472	380	19.4
3	2	491	395	19.5
	3	946	750	20

References

[1] Arani, T., "A multi-phase final examination scheduling problem", Doctoral Dissertation, SUNYAB, 1986.

[2] Balas, E., "An additive algorithm for solving linear programs with zero–one variables", *Operations Research* 13 (1965) 517–546.

[3] Brelaz, D., "New method to color the vertices of a graph", *Communication of the ACM* 22 (1979) 251–256.

[4] Broder, S., "Final examination scheduling", *Communication of the ACM* 7 (1964) 494–498.

[5] Carter, M.W., "A decomposition algorithm for practical timetabling problems", *University of Toronto, Dept. of Industrial Engineering*, Working Paper No. 83-86, 1983.

[6] Carlson, R., and Nemhauser, G., "Scheduling to minimize interaction cost", *Operations Research* 14 (1967) 52–58.

[7] Coljin, A.W., "Reduction of second order conflict in examination timetables", as cited in White and Chan (1979).

[8] Dunston, F.D.J., "Sequential colorings of graphs", *Proc. 5th British Comb. Conf., Aberdeen*, 1975, *Cong. Num. XV, Utilitas Math.* (1976) 151–158.

[9] Desroches, S., Laporte, G., and Rousseau, J.M., "Horex: A computer program for the construction of examination schedules", *INFOR* 16 (1978) 294–298.

[10] Grimes, J., "Scheduling to reduce conflict in meetings", *Communications of the ACM* 13 (1970) 351–352.

[11] Hall, A., and Acton, F., "Scheduling university course examinations by computers", *Communications of the ACM* 10 (1967) 235–238.

[12] Held, K., and Karp, R.M., "The traveling salesman problem and minimum spanning trees", *Operations Research* 18 (1970) 1138–1162.

[13] Korman, S.M., "The graph-coloring problem", in: *Combinational Optimization*, Wiley, New York, 1979.

[14] Leighton, F., "A graph coloring algorithm for large scheduling problems", *Journal of Research of the National Bureau of Standard* 84 (1979) 489–506.

[15] Lotfi, V., and Sarin, A., "A graph coloring algorithm for large scale scheduling problems", CORS, in print.

[16] Matula, D.W., Marble, G., and Isaacon, I.B., "Graph coloring algorithms", in: *Graph Theory and Computing*, Academic Press, New York, 1972.

[17] Michaud, P., "Exact implicit enumeration method for

solving the set participating problem", *IBM Journal of Research and Development* 16 (1972) 573–578.

[18] Peck, J.E., and Williams, M.R., "Algorithm 286 examination scheduling", *Communications of the ACM* 9 (1969) 433–434.

[19] Pierce, J.F., "Application of combinatorial programming to a class of all zero–one integer programming problems", *Management Science* 15 (1968) 191–209.

[20] Pierce, J.F., and Lasky, J.S., "Improved combinatorial programming algorithms for a class of all zero–one integer programming problems", *Management Science* 19 (1973) 528–543.

[21] White, G., and Chan, P., "Towards the construction of optimal examination schedules", *Canadian Journal of Operational Research and Information Processing* 17 (1979) 219–229.