(b) Explain
(i) using a linear programming formulation
(ii) using the transportation algorithm
why the set of all optimal solutions is the set of all solutions which use up completely the supplies from the first three supply points.

6 Explain what modifications you would make to the transportation algorithm to convert maximisation problems to minimisation problems, giving reasons.

4 Assignment Problems

4.1 Introduction

Another special linear programming problem is the *assignment problem.* Indeed it is also a very special form of the transportation problem. Although, as such, it may be solved in principle using linear programming, or the transportation stepping-stone algorithm, there is a simpler way of solving the problem, unless it is very large and needs to be computerised, known as the *Hungarian algorithm*, being a development by H. W. Kuhn of some work of the Hungarian E. Egerváry.

As we shall see later on, the linear programming formulation is a very degenerate one which leads to some difficulties, but these can be overcome.

As an example let us suppose that we have a set I of people and a set J of jobs, each set having an equal number of members. Each person in I has to be assigned to exactly one job in J.

If element i in I is assigned to element j in J, there is a cost (or some other measure of performance) c_{ij}. The problem is to determine a total assignment which minimises the sum of the $\{c_{ij}\}$ in the assignment.

The standard assignment problem is in *minimisation* form, in which case c_{ij} must be non-negative. *Maximisation* problems can be handled by using the fact that the maximal value of a function is equal to the negative of the minimal value of its negative, in which case we can cater for negative c_{ij} values.

Let us consider a few examples.

4.2 Some examples

(i) *Medley relay*

A swimming team has 5 members and has to decide which member will swim which leg of a 5 leg relay race. The times in seconds for each swimmer, in excess of a base time for each leg (the base time does not matter since it will be incurred anyhow for any assignment) are given in Table 4.1.

The reader may like to try to solve this. The minimal total time and assignments are given later in the text where this example is needed to illustrate the Hungarian algorithm. Without the aid of the algorithm one might try to solve it using similar procedures to those used to obtain a good starting solution for the transportation problem. Thus one might use the minimal unit cost method: find the smallest c_{ij}, and assign i to j where this occurs, then delete the ith row and jth column,

Table 4.1

		Relay leg j				
		1	2	3	4	5
Swimmer i	1	4	10	3	8	3
	2	7	8	7	6	2
	3	5	6	5	3	2
	4	5	2	7	3	2
	5	6	3	9	5	2

repeating the process until an assignment is obtained. If we do this, beginning with $c_{25} = 2$ as the smallest unit cost, we will accidentally obtain the correct solution, but not if we begin with c_{35}, c_{45} or c_{55} as the smallest unit costs. An alternative approach is to see if the smallest time for each leg is achievable by some assignment since, if so, this must be optimal. For this problem such an assignment would involve swimmer 1 doing both legs 1 and 3, which is not allowed in our problem.

(ii) ***Production scheduling***
A company has n tasks to carry out. It can do only one task in any one day and a task takes exactly one day to complete. Each task, i, has a desired completion time, namely by the end of day t_i. A penalty p_i is paid for each day job i is late in being completed. The problem is to determine which task should be done on each day in order to minimise the total penalty paid.

This sort of problem is tackled in the sequencing and scheduling chapter later on, although the methods given there do not guarantee to give an optimal solution. It may be formulated as an assignment problem simply by setting $c_{ij} = p_i \max\{j - t_i, 0\}$, as is easily checked.

(iii) ***Tendering***
A company has n items to put out for contract, and n contractors. It is company policy to place each item with a different contractor. The cost of placing item i with contractor j is c_{ij}. The problem is to determine which item to place with each contractor in order to minimise the total cost.

(iv) ***Horse jumping***
Consider a problem in which we have n horses and n riders, and an expected number of penalty points for assigning a given rider to a given horse in a horse-

jumping competition. With c_{ij} equal to the penalty for assigning i to j, the problem becomes one of minimising the total expected penalty points.

(v) ***Bus crew scheduling***
A company has n bus crews and each has to do a morning route and an afternoon route. A morning route is designated i and an afternoon route j. The morning and afternoon routes are separated by a lunch break. A crew who does a morning route i and an afternoon route j has to travel a distance c_{ij} from the end of the former to the beginning of the latter. The problem is to assign crews to routes to minimise the total distance travelled.

Let us now look briefly at the formulation of the assignment problem as a linear programming problem.

4.3 Linear programming formulation

Let $x_{ij} = 1$ if member i of I is assigned to member j of J, and otherwise let $x_{ij} = 0$. Assuming that we have a balanced assignment problem, with I and J each having n members, numbered from 1 to n, then the linear programming formulation is as follows, where c_{ij} is the cost of assigning i in I to j in J.

$$\text{minimise } M = \sum_i \sum_j c_{ij} x_{ij}$$

subject to

$$\sum_j x_{ij} = 1, \quad \text{for all } i \quad \text{(i.e. } i \text{ is assigned to exactly one } j\text{)}$$

$$\sum_i x_{ij} = 1, \quad \text{for all } j \quad \text{(i.e. } j \text{ has exactly one } i \text{ assigned to it)}$$

$$x_{ij} = 0 \text{ or } 1 \quad \text{for all } i \text{ and } j.$$

This is actually a transportation problem, with the extra conditions that $x_{ij} = 0$ or 1. The transportation method will, as we know, give whole numbers, and hence 0–1 solutions in any event. Note, however, that the problem is highly degenerate, since we require n positive components for a solution, whereas linear programming requires $2n - 1$ basic variables.

4.4 The Hungarian assignment algorithm

The Hungarian algorithm proceeds by reducing the initial unit cost table to a new unit cost table for which the optimal solutions are identical to those of the initial problem, and for which the optimal solutions are more easily obtainable. The objective is to get as many zeros in the new cost matrix as possible, so that we may easily find n zeros, corresponding to a feasible assignment solution. In this case this assignment is optimal with total cost of zero, and since we keep all the unit cost entries non-negative, we cannot do better than this.

The steps in the algorithm are as follows. We will explain the algorithm as we proceed with the aid of the example of Table 4.1, which we restate as Table 4.2. We will explain the reasons for the steps afterwards.

Table 4.2

				Job *j*		
		1	2	3	4	5
Person *i*	1	4	10	3	8	3
	2	7	8	7	6	2
	3	5	6	5	3	2
	4	5	2	7	3	2
	5	6	3	9	5	2

(1) *Reducing the unit cost table to an equivalent one.* Subtract the smallest unit cost in each column from each unit cost in that column. Do the same for the rows. The resulting table tends to have more zeros than the initial one, and no unit costs are negative.

Beginning with Table 4.2 we subtract 4, 2, 3, 3, 2 respectively from each unit cost in columns 1, 2, 3, 4, 5 to obtain Table 4.3. The minimal unit cost in each row is zero and hence no row subtractions take place.

Table 4.3

				Job *j*		
		1	2	3	4	5
Person *i*	1	0	8	0	5	1
	2	3	6	4	3	0
	3	1	4	2	0	0
	4	1	0	4	0	0
	5	2	1	6	2	0

(2) *Looking for a feasible assignment solution with zero unit costs.* We look for an assignment solution in which each unit cost component is zero. If one exists it is clearly optimal for the unit cost table at this stage. If one does not exist we try to determine whether or not it is possible to draw less than five lines (row and or column lines) so that each zero of the new table lies on at least one such line. If we cannot do this, then an assignment in which each unit cost component is zero must exist, and we look again for one. If we can find less than five lines to cover all the zeros, we move to the next step. In this case we see in Table 4.3 that we can find less than five lines to cover the zeros.

It is best to find the minimal number of lines to cover all the zeros, but it is not essential to do so, and it is sufficient to find less than five lines.

Step (2) of the algorithm is non-trivial in general. Indeed it must be stressed that the Hungarian algorithm is really only suitable for small problems because of the care needed in programming computers to undertake the operations in this step. The best currently available method for solving the assignment problem is a modification of the stepping-stone method of the transportation algorithm.

Returning to step (2), it is useful, when looking for a zero assignment solution at that stage, to look for rows and columns with a single zero, since these zeros must be part of an assignment if one exists. If we need to look for a minimal cover set of lines, a useful heuristic is to find the row or column with most zeros, put a line through this, and proceed in the same way.

(3) *Further modification of the unit cost matrix to an equivalent one.* Find the smallest unit cost entry in Table 4.3 which is not on any of the lines obtained in step (1). This must be positive. Subtract it from all the unit cost entries not on any line and add it to all the unit cost entries on two of the lines. From Table 4.3, the smallest uncovered unit cost is 1 (circled) and we obtain Table 4.4.

Table 4.4

				Job *j*		
		1	2	3	4	5
Person *i*	1	0	8	0	5	2
	2	2	5	3	2	0
	3	1	4	2	0	1
	4	1	0	4	0	1
	5	1	0	5	1	0

We now begin the cycle again, starting at (1).

(1) Each row and column contains a zero, and hence no subtractions can be made without creating negative unit costs.
(2) We can cover all the zeros in Table 4.4 with four lines as indicated.
(3) The minimal uncovered unit cost is 1 and we obtain Table 4.5.

Table 4.5

		Job j				
		1	2	3	4	5
Person i	1	0	9	$0^{*\surd+}$	6	3
	2	1	5	2	2	$0^{*\surd+}$
	3	0^{+}	4	1	$0^{*\surd}$	1
	4	$0^{\surd}$	0^{*}	3	0^{+}	1
	5	0^{*}	$0^{\surd+}$	4	1	0

At this stage we are able to see that assignments exist each of whose unit cost components is zero. We thus have optimal solutions. There are three—those marked with *, $\surd$, $+$ as indicated in Table 4.5. Returning to Table 4.2, the associated minimal total cost is 16. In $\{x_{ij}\}$ form the solutions are:

$$x_{13}=1,\quad x_{25}=1,\quad x_{34}=1,\quad x_{42}=1,\quad x_{51}=1,\quad x_{ij}=0\quad \text{otherwise}$$
$$x_{13}=1,\quad x_{25}=1,\quad x_{34}=1,\quad x_{41}=1,\quad x_{52}=1,\quad x_{ij}=0\quad \text{otherwise}$$
$$x_{13}=1,\quad x_{25}=1,\quad x_{31}=1,\quad x_{44}=1,\quad x_{52}=1,\quad x_{ij}=0\quad \text{otherwise.}$$

The reasons for the steps in the algorithm

Let us now look at the reasons why the various steps are taken in the algorithm.

The validity of step (1) resides in the fact that if we subtract a constant k from each unit cost in a row or in a column (and hence from any combination of rows and columns), the ranking of the total costs of all the assignments remains the same, since we are in effect subtracting a constant k from all such costs. By using this step we try to get as many zeros as we can in the new, equivalent, unit cost table.

The line covering operation in step (2) results from the fact that an assignment solution, with zero component unit costs, exists *if and only if* the minimal number of row and/or column lines required to ensure that each zero of the whole unit cost table lies on at least one such line is n for an $n \times n$ problem. The fact that this

latter condition is necessary arises from the fact that, if we have such an assignment, we may rearrange the rows, or columns, to ensure that the assignment is the diagonal assignment, from which the necessity condition is immediately obvious. We will not prove the sufficiency condition.

It is of interest to note that the actual result we use is known as *Konig's theorem.* A set of cells is called **independent** if each row and column contains one cell or none. The theorem states that the maximal number of independent cells in a given set is equal to the minimal number of lines required to cover all the cells in the set. It is a duality result (see Chapter 2) and is a special case of our transportation problem dual. We construct a new unit cost table with ones in place of the zeros and vice versa and then use the duality theory of Chapter 2, or of Chapter 3 (in transportation form).

The further modification operations in step (3) derive from the following consideration.

First of all, if we subtract a constant from each unit cost entry not on any line and add it to all those unit cost entries on two lines, this is the same as subtracting the same constant from the unit cost entries of each row not having a line through it and adding the same constant to all unit costs in each column having a line through it. This is easily seen by partitioning the unit cost entries into four distinct sets. In set 1, the unit cost entry is not on any line; in set 2 the unit cost entry is on a column line, but not on a row line; in set 3 the unit cost entry is on a row line but not on a column line; in set 4 the unit cost entry is on a row line and on a column line.

For example, let us look at Table 4.4. Using the line set used there (and noting that this set need not be unique), we have the following sets:

set 1: $\{(2,1), (2,3), (3,1), (3,3), (4,1), (4,3), (5,1), (5,3)\}$

set 2: $\{(2,2), (2,4), (2,5), (3,2), (3,4), (3,5), (4,2), (4,4), (4,5), (5,2), (5,4), (5,5)\}$

set 3: $\{(1,1), (1,3)\}$

set 4: $\{(1,2), (1,4), (1,5)\}$.

If $k = 1$ is the constant used in step (3), then it is easily seen that the stipulated operations are equivalent to changing the cost entries in sets $\{1, 2, 3, 4\}$ by quantities $\{-1, 0, 0, 1\}$ respectively, and that this is exactly what is achieved by subtracting from rows and adding to columns as stated. Thus, in line with the basis for step (1), the new problem is equivalent to the old one. The second reason why step (3) is important is that we need to be sure that the algorithm will terminate in a finite number of steps. Now if l rows have no lines through them, and m columns have a line through them, the reduction in cost of any assignment solution, in moving from the old to the new unit cost table, is equal to $(l-m)k$, by virtue of the previous paragraph. Since step (2) has a minimal total number of requisite lines less than n, we have $(n-l)+m<n$, i.e. $l>m$. Hence $(l-m)k>0$, and we have a positive total cost reduction. We can do this only a finite number of times before the total cost of an optimal assignment is zero. Eventually we must reach a stage when $l = m$.

4.5 Extra aspects

(i) ***Maximisation problems***

Let us suppose that, given Table 4.1, we wished to *maximise* our objective instead of *minimising* it. From a linear programming point of view we may use a straight maximisation algorithm, but for the Hungarian method to work we must work with minimisation problems. Now let $\bar{c}$ be the maximal value of all the $\{c_{ij}\}$. Then from Chapter 2 we see that maximising $\Sigma_i \Sigma_j c_{ij}x_{ij}$ is equivalent to minimising $\Sigma_i \Sigma_j c'_{ij}x_{ij}$ when $c'_{ij} = \bar{c} - c_{ij}$. We thus change Table 4.1 to Table 4.6 and treat the problem as a minimisation problem.

Table 4.6

		Job j				
		1	2	3	4	5
Person i	1	6	0	7	2	7
	2	3	2	3	4	8
	3	5	4	5	7	8
	4	5	8	3	7	8
	5	4	7	1	5	8

To speed up the calculation we may equally well subtract each cost entry in each row, or column, from the maximal cost entry in that row, or column.

(ii) ***Unbalanced problems***

For each problem, I and J may contain different numbers of elements. If J contains n elements and I contains m elements, with $n > m$, and if each element of I is to be assigned to some element of J, with a penalty p_j if element j in J is not used at all, then Table 4.7 gives the equivalent balanced assignment cost unit table.

When I contains more members than J a similar approach is possible. If one wishes to allow for not assigning some i to any j, or vice versa, then a general balanced problem, with dummy rows and columns is possible as for the transportation problem of Chapter 3.

(iii) ***Other assignment objective functions***

The assignment problem we have studied is one in which the objective function is the sum of all the component unit costs of an assignment. There are other problems in which the objective function is different to this. One such problem is

Table 4.7

		Job j			
		1	2 ...	j ...	n
Person i	1	c_{11}	c_{12} ...	c_{1j} ...	c_{1n}
	2	c_{21}	c_{22} ...	c_{2j} ...	c_{2n}
	⋮	⋮	⋮	⋮	⋮
	i	c_{i1}	c_{i2} ...	c_{ij} ...	c_{in}
	⋮	⋮	⋮	⋮	⋮
	m	c_{m1}	c_{m2} ...	c_{mj} ...	c_{mn}
	$m+1$	p_1	p_2 ...	p_j ...	p_n
	⋮	⋮	⋮	⋮	⋮
	n	p_1	p_2 ...	p_j ...	p_n

known as the **maximin** assignment problem. For this problem the objective function for an assignment is the minimal unit cost component value of the assignment, and this is to be maximised.

As an example consider the following problem. We have n men and n tasks to be performed on a production line. The production rate of man i assigned to task j is a_{ij}. The items produced have to have each of the n tasks carried out on them. Then the production rate is determined by the smallest value of a_{ij} occurring in an assignment. The problem is to determine an assignment to maximise the production rate.

For the assignments *, $\surd$, + of Table 4.5, for example, the component unit costs are, respectively, from Table 4.1,

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

The minimal component unit costs for assignments *, $\surd$, + respectively are 2, 2, 2. For the maximin assignment problem, assignments *, $\surd$, + are equally good, although not optimal. In fact the assignment $x_{15} = x_{24} = x_{33} = x_{41} = x_{52} = 1$ has a minimal component value of 3 and, from this point of view, is better than any of the assignments *, $\surd$, + referred to. This raises the question of what is the correct formulation of the problem, an issue mentioned briefly at the end of Chapter 2 in the context of multiple objectives and goal programming. Let us now solve the maximin assignment problem of Table 4.1, which we reproduce as Table 4.8.

Table 4.8

				Job		
				j		
		1	2	3	4	5
	1	4	10	3	8	3
	2	7	8	7	6	2
Person *i*	3	5	6	5	3	2
	4	5	2	7	3	2
	5	6	3	9	5	2

Before doing so, note that we cannot reduce the unit costs in each row, or column, by a constant without changing the problem completely. This is a cautionary point. The reader should check this.

The following algorithm will solve the problem.

Step 1 Choose any assignment solution, e.g. the diagonal assignment.

Step 2 Find the minimal unit cost component for this solution. In this case it is 2.

Step 3 Construct a new table from the original unit cost table of Table 4.8 by putting a 0 where the original unit cost is greater than 2, and otherwise a 1. The reason for this is that a better assignment than in Step 1 exists *if and only if* this assignment has all its costs greater than 2, so that its minimal unit cost component is then greater than 2. Thus such an assignment corresponds to a set of zeros in the new table. We obtain Table 4.9.

Table 4.9

				Job		
				j		
		1	2	3	4	5
	1	0	0	0	0	0*
	2	0	0	0	0*	1
Person *i*	3	0	0	0*	0	1
	4	0*	1	0	0	1
	5	0	0*	0	0	1

Step 4 Treat the matrix in Table 4.9 as if it were an assignment problem with the specified unit costs and where the problem is to minimise the total costs. A better assignment to that in Step 1 will exist if and only if an assignment with five zeros in Table 4.9 exists. We can easily spot this; thus the assignment * in Table 4.9 is one such assignment. If we had not been able to spot such an assignment, we would have looked for less than five covering lines for all the zeros and, if we had found one, this would have meant that the minimal total cost for Table 4.9 was positive, and hence that the solution for Table 4.8 given in Step 1 was optimal.

We now repeat the procedure beginning at Step 2, and continue until we can no longer get an improvement.

Step 2 The minimal unit cost in Table 4.8, for the solution * of Table 4.9, is 3.
Step 3 The new table corresponding to Table 4.7 is given in Table 4.10.

Table 4.10

				Job		
				j		
		1	2	3	4	5
	1	0	0	1	0	1
	2	0	0	0	0	1
Person *i*	3	0	0	0	1	1
	4	0	1	0	1	1
	5	0	1	0	0	1

Step 4 We see from Table 4.10 that the minimal total cost for this table is positive (we also see that we can cover all the zeros with less than five lines) and that the solution in Table 4.9 is optimal, with a minimal unit cost component equal to 3.

If we wish to find *all* the optimal solutions, instead of using Step 2 as it is, we put a zero where all the unit costs in Table 4.8 are no less than 3, and otherwise put a 1. We obtain Table 4.11.

Any assignment corresponding to five zeros in Table 4.11 is optimal.

In Chapter 2 mention was made of multiple-objective problems and goal-programming problems, and this is relevant in this context. The original objective function was the sum of all the component costs of an assignment solution, but, as shown with the maximin solution, this need not be the case, and, indeed the solution sets may be quite different. In Exercise 6 we will specifically introduce a

Table 4.11

				Job *j*		
		1	2	3	4	5
Person *i*	1	0	0	0	0	0
	2	0	0	0	0	1
	3	0	0	0	0	1
	4	0	1	0	0	1
	5	0	0	0	0	1

multi-objective type of assignment problem in which we have, in effect, an objective function for each column. This means that, whereas the maximin problem focuses on the worst outcome for each task, we will extend this to a comparison of all tasks, so that it is the performance individually on all tasks which is important.

Exercises

1 Solve the following assignment problems with the objectives as stated.

(a) *Maximise*

				j		
		1	2	3	4	5
i	1	4	2	0	1	5
	2	1	3	5	2	7
	3	7	4	2	8	9
	4	10	0	3	4	5
	5	2	8	9	10	1

(b) *Minimise*

				j		
		1	2	3	4	5
i	1	28	25	35	33	34
	2	20	30	23	25	26
	3	36	32	36	32	40
	4	36	33	37	33	42
	5	28	30	33	35	35

(c) *Maximise*

				j		
		1	2	3	4	5
i	1	8	3	7	6	2
	2	5	1	4	9	3
	3	6	0	1	7	4
	4	8	3	8	2	8
	5	4	1	5	0	1

(d) *Maximin*

				j		
		1	2	3	4	5
i	1	4	3	9	6	2
	2	3	8	6	6	5
	3	9	1	7	4	4
	4	8	6	7	5	3
	5	4	9	5	8	2

(e) *Minimise*

				j		
		1	2	3	4	5
i	1	1	6	4	2	8
	2	6	3	4	7	2
	3	4	5	6	3	7
	4	6	2	8	6	3
	5	2	8	5	2	6

2 After qualifying, medical students must take two six-month jobs in hospital departments, but they cannot take both jobs in the same department. A hospital has four such students and vacancies in four departments; casualty, maternity, medical and surgical. The number of fatal mistakes each student will make in each department is given by

	Casualty	Maternity	Medical	Surgical
Student 1	3	0	2	6
Student 2	2	1	4	5
Student 3	4	2	5	7
Student 4	2	0	2	4

(a) How should they be allotted to departments for the first job so as to minimise total mistakes?

(b) Given that allocation, how should they be allotted for the second six months so no one stays in the same job, and mistakes are minimised?

(c) For this problem, do these two allocations minimise total mistakes over both six-month periods? Explain why or why not.

(d) For any problem, of the above kind, does this method always minimise total number of casualties? Either prove it or give a counter example.

3 A council has three jobs to be done and receives tenders from three firms. The tenders in appropriate units are as follows, where i denotes the firm, and j denotes the job.

		j		
		1	2	3
	1	20	35	8
i	2	15	40	7
	3	12	33	6

Firm 1 is capable of doing all jobs at the same time, firm 2 is capable of doing jobs 2 and 3 at the same time, and firm 3 is capable of doing all jobs, but can only do one at a time. Set up the problem of assigning jobs to firms as an assignment problem and find the solution. Is it unique?

4 In the text it was stated that the minimal number of lines required to cover all the zero cells in an assignment table was equal to the maximal number of independent zero cells, and that this could be deduced from duality theory. Use the duality formulation of the transportation problem to deduce this.

5 In a transportation problem if $x_{ij} \neq 0$, there is a time t_{ij} taken to carry out the actual transportation. How would you modify the assignment maximin algorithm to find the min max transportation time?

6 In an assignment problem, let the component unit costs for the n columns be, for solution x,

$c_1(x), c_2(x), \ldots, c_j(x), \ldots, c_n(x)$, where $c_j(x)$ is the jth column unit cost.

Solution x is said to dominate solution y if

$$c_j(x) \leqslant c_j(y), \quad j = 1, 2, \ldots, n$$

and $c_j(x) < c_j(y)$, for some j.

For example in Table 4.12 the circled assignment dominates the assignment given in squares.

Table 4.12

		Job *j*				
		1	2	3	4	5
	1	④	$\boxed{10}$	3	8	3
	2	7	8	7	$\boxed{6}$	②
Person *i*	3	5	⑥	5	3	$\boxed{2}$
	4	$\boxed{5}$	2	⑦	3	2
	5	6	3	$\boxed{9}$	⑤	2

How would you formulate as an assignment problem the problem of determining whether the circled assignment is dominated by some other assignment? Just put down the appropriate unit cost table and explain its entries. Clue: try to find a unit cost table using similar approaches to that of the max min assignment problem.