1 Allocation

This chapter is about matching the elements of one set with elements of another. When you have completed it you should

- be able to interpret allocation problems as matching problems where cost must be minimised
- know how to use the Hungarian algorithm to solve allocation problems.

1.1 Introduction

Matching the elements of two different sets is a common task which you often perform without being aware of it. When you distribute copies of a newsletter to your friends you are matching the set of newsletters with the set of friends. In this case the task is easy because any copy can be assigned to any friend.

Matching becomes more difficult when the two sets to be matched have characteristics that make certain assignments undesirable or impossible. For example, in matching people with jobs, the skills of the people and the requirements of the jobs mean that some assignments should not be made.

You have already met this kind of matching problem in D1 Chapter 5. In this chapter you will study another type of matching problem in which each assignment has a cost which you must try to minimise.

Suppose that a building company has four contracts which must be completed at the same time. The work is to be done by subcontractors, each of whom can carry out only one contract. The subcontractors' quotes for each of the four jobs are shown in Table 1.1.

		Contract			
		A	*B*	*C*	*D*
Subcontractor	1	10	5	9	–
	2	9	6	9	6
	3	10	–	10	7
	4	9	5	9	8

Quotes are in £1000s.
'–' indicates no quote.

Table 1.1

Here there is no difficulty in finding a matching. An example is 1-*A*, 2-*B*, 3-*C*, 4-*D*. The problem is to find a matching which minimises the total cost; such a problem is called an **allocation problem**.

A good method of tackling allocation problems is to reduce the array of costs by subtracting from each element of a row (or column) the least element in that row (or column). For example, Table 1.3 is obtained from Table 1.2 by subtracting 5, 6, 7 and 5 from the first,

second, third and fourth rows respectively. The smallest number in each new row is now equal to zero.

10	5	9	–	–5 →	5	0	4	–
9	6	9	6	–6 →	3	0	3	0
10	–	10	7	–7 →	3	–	3	0
9	5	9	8	–5 →	4	0	4	3

Table 1.2 Table 1.3

The solution to the original problem will then simply be the solution to the new problem with an extra cost of £23,000 because $5 + 6 + 7 + 5 = 23$. Reducing the columns of Table 1.3 in a similar way, in this case by subtracting 3 from the first and third columns, leads to the array in Table 1.4. (The extra cost to be added to a solution is now £29,000.)

2	0	1	–
0	0	0	0
0	–	0	0
1	0	1	3

Table 1.4

For this matrix, it is easy to see that there are no matchings of zero cost but there are several of cost only 1. For example, Table 1.5 shows a matching of cost 1 in bold type. So the original problem has an optimal allocation, of cost £30,000, shown in Table 1.6.

2	**0**	1	–
0	0	**0**	0
0	–	0	**0**
1	0	1	3

Table 1.5

10	**5**	9	–
9	6	**9**	6
10	–	10	**7**
9	5	9	8

Table 1.6

In general, if you obtain one array from another by adding or subtracting *any* fixed number from each element of a row (or column), a solution to the new allocation problem gives a solution to the original problem, and vice versa. In Table 1.2, for example, adding 10 to each element of the first column corresponds to all the contractors raising their quotes on contract *A* by £10,000. The costs of all possible matchings go up by £10,000, so your best choice of matching remains the same.

To solve an allocation problem, first reduce the array of costs by subtracting the least number in a row (or column) from each element of that row (or column).

Example 1.1.1
Choose five numbers from the array given below so that the sum of the five numbers is the least possible. No two numbers can be in the same row or column.

10	10	9	8	10
10	12	12	9	13
16	16	14	12	15
14	15	12	12	16
15	16	14	13	14

Reducing the array by rows leads to the array on the left; then reducing by columns leads to the array on the right, in which the minimum allocation is shown in bold type.

2	2	1	0	2
1	3	3	0	4
4	4	2	0	3
2	3	0	0	4
2	3	1	0	1

1	**0**	1	0	1
0	1	3	0	3
3	2	2	**0**	2
1	1	**0**	0	3
1	1	1	0	**0**

For the original array the pattern indicated by the emboldened numbers yields the minimum allocation of

$$10 + 10 + 12 + 12 + 14 = 58.$$

Having negative numbers in the array does not affect the method because the stage of 'subtracting the least number in a row from each element of that row' will make all elements at least zero. For example, in Table 1.7 each element has −3 subtracted from it, which is just the same as adding 3 to each element.

8	−3	4	7	−(−3) or +3 →	11	0	7	10

Table 1.7

To maximise an allocation you can therefore simply change the sign of every element and then minimise in the usual way, as in Table 1.8.

3	1	2	Change sign →	−3	−1	−2	−(−3)→	0	2	1

Table 1.8

The same effect is produced by subtracting each element from the largest value in the array, shown in Table 1.9.

3	1	2	Subtract from 3 →	0	2	1

Table 1.9

The next example illustrates this process.

Example 1.1.2
Choose five numbers from the array given below so that the sum of the five numbers is the greatest possible. No two numbers can be in the same row or column.

10	10	9	8	10
10	12	12	9	13
16	16	14	12	15
14	15	12	12	16
15	16	14	13	14

First, convert the problem to a standard minimising one by changing the sign of each element of the array or by subtracting each element from 16. Reducing by rows then gives:

0	0	1	2	0
3	1	1	4	0
0	0	2	4	1
2	1	4	4	0
1	0	2	3	2

Reducing by columns gives:

0	0	0	**0**	0
3	1	**0**	2	0
0	0	1	2	1
2	1	3	2	**0**
1	**0**	1	1	2

The numbers in bold type show the minimum allocation. For the original array the same pattern yields the maximum allocation of $16 + 16 + 12 + 8 + 16 = 68$.

1.2 The Hungarian algorithm

The opening example of Section 1.1 led to the reduced array in Table 1.10.

2	0	1	–
0	0	0	0
0	–	0	0
1	0	1	3

Table 1.10

At this stage you had to 'spot' that an allocation of four zeros was impossible and that it was necessary to use a 1. In larger and more complicated examples it is better to have a systematic procedure. One such method is called the **Hungarian algorithm**.

To apply this algorithm start with an array that has been reduced by rows and columns. Cover all the zero elements with the minimum number of vertical or horizontal lines or both. For the array in Table 1.10 three lines are needed, as shown in Table 1.11.

2	0	1	–
0	0	0	0
0	–	0	0
1	0	1	3

Table 1.11

Note the least uncovered element, in this case 1. Add this element to the elements of each covered row and then to the elements of each covered column. Where an element is covered twice, add the least uncovered element twice. See Table 1.12 for the result of this process.

2	1	1	–
1	2	1	1
1	–	1	1
1	1	1	3

Table 1.12

Then subtract this element from every element of the array. The result is shown in Table 1.13.

1	0	0	–
0	1	0	0
0	–	0	0
0	0	0	2

Table 1.13

1	**0**	0	–
0	1	0	0
0	–	0	**0**
0	0	**0**	2

Table 1.14

It is now possible to allocate four zeros in several ways; one is shown by the numbers in bold type in Table 1.14. This pattern, applied to the original array, yields the minimum value of $9 + 5 + 9 + 7 = 30$, that is £30,000.

The reason that this procedure further reduces the array is that the minimum number is added on $3 \times 4 = 12$ times but is then subtracted 16 times. The following statement of the Hungarian algorithm shortens this procedure.

Hungarian algorithm

Step 1 Reduce the array of costs by both row and column subtractions.

Step 2 Cover the zero elements with the minimum number of lines. If this minimum number is the same as the size of the array (which for a square matrix means the number of rows) then go to Step 4.

Step 3 Let m be the minimum uncovered element. The array is augmented by reducing all uncovered elements by m and increasing all elements covered by two lines by m. Return to Step 2. This process is called **augmenting the elements**.

Step 4 There is a maximal matching using only zeros. Apply this pattern to the original array.

Step 3 of the Hungarian algorithm is equivalent to the procedure used on the previous page: adding the least uncovered element to all elements covered by one line, adding twice the least uncovered element to all elements covered by two lines and then subtracting the least uncovered element from every element in the array. The longer procedure was presented first because it shows more clearly that the best matching remains the same at each stage.

Example 1.2.1
A company has four sales representatives to allocate to four groups of retailers. The table shows the estimated weekly mileage of each representative when assigned a particular group. How should the groups be allocated to minimise the total mileage?

	1	2	3	4
Alex	280	280	260	210
Ben	470	480	460	420
Charles	370	390	380	330
Davinia	220	250	240	220

The following array shows the situation after Step 1, after row and column reductions.

7	4	3	0
5	3	2	0
4	3	3	0
0	0	0	0

Numbers are tens of miles.

The next array, on the left, shows the situation after Step 2, where two lines are needed to cover the zeros. The right array shows Step 3, where 2, the minimum uncovered element, has been subtracted from all uncovered elements, and elements covered by two lines have been increased by 2. After this, the algorithm returns to Step 2.

7	4	3	0
5	3	2	0
4	3	3	0
~~0~~	~~0~~	~~0~~	~~0~~

5	2	1	0
3	1	0	0
2	1	1	0
~~0~~	~~0~~	~~0~~	~~2~~

Steps 2 and 3 are carried out again, as before.

5	2	1	0
~~3~~	~~1~~	~~0~~	~~0~~
2	1	1	0
~~0~~	~~0~~	~~0~~	~~2~~

4	1	0	0
~~3~~	~~1~~	~~0~~	~~1~~
1	0	0	0
~~0~~	~~0~~	~~0~~	~~3~~

At this stage, carrying out Step 2 requires four lines. As this is the same as the size of the array, you can go to Step 4, where the pattern of zeros is shown in bold type.

~~4~~	~~1~~	~~0~~	~~0~~
~~3~~	~~1~~	~~0~~	~~1~~
~~1~~	~~0~~	~~0~~	~~0~~
~~0~~	~~0~~	~~0~~	~~3~~

4	1	0	**0**
3	1	**0**	1
1	**0**	0	0
0	0	0	3

Using the pattern of zeros, the optimum allocation is 1-Davinia, 2-Charles, 3-Ben, 4-Alex, with mileage $220 + 390 + 460 + 210 = 1280$.

1.3 Non-square arrays

Suppose five workers are available for four tasks. The times each worker would take at each task are in Table 1.15. How can each task be allotted to a different worker to minimise the total time?

	1	2	3	4
Angel	170	220	190	200
Britney	140	230	150	160
Cleo	180	210	170	170
Dimitri	190	240	210	220
Ed	160	220	170	170

Times are in minutes.

Table 1.15

To be able to apply the methods of this chapter it is first necessary to create a square array by adding a dummy column.

The trick is to add in a column of equal numbers so that this column does not influence the choice of workers for the other tasks. It is conventional (but not necessary) to make these numbers equal to the largest number in the array. This technique is shown in the next example.

Example 1.3.1
Select four numbers from the array given below so that the sum of the four numbers is the least possible. No two numbers can be in the same row or column.

170	220	190	200
140	230	150	160
180	210	170	170
190	240	210	220
160	220	170	170

Add in a column of 240s to obtain a square array.

170	220	190	200	240
140	230	150	160	240
180	210	170	170	240
190	240	210	220	240
160	220	170	170	240

Reducing rows and then columns, and dividing by 10, so that the entries are in tens, gives the following array, which requires three lines to cover the zeros.

0	1	2	3	2
0	5	1	2	5
1	0	0	0	2
0	1	2	3	0
0	2	1	1	3

After covering the first column, and the third and fourth rows, and augmenting the elements (only one return to Step 2 is required), you obtain the array on the next page in which the zeros in each row and column are in bold type.

0	**0**	1	2	1
0	4	**0**	1	4
2	0	0	**0**	2
1	1	2	3	**0**
0	1	0	0	2

The solution to the original problem is $160 + 220 + 150 + 170 = 700$.

To apply the Hungarian algorithm to a non-square array, first add in dummy rows or columns to make the numbers of rows and columns equal.

Exercise 1

1 The four members of a swimming relay team must, between them, swim 100 metres of each of backstroke, breaststroke, butterfly and crawl.

Five hopefuls for the team have personal best times as follows.

	Back	Breast	Butterfly	Crawl
A	66	68	71	60
B	69	69	72	60
C	68	70	73	61
D	65	66	71	63
E	63	65	74	60

Times in seconds for 100 meters.

(a) Convert the array into a square array to which the Hungarian algorithm can be applied.

(b) Hence find which four swimmers should be chosen and the stroke for which each should be used.

2 The scores of the four members of a quiz team on practice questions are as follows.

	Sport	Music	Literature	Science
Ali	16	18	17	14
Bea	19	17	14	18
Chris	12	16	16	15
Deepan	11	15	17	14

A different person needs to be picked for each of the four different topics.

(a) How could the table be altered in order to use the Hungarian algorithm?

(b) Hence allocate the topics to the members of the team.

3 Apply the Hungarian algorithm to find the minimum possible total of six numbers, chosen from the table below in such a way that no two numbers lie in the same row or column.

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

4 Find the maximum possible total for six numbers chosen as in Question 3.

5 Suppose that the Hungarian algorithm is being applied to an array.

(a) Suppose further that the zeros in a 5×5 array are covered by three lines and the least non-covered element is a 2. When the array is augmented once, what is the reduction in the total of all the elements?

(b) Suppose that the zeros of an $n \times n$ array are covered by k lines and that the least non-covered element is l. What is the reduction in the total of all elements when this array is augmented once?

6 A builder has four labourers who must be assigned to four tasks. The estimates of the times each labourer would take for the different tasks are as shown in the table.

	Task			
	1	2	3	4
A	3	4	4	3
B	3	3	1	2
C	4	3	2	4
D	4	1	2	3

Times in hours.

Given that no labourer can be assigned to more than one task, use the Hungarian algorithm to find the optimum assignment.

7 A relay team has four runners who must be assigned to the four legs of a 4×2 mile race. The estimates of the times that each runner would take for the different legs are in the table.

	Leg			
	1	2	3	4
Haile	710	710	710	700
Josh	700	670	680	690
Stephen	670	660	670	680
Tom	680	670	660	710

Find the optimum assignment of runners.