AUTOMATED SCHEDULE GENERATION USING BINARY INTEGER PROGRAMMING PROBLEMS

ISAAC J. LEE AND BRIAN M. MCCARTHY

ABSTRACT. We consider the generalized assignment problem (GAP) of assigning m tutors to cover n sessions. Each tutor specifies which courses he or she is qualified to cover, how many hours per week he or she would spend on a session, and the maximum number of hours for which he or she can work per week. If there are not enough man hours to cover all sessions, we ensure that the sessions with a high priority level are covered.

Keywords and Phrases: generalized assignment problem, timetabling problem, binary integer programming, Balas' additive algorithm

1. INTRODUCTION

It is often useful to have an automated way of distributing tasks among a set of workers, or "agents," while minimizing cost. This sort of problem, dealing with tasks and agents, is called an assignment problem. The classic assignment problem involves assigning a single task to each agent, and no more than one agent per task, so that there is a one-to-one match between the two sets [4]. While this can be useful for a number of applications, such as assigning factory workers to machines, a great number of additional applications may be uncovered by slightly changing the problem. In this paper, we will be talking about assignment problems in which there may be multiple tasks assigned to a single agent, but no more than one agent for a given task. This is known as the generalized assignment problem (GAP) [4]. For the remainder of the paper, agents will be referred to as tutors, and tasks will be known as sessions.

GAPs are encountered in many areas, often in relation to the creation of schedules. Now, we need to make a yes-or-no decision for whether a tutor is assigned to a particular session, so we will set up the problem as a binary integer programming (BIP) problem, rather than a linear programming (LP) problem. The examples discussed in this paper were test cases for an online scheduling program that will be used by the University of Kentucky's Student Support Services to assign tutors to sessions. When implementing an algorithm for solving IP problems in a web-based application, it is important to choose one that can handle large data sets with a relatively fast computation time. We considered three different algorithms.

The first algorithm to consider was the branch-and-bound method. This method, which is recursive in nature, "performs very badly in the worst case" [3]. Another option was a cutting-plane method, which attempts to progressively reduce the difference between the relaxed LP solution and the desired IP optimal solution [3]. The one we ended up using is Lau's implementation of Balas' additive algorithm. This algorithm, which is tailored for BIP problems, only uses the addition and subtraction operations and has no round-off errors [1]. It starts by setting all variable values to zero, and then setting some of the values to one and checking the outcome [1]. The program is fully functioning and has been extensively tested.

2. The Problem

We use the following notation:

- I, the set of tutors $(i = 1, \cdots, m)$
- J, the set of sessions $(j = 1, \cdots, n)$
- x_{ij}, equals 1 if tutor i is assigned to session j and 0 otherwise
- q_{ij}, equals 1 if tutor i is qualified for session j and 0 otherwise
- a_{ij}, the number of hours that tutor i would spend on session j per week
- b_i, the maximum number of hours for which tutor i can work per week
- c_{ij}, the priority level of tutor i's covering session j (lower c_{ij} corresponds to higher priority level)

We seek to solve the following binary integer programming problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_i \sum_j c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_i q_{ij} x_{ij} = 1, \ \forall\, j \in J \\
& \sum_j a_{ij} x_{ij} \leq b_i, \ \forall\, i \in I \\
& x_{ij} = 0 \text{ or } 1
\end{aligned}$$

The first set of constraints $\sum_i q_{ij} x_{ij} = 1, \forall\, j \in J$ ensures that only one qualified tutor is assigned to a session. The second set $\sum_j a_{ij} x_{ij} \leq b_i, \forall\, i \in I$ ensures that the total number of hours for which a tutor would be working does not exceed the maximum number of hours that he or she can work per week.

Now, Lau's implementation requires the following formulation:

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ \forall\, i = 1, \cdots, m \\
& x_j = 0 \text{ or } 1, \ \forall\, j = 1, \cdots, n
\end{aligned}$$

Therefore, we rewrite our problem by considering the matrix A whose elements consist of q_{ij}'s and a_{ij}'s, and the vectors $\vec{x} = [x_{11}, \cdots, x_{1n}, \cdots, x_{m1}, \cdots, x_{mn}]^T$, $\vec{b} = [1, -1, \cdots, 1, -1, b_1, \cdots, b_m]^T$, and $\vec{c} = [c_{11}, \cdots, c_{1n}, \cdots, c_{m1}, \cdots, c_{mn}]^T$:

$$\begin{aligned}
\text{minimize} \quad & \vec{c}^T \vec{x} \\
\text{subject to} \quad & A\vec{x} \leq \vec{b}, \ \vec{x} \in \{0, 1\}^{mn}
\end{aligned}$$

3. Examples

The following three examples will illustrate various aspects of the problem.

Example 1. Consider the following scenario with $m = 8$ tutors (A - H) and $n = 15$ sessions (1 - 15):

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A (1)	✓	✓	✓				✓								
B (3)	✓	✓	✓	✓	✓	✓									
C (3)	✓	✓	✓	✓	✓		✓								
D (1)								✓	✓						
E (2)								✓	✓						
F (2)										✓	✓	✓			
G (4)											✓	✓	✓		
H (3)										✓				✓	✓

A checkmark indicates that tutor i is qualified for session j, i.e. $q_{ij} = 1$. The maximum number of sessions that a tutor can handle is indicated next to the letter in parentheses. (Note that these two data sets would be obtained from input on the website.) For convenience, we will assume that each session is to be held for two hours per week and has a priority level of 3 units for any tutor, i.e. $a_{ij} = 2$, $b_i = 2 \cdot$ (the maximum number of sessions that tutor i can handle), and $c_{ij} = 3$.

We solve the following problem:

$$
\begin{aligned}
\text{minimize} \quad & 3 \sum_i \sum_j x_{ij} \\
\text{subject to} \quad & x_{1,1} + x_{2,1} + x_{3,1} \leq 1 \\
& - x_{1,1} - x_{2,1} - x_{3,1} \leq -1 \\
& x_{1,2} + x_{2,2} + x_{3,2} \leq 1 \\
& - x_{1,2} - x_{2,2} - x_{3,2} \leq -1 \\
& \vdots \\
& x_{8,14} \leq 1 \\
& - x_{8,14} \leq -1 \\
& x_{8,15} \leq 1 \\
& - x_{8,15} \leq -1 \\
& 2x_{1,1} + 2x_{1,2} + 2x_{1,3} + 2x_{1,7} \leq 2 \\
& 2x_{2,1} + 2x_{2,2} + 2x_{2,3} + 2x_{2,4} + 2x_{2,5} + 2x_{2,6} \leq 6 \\
& \vdots \\
& 2x_{7,11} + 2x_{7,12} + 2x_{7,13} + \leq 8 \\
& 2x_{8,10} + 2x_{8,14} + 2x_{8,15} \leq 6 \\
& x_{ij} \in \{0,\, 1\}
\end{aligned}
$$

Here is the solution given by the program:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A (1)	✓	✓	✓				**✓**								
B (3)	✓	✓	✓	**✓**	**✓**	**✓**									
C (3)	**✓**	**✓**	**✓**	✓	✓		✓								
D (1)								✓	**✓**						
E (2)								**✓**	✓						
F (2)										✓	**✓**	**✓**			
G (4)											✓	✓	**✓**		
H (3)										**✓**				**✓**	**✓**

A dark green-shaded cell indicates that tutor i has been assigned to session j, i.e. $x_{ij} = 1$. We see that all 15 sessions are covered.

Example 2. We reconsider the above example with b_i's such that there aren't enough man hours to cover all sessions.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A (1)	✓	✓	✓				✓								
B (2)	✓	✓	✓	✓	✓	✓									
C (1)	✓	✓	✓	✓	✓		✓								
D (1)								✓	✓						
E (2)								✓	✓						
F (2)										✓	✓	✓			
G (2)											✓	✓	✓		
H (1)										✓				✓	✓

From the above figure, we easily see that at most 12 sessions can be covered by the tutors. Lau's implementation, on its own, actually halts the procedure with an apologetic message that there is no feasible solution. We, however, seek to avoid this problem. So we will ensure that there is always a feasible solution that covers sessions with a high priority level by introducing a "ghost."

We introduce an additional tutor, who is unearthly and thus aptly named the ghost, into I. This ghost, who is qualified for all sessions and has all the time in the world to devote to tutoring (as he is dead), would take up all sessions that could previously not be covered. Now, we want to assign the sessions to those who are corporeal if at all possible, so we assign a ghastly large number to $c_{m+1,j}$, say 100.

To take account for the ghost, we need to append n variables $x_{m+1,1}, \cdots, x_{m+1,n}$ to the existing $m + 2n$ inequalities, but because $b_{m+1} = \infty$, we do not require an additional inequality $\sum_j a_{m+1,j}\, x_{m+1,j} \le b_{m+1}$. Therefore, A now has dimensions $(m + 2n) \times (m + 1)n$.

So given any assignment problem, we solve it with the following procedure:

```
Initialize A_{(m+2n)×(m+1)n}, b_{m+2n}, and c_{(m+1)n} to all zeroes.
Set A(1 : m + 2n, 1 : mn), b(1 : m + 2n), and c(1 : mn) according to the problem
solve()
IF the solution is not feasible
        Set A(1 : m + 2n, mn + 1 : (m + 1)n) and c(mn + 1 : (m + 1)n) for the ghost
        solve()
END
```

We solve the following problem:

$$
\begin{aligned}
\text{minimize} \quad & 3 \sum_{i \in \{1,\cdots,8\}} \sum_j x_{ij} + 100 \sum_j x_{9,j} \\
\text{subject to} \quad & x_{1,1} + x_{2,1} + x_{3,1} + x_{9,1} \leq 1 \\
& - x_{1,1} - x_{2,1} - x_{3,1} - x_{9,1} \leq -1 \\
& x_{1,2} + x_{2,2} + x_{3,2} + x_{9,2} \leq 1 \\
& - x_{1,2} - x_{2,2} - x_{3,2} - x_{9,2} \leq -1 \\
& \vdots \\
& x_{8,14} + x_{9,14} \leq 1 \\
& - x_{8,14} - x_{9,14} \leq -1 \\
& x_{8,15} + x_{9,15} \leq 1 \\
& - x_{8,15} - x_{9,15} \leq -1 \\
& 2x_{1,1} + 2x_{1,2} + 2x_{1,3} + 2x_{1,7} \leq 2 \\
& 2x_{2,1} + 2x_{2,2} + 2x_{2,3} + 2x_{2,4} + 2x_{2,5} + 2x_{2,6} \leq 4 \\
& \vdots \\
& 2x_{7,11} + 2x_{7,12} + 2x_{7,13} + \leq 4 \\
& 2x_{8,10} + 2x_{8,14} + 2x_{8,15} \leq 2 \\
& x_{ij} \in \{0, 1\}
\end{aligned}
$$

Here is the solution given by the program:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A (1)	**✓**	✓	✓				✓								
B (2)	✓	**✓**	✓	✓	✓	**✓**									
C (1)	✓	✓	✓	✓	✓		**✓**								
D (1)								**✓**	✓						
E (2)								✓	**✓**						
F (2)										**✓**	**✓**	✓			
G (2)											✓	**✓**	**✓**		
H (1)										✓				✓	**✓**
Ghost (∞)	✓	✓	**✓**	**✓**	**✓**	✓	✓	✓	✓	✓	✓	✓	✓	**✓**	✓

Example 3. We investigate how fast the program produces a solution by considering a problem with $m = 20$ tutors (A - T) and $n = 30$ sessions (1 - 30). (See the next page for the problem and its solution.) Note that an exhaustive search would require $2^{600} \approx 4.15 \times 10^{180}$ considerations at its worst case, each of which requires ensuring that the 30 equalities and 20 inequalities are satisfied. If we are unlucky enough that we do need the ghost, we would have to repeat the search.

Fortunately, our trials showed that the program found a feasible solution in one or two seconds while running on an average computer. This was an extremely good news, as it is important for the website to display the solution in a reasonable time.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
A (2)	**✓**	**✓**	✓	✓	✓																									
B (3)	✓	✓			**✓**	**✓**		**✓**																						
C (4)		✓	**✓**	**✓**			**✓**	✓				✓	✓																	
D (1)				✓						**✓**																				
E (2)							✓		✓		**✓**	✓																		
F (2)						✓								**✓**	✓		✓	✓												
G (3)					✓			✓	✓				✓	✓																
H (2)			✓							✓					**✓**	**✓**														
I (1)											✓																			
J (3)									**✓**								**✓**		✓	**✓**	✓									
K (1)																		✓	**✓**											
L (1)												**✓**										✓		✓						
M (2)												✓										**✓**		**✓**						
N (2)																					**✓**	✓			**✓**	✓				
O (1)										✓	✓		**✓**								✓									
P (2)																	✓	**✓**					**✓**							
Q (3)																					✓		✓	✓			**✓**			
R (2)																										**✓**		**✓**	✓	✓
S (2)																									✓			✓	**✓**	**✓**
T (1)																									✓		✓			

Example 3. $m = 20$, $n = 30$

4. Conclusion

Balas' additive algorithm has worked well overall for our efforts to create a schedule for tutors and tutoring sessions. There are a few areas in which the program can be improved, however, by making modifications to the constraints. For instance, the algorithm currently focuses on ensuring that all sessions (or as many of them as possible) have tutors rather than that all tutors are assigned to a session. In Example 3, we see that tutors G, I, and T were not assigned to any sessions. In order to fix this problem, we can introduce a new set of constraints:

$$\sum_j x_{ij} \geq 1, \, \forall \, i \in I$$

This would be an improvement because tutors are paid employees of the tutoring organization, so it would be a waste of resources for that organization not to give work to some of its employees.

Also, it would help to ensure that the tutors are assigned a session load proportional to the tutor's availability. In Example 1, tutor E requested two sessions and was assigned one, whereas tutor G requested four sessions and was assigned one as well. This is not a proportional distribution of session load for the number of sessions that each of the tutors can handle.

Another possible feature could be is to assign a "group" of sessions to a tutor. In Example 2, we easily see that tutor B could have been assigned to any one of sessions 3 to 5 instead of session 2. Suppose both sessions 5 and 6 were of a calculus course, while session 2 was of a history course. Tutor B may prefer to take on sessions 5 and 6 instead, as it requires less effort to prepare for two sessions of the same course. The system should, when there are multiple sessions for a particular course, show a preference for assigning the same tutor to do those sessions rather than assigning different tutors to those sessions.

Despite these opportunities for improvement, the system is more than sufficient for use in its intended application. The main obstacle in Balas' additive algorithm was the possibility of producing no feasible solution [1]. In a program where the problem is supposed to be solved automatically at the press of a button, this could have proved to be a fatal flaw. The administrator of the scheduling program would have lacked the understanding of how to manually avoid this error. As demonstrated in Example 3, however, the introduction of the ghost allows the program to run smoothly without any need for oversight by a mathematician. This feature allows the program to be a success.

References

[1] E. Balas, An Additive Algorithm for Solving Linear Programs with Zero-One Variables, Operations Research **13** (1965), 517-546.

[2] H. T. Lau, A Java Library of Graph Algorithms and Optimization (Discrete Mathematics and Its Applications), Chapman & Hall/CRC, Boca Raton, 2007.

[3] J. Lee, A First Course in Combinatorial Optimization, Cambridge University Press, 2004.

[4] D. W. Pentico, Assignment Problems: A Golden Anniversary Survey, European Journal of Operations Research **176** (2007), 774-793.

[5] R. J. Vanderbei, Linear Programming: Foundations and Extensions, Springer, New York, 2008.

Appendix A. Code (integerProgram.java)

```java
import java.io.*;
import java.util.*;

public class integerProgram extends Object {
   /****************************************************************************
    *
    * AUTHORS: Isaac J. Lee, Brian M. McCarthy
    *
    ****************************************************************************/
   public static ArrayList initialize(){
      int m = 0, n = 0;                              // m = number of tutors, n = number of sessions
      ArrayList<int[][]> arrayList = new ArrayList<int[][]>();        // List of matrices to be returned

      String[] fileNames;                            // Names of files that contain tutors and sessions information
      // Give a first-read of the files to figure out how large m and n are
      File directory = new File("Schedules");                   // The files are to be located under the directory named Schedules (we know this has been created by tasks.sphp and thus exists)
      fileNames = directory.list();
      if (fileNames.length == 1) {
         System.out.println("ERROR: There are no tutor cards submitted.");
         System.exit(0);
      }
      // Find out how many tutors (m) and sessions (n) there are
      for (int i = 0; i < fileNames.length; i++) {
         if (fileNames[i].startsWith("Tutor"))               // Tutor<tutor id>.txt
           m++;
         else if (fileNames[i].startsWith("Courses"))           // Courses.txt
            try {
               BufferedReader reader = new BufferedReader(new FileReader("Schedules/" + fileNames[i]));
               while (reader.readLine() != null)
                  n++;                         // Each line contains one session id
               reader.close();
            } catch (FileNotFoundException e) {
               e.printStackTrace();
            } catch (IOException e) {
               e.printStackTrace();
            }
      }

      // Initialize the matrices to 0
      int a[][] = new int[m + 1][n + 1];                        // A
      int q[][]= new int[m + 1][n + 1];                         // Q
      int b[] = new int[m + 1];                          // b
      int c[] = new int[n + 1];                          // c
      int A[][] = new int[m + 2*n + 1][(m + 1)*n + 1];             // Large matrix A, to be returned
      int B[][] = new int[m + 2*n + 1][1];                      // Large vector B, to be returned
      int C[][] = new int[(m + 1)*n + 1][1];                    // Large vector C, to be returned
      int tutors[][] = new int[m + 1][1];                    // Store in which index of the array a tutor belongs, to be returned
      int sessions[][] = new int[n + 1][1];                     // Store in which index of the array a session belongs, to be returned

      // For right now, the default priority level for a session is 3
      for (int i = 0; i < n + 1; i++) {
         c[i] = 3;
      }
      // Form large vector C by assuming the priority levels among tutors for a session are equal
      for (int i = 0; i < m; i++)
         for (int j = 0; j < n; j++)
            C[i*n + j + 1][0] = c[j + 1];

      int countTutors = 0, countSessions = 0;
      for (int i = 0; i < fileNames.length; i++) {
         String fileName = fileNames[i];
         try {
            // Count how many lines are in the file
            BufferedReader initReader = new BufferedReader(new FileReader("Schedules/" + fileName));
            int numLines = 0;
            while (initReader.readLine() != null)
```

```
                numLines++;
            initReader.close();

            BufferedReader reader = new BufferedReader(new FileReader("Schedules/" + fileName));
            if (fileName.startsWith("Tutor")) {
                countTutors++;
                // The first line contains the tutor ID
                int tutorID = Integer.parseInt(reader.readLine());
                tutors[countTutors][0] = tutorID;
                // The second line contains the maximum number of hours for which the tutor can work
                b[countTutors] = Integer.parseInt(reader.readLine());
                // Read the remaining lines which indicate what courses the tutor can teach and for how many hours
                for (int j = 0; j < numLines - 2; j++){
                    String line = reader.readLine();
                    int index = line.indexOf(" ");
                    // The first word in the line is the session ID
                    int sessionID = Integer.parseInt(line.substring(0, index));
                    // The second word, separated by a comma, indicates for how many hours the tutor can teach
                    // Multiply the session hours by 4 (specified in quarters)
                    int sessionHours = (int)(4 * Double.parseDouble(line.substring(index + 1, line.length())));

                    // q_{ij} = 1 if the tutor is qualified to teach the session j
                    q[countTutors][sessionID] = 1;
                    a[countTutors][sessionID] = sessionHours;
                }
            } else if (fileName.startsWith("Courses")) {
                for (int j = 0; j < numLines; j++){
                    countSessions++;
                    sessions[countSessions][0] = Integer.parseInt(reader.readLine());
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // The first 2n lines of A ensure that only one qualified tutor is assigned to a session
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++){
            A[2*i + 1][j*n + i + 1] = q[j + 1][i + 1];
            A[2*i + 2][j*n + i + 1] = -q[j + 1][i + 1];
            B[2*i + 1][0] = 1;
            B[2*i + 2][0] = -1;
        }
    }
    // The last m lines of A ensure that the total number of hours for which a tutor would be working
    // does not exist the maximum number of hours that the tutor can work per week
    for (int i = 0; i < m; i++){
        for (int j = 0; j < n; j++){
            A[2*n + i + 1][i*n + j + 1] = a[i + 1][j + 1];
            B[2*n + i + 1][0] = 4 * b[i + 1];                // Multiply by 4 on the right side of the equations
        }
    }

    arrayList.add(A);
    arrayList.add(B);
    arrayList.add(C);
    arrayList.add(tutors);
    arrayList.add(sessions);

    return arrayList;
}

// SOURCE: A Java Library of Graph Algorithms and Optimization by H. T. Lau
public static void solveIntegerProgram(boolean minimize, int m, int n, int a[][], int b[], int c[], int sol[]) {
    int i, j, k, optvalue, elm1 = 0, elm2, elm3, elm4, idx, sub1, sub2, sub3;
    int item1, item2, item3;
    int ccopy[] = new int [n + 1];
    int aux1[] = new int [n + 1];
    int aux2[] = new int [n + 1];
```

```
      int aux3[] = new int [n + 1];
      int aux4[] = new int [n + 1];
      int aux5[] = new int [n + 1];
      int aux6[] = new int [n + 1];
      int aux7[] = new int [n + 1];
      boolean cminus[] = new boolean[n + 1];
      boolean optimalfound, backtrack = false, outer;

      // scan for the negative objective coefficients
      if (!minimize)
        for (j = 1; j <= n; j++)
          c[j] = -c[j];
      for (j = 1; j <= n; j++) {
        cminus[j] = false;
        ccopy[j] = c[j];
      }
      for (j = 1; j <= n; j++)
        if (c[j] < 0) {
          cminus[j] = true;
          c[j] = -c[j];
          for (i = 1; i <= m; i++) {
            b[i] -= a[i][j];
            a[i][j] = -a[i][j];
          }
        }

      for (i = 1; i <= m; i++)
        aux5[i] = b[i];
      elm4 = 1;
      for (j = 1; j <= n; j++) {
        aux3[j] = 0;
        elm4 += c[j];
      }
      optvalue = elm4 + elm4;
      sub2 = 0;
      sub3 = 0;
      elm4 = 0;
      aux4[1] = 0;
      optimalfound = false;
      iterate:
        while (true) {
          if (backtrack) {
            // backtracking
            backtrack = false;
            outer = false;
            for (j = 1; j <= n; j++)
              if (aux3[j] < 0) aux3[j] = 0;
            if (sub2 > 0)
              do {
                sub1 = sub3;
                sub3 -= aux4[sub2 + 1];
                for (j = sub3 + 1; j <= sub1; j++)
                  aux3[aux2[j]] = 0;
                sub1 = Math.abs(aux1[sub2]);
                aux4[sub2] += sub1;
                for (j = sub3 - sub1 + 1; j <= sub3; j++) {
                  sub1 = aux2[j];
                  aux3[sub1] = 2;
                  elm4 -= c[sub1];
                  for (i = 1; i <= m; i++)
                    aux5[i] += a[i][sub1];
                }
                sub2--;
                if (aux1[sub2 + 1] >= 0) {
                  outer = true;
                  continue iterate;
                }
              } while (sub2 != 0);
            if (outer) continue;
            sol[0] = optvalue;
            a[0][0] = (optimalfound ? 0 : 1);
            for (j = 1; j <= n; j++)
              if (cminus[j]) {
                sol[j] = ((sol[j] == 0) ? 1 : 0);
                sol[0] += ccopy[j];
              }
            for (j = 1; j <= n; j++)
              c[j] = ccopy[j];
            if (!minimize) sol[0] = -sol[0];
            return;
          }
          sub1 = 0;
          idx = 0;
```

```
            for (i = 1; i <= m; i++) {
               item1 = aux5[i];
               if (item1 < 0) {
                  // infeasible constraint i
                  sub1++;
                  elm3 = 0;
                  elm1 = item1;
                  elm2 = -Integer.MAX_VALUE;
                  for (j = 1; j <= n; j++)
                     if (aux3[j] <= 0)
                        if (c[j] + elm4 >= optvalue) {
                           aux3[j] = 2;
                           aux4[sub2 + 1]++;
                           sub3++;
                           aux2[sub3] = j;
                        } else {
                           item2 = a[i][j];
                           if (item2 < 0) {
                              elm1 -= item2;
                              elm3 += c[j];
                              if (elm2 < item2) elm2 = item2;
                           }
                        }
                  if (elm1 < 0) {
                     backtrack = true;
                     continue iterate;
                  }
                  if (elm1 + elm2 < 0) {
                     if (elm3 + elm4 >= optvalue) {
                        backtrack = true;
                        continue iterate;
                     }
                     for (j = 1; j <= n; j++) {
                        item2 = a[i][j];
                        item3 = aux3[j];
                        if (item2 < 0) {
                           if (item3 == 0) {
                              aux3[j] = -2;
                              for (k = 1; k <= idx; k++) {
                                 aux7[k] -= a[aux6[k]][j];
                                 if (aux7[k] < 0) {
                                    backtrack = true;
                                    continue iterate;
                                 }
                              }
                           }
                        } else if (item3 < 0) {
                           elm1 -= item2;
                           if (elm1 < 0) {
                              backtrack = true;
                              continue iterate;
                           }
                           elm3 += c[j];
                           if (elm3 + elm4 >= optvalue) {
                              backtrack = true;
                              continue iterate;
                           }
                        }
                     }
                     idx++;
                     aux6[idx] = i;
                     aux7[idx] = elm1;
                  }
               }
            }
            if (sub1 == 0) {
               // updating the best solution
               optvalue = elm4;
               optimalfound = true;
               for (j = 1; j <= n; j++)
                  sol[j] = ((aux3[j] == 1) ? 1 : 0);
               backtrack = true;
               continue iterate;
            }
            if (idx == 0) {
               sub1 = 0;
               elm3 = -Integer.MAX_VALUE;
               for (j = 1; j <= n; j++)
                  if (aux3[j] == 0) {
                     elm2 = 0;
                     for (i = 1; i <= m; i++) {
                        item1 = aux5[i];
                        item2 = a[i][j];
```

```
                                if (item1 < item2) elm2 += (item1 - item2);
                            }
                            item1 = c[j];
                            if ((elm2 > elm3) || (elm2 == elm3) && (item1 < elm1)) {
                                elm1 = item1;
                                elm3 = elm2;
                                sub1 = j;
                            }
                        }
                    if (sub1 == 0) {
                        backtrack = true;
                        continue iterate;
                    }
                    sub2++;
                    aux4[sub2 + 1] = 0;
                    sub3++;
                    aux2[sub3] = sub1;
                    aux1[sub2] = 1;
                    aux3[sub1] = 1;
                    elm4 += c[sub1];
                    for (i = 1; i <= m; i++)
                        aux5[i] -= a[i][sub1];
                } else {
                    sub2++;
                    aux1[sub2] = 0;
                    aux4[sub2 + 1] = 0;
                    for (j = 1; j <= n; j++)
                        if (aux3[j] < 0) {
                            sub3++;
                            aux2[sub3] = i;
                            aux1[sub2]--;
                            elm4 += c[j];
                            aux3[j] = 1;
                            for (i = 1; i <= m; i++)
                                aux5[i] -= a[i][j];
                        }
                }
            }
    }

    public static void main(String[] args) {
        ArrayList arrayList = initialize();
        int A[][] = (int[][]) arrayList.get(0);
        int m = A.length - 1, n = A[0].length - 1;
        int tempB[][] = (int[][]) arrayList.get(1), B[] = new int[tempB.length];
        for (int i = 0; i < tempB.length; i++)
            B[i] = tempB[i][0];
        int tempC[][] = (int[][]) arrayList.get(2), C[] = new int[tempC.length];
        for (int i = 0; i < tempC.length; i++)
            C[i] = tempC[i][0];
        int tempTutors[][] = (int[][]) arrayList.get(3), tutors[] = new int[
              tempTutors.length + 1];
        for (int i = 0; i < tempTutors.length; i++)
            tutors[i] = tempTutors[i][0];
        tutors[tutors.length - 1] = -1;                    // Ghost ID is -1
        int tempSessions[][] = (int[][]) arrayList.get(4), sessions[] = new int[
              tempSessions.length];
        for (int i = 0; i < tempSessions.length; i++)
            sessions[i] = tempSessions[i][0];
        int[] solution = new int[n + 1];

        solveIntegerProgram(true, m, n, A, B, C, solution);
        // If there is no feasible solution, then we introduce a ghost to derive a
              feasible solution
        if (A[0][0] > 0) {
            // The first 2n lines of A ensure that only one qualified tutor is
                  assigned to a session
            for (int i = 0; i < sessions.length - 1; i++){
                A[2*i + 1][(tutors.length - 2)*(sessions.length - 1) + i + 1] = 1;
                A[2*i + 2][(tutors.length - 2)*(sessions.length - 1) + i + 1] = -1;
                C[(tutors.length - 2)*(sessions.length - 1) + i + 1] = 100;
            }

            // Run it again
            solveIntegerProgram(true, m, n, A, B, C, solution);
        }

//      System.out.print("Optimal value of the objective function = " + solution[0]
    + "\nSolution matrix:\n");
        // Print out the session IDs
        for (int i = 1; i < sessions.length; i++) {
            if (i < sessions.length - 1)
                System.out.print(sessions[i] + " ");
```

```
            else
                System.out.println(sessions[i]);
        }
        // Print out the tutor IDs and the assignments
        System.out.print(tutors[1] + ":");
        for (int i = 1; i <= n; i++) {
            System.out.print(solution[i]);
            if (i % (sessions.length - 1) != 0)
                System.out.print(" ");
            else if (i < n)
                System.out.print("\n" + tutors[i / (sessions.length - 1) + 1] + ":");
        }
    }
}
```