Decision Mathematics 2

Stan Dolan

Series editor Hugh Neill

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 2RU, UK

www.cambridge.org
Information on this title: www.cambridge.org/9780521619158

First published as *Discrete Mathematics 2* 2001
Second edition 2005

Printed in the United Kingdom at the University Press, Cambridge

A catalogue record for this publication is available from the British Library

ISBN-13 978-0-521-61915-8 paperback
ISBN-10 0-521-61915-7 paperback

Cover image © Digital Vision

Contents

1 Matching

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

- be able to represent a matching problem by means of a bipartite graph
- be able to find a maximal matching
- be able to apply the Matching Augmentation algorithm
- 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.

Bipartite graphs provide a useful way to represent such **matching problems**. You met the idea of a bipartite graph in D1 Section 2.1.

A **bipartite graph** is a graph with two sets of nodes such that arcs only connect nodes from one set to the other and do not connect nodes within a set.

It is convenient to always consider the two sets of nodes of a bipartite graph to be the **left-nodes** and the **right-nodes**. For example, $K_{3,3}$ would be drawn as shown in Fig. 1.1.

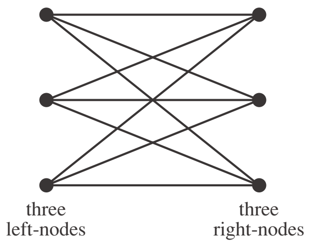


Fig. 1.1

The two sets of nodes correspond to the two sets to be matched and the arcs correspond to the assignments that can be made.

For any bipartite graph, a **matching** is a set of arcs which have no nodes in common.

A **maximal matching** is any matching which contains the largest possible number of arcs.

You have already met a matching problem in D1 Exercise 2A Question 8. Example 1.1.1 shows the problem again.

Example 1.1.1
Four Members of Parliament, Ann, Brian, Clare and David, are being considered for four cabinet posts. Ann could be Foreign Secretary or Home Secretary, Brian could be Home Secretary or the Chancellor of the Exchequer, Clare could be Foreign Secretary or the Minister for Education and David could be the Chancellor or the Home Secretary.

(a) Draw a bipartite graph to represent this situation.

(b) How many options does the Prime Minister have?

(a)

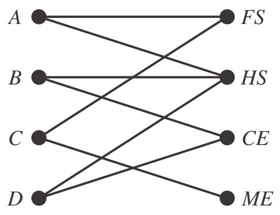


Fig. 1.2

(b)

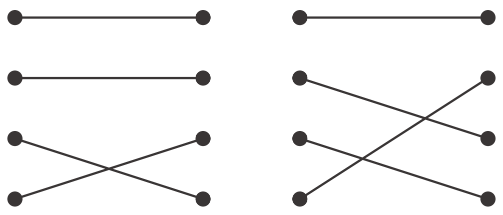


Fig. 1.3

Figure 1.2 shows the graph, and Fig. 1.3 shows the two options.

The maximal matching of Example 1.1.1 covered all of the nodes and each diagram in Fig. 1.3 is called a **complete matching**. A complete matching is not always possible.

Example 1.1.2
Four couples are booked into a small hotel. The Smiths have requested a double room and the Joneses have asked for a double room on the ground floor. The Browns require a twin-bedded room and the Greens will be happy with any room on the ground floor. The hotel manager has just four rooms to assign, three double and one twin-bedded. The twin-bedded room and one of the doubles is on the ground floor. Can the manager satisfy the requirements of all the couples?

The relevant bipartite graph is shown in Fig. 1.4.

The requirements of the Joneses, Browns and Greens cannot all be satisfied, because the Smiths are the only couple who are content with the double rooms which are not on the ground floor, and they cannot use both of them. The maximal matching has three pairings. One example is $\{S, D1\}$, $\{J, DG\}$, $\{B, TG\}$.

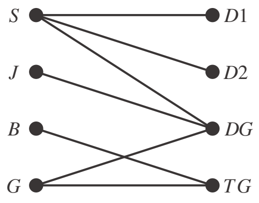


Fig. 1.4

As well as using bipartite graphs to represent the information in assignment problems, you can also use **adjacency matrices**.

The left-nodes are put down one side, and the right-nodes along the top. An entry of 1 means that the corresponding nodes are linked by an arc.

For example, the matrix in Fig. 1.5 contains the same information as the graph of Fig. 1.4.

$$\begin{array}{c|cccc} & D1 & D2 & DG & TG \\ \hline S & 1 & 1 & 1 & 0 \\ J & 0 & 0 & 1 & 0 \\ B & 0 & 0 & 0 & 1 \\ G & 0 & 0 & 1 & 1 \end{array}$$

Fig. 1.5

For a problem formulated as an adjacency matrix, the task is to find the maximum number of 1s such that no two of these 1s are in the same row or in the same column. In Fig. 1.6 the solution is shown by the 1s in bold-faced type. If you were solving this problem you would probably wish to circle the 1s in the solution.

$$\begin{pmatrix} \mathbf{1} & 1 & 1 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Fig. 1.6

1.2 The Matching Augmentation algorithm

In the previous section you solved simple matching problems by inspection. As the numbers of nodes and arcs increase, inspection becomes a hit or miss affair and the chance of overlooking a maximal matching increases.

Fortunately, there is a simple algorithm which can be used to improve upon (or augment) an initial matching or to show that you have already obtained a maximal matching.

Matching Augmentation algorithm

Step 1 Consider all arcs of the matching to be directed from right to left. Consider all other arcs to be directed from left to right.

Step 2 Identify all nodes which do not belong to the matching. Create a new node, *X* say, joined with directed arcs to all left-nodes which do not belong to the matching.

Step 3 Give each arc a weighting of 1.

Step 4 Apply Dijkstra's algorithm from *X* until one of the following happens:

- a right-node which does not belong to the matching is reached
- no further labelling is possible. In this case, the initial matching cannot be improved and the algorithm stops.

Step 5 Retrace any path from an unmatched left-node to the unmatched right-node. This is called an alternating path.

Step 6 Remove from the original matching any arcs in the path of Step 5. Add to the matching the other steps in the path. This increases, by 1, the number of arcs in the matching.

Although the following example has only 10 nodes, it is nevertheless difficult to spot a maximal matching. This example will be used to illustrate the Matching Augmentation algorithm.

Example 1.2.1
A builder employs five workers: Alan, who does labouring, plastering and joinery; Betty, who does labouring and wiring; Colin, who does bricklaying and wiring; Di, who does plastering and bricklaying; and Ed, who does plastering, joinery and bricklaying. Can each of the five workers be assigned to a task so that all five tasks are covered?

First, draw a bipartite graph for this problem. In Fig. 1.7, the heavy lines represent a first try at a matching, which assigns just four workers to tasks.

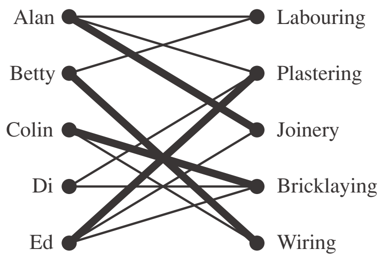


Fig. 1.7

The Matching Augmentation algorithm will show how to improve this matching, if it is possible.

The result of applying the first four steps of the Matching Augmentation algorithm to Example 1.2.1 is shown in Fig. 1.8.

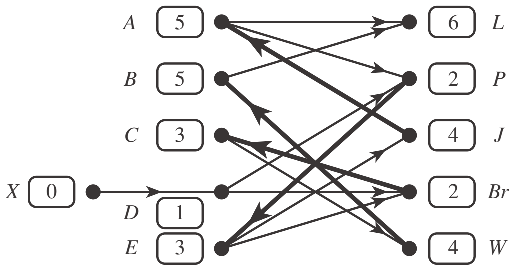


Fig. 1.8

You can see that all the paths have been given directions, as required in Step 1.

The new node *X* has been created and joined to *D*, the left-node which does not belong to the initial matching.

Dijkstra's algorithm is then applied from *X*, and the right-node *L*, which doesn't belong to the initial matching, is reached. At this stage Step 4 has been completed.

To carry out Step 5, you need to trace a path from *D* to *L*: there are two possibilities, *D-P-E-J-A-L* and *D-Br-C-W-B-L*, and you could choose either one.

Looking at the first of these, as you trace the path you cover *P-E* and *J-A*, both of which were in the initial matching. Remove them from the initial matching, and add the other arcs from the path, *D-P, E-J* and *A-L*, to the initial matching. This increases by 1 the number of arcs in the matching. In this case, the matching is now maximal. It is

A-L, *B-W*, *C-Br*, *D-P*, *E-J*.

If you had chosen the other path, *D-Br-C-W-B-L*, you would have covered *Br-C* and *W-B* from the initial matching. If you removed them, and added *D-Br, C-W* and *B-L* to the original matching, the new matching would be

A-J, *B-L*, *C-W*, *D-Br*, *E-P*,

an alternative maximal matching.

The next example illustrates how the Matching Augmentation algorithm recognises that a maximal matching has already been reached.

Example 1.2.2
A school timetabling team is trying to timetable four teachers, Andy, Barbara, Chris and Dave to four classes, denoted by *R, S, T* and *U*.

Andy can teach *R* or *S*	Barbara can teach *R, S, T* or *U*
Chris can teach *R* or *S*	Dave can teach *S*.

Draw a bipartite graph and apply the Matching Augmentation algorithm to find a maximal matching.

An initial attempt at a matching is illustrated by the heavy lines in Fig. 1.9.

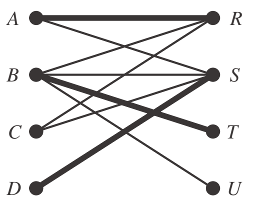


Fig. 1.9

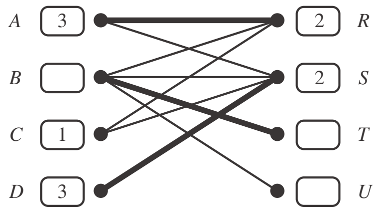


Fig. 1.10

The result of applying the Matching Augmentation algorithm is then shown in Fig. 1.10. The label *X*, that was initially created and then joined to *C* is not shown.

No further labelling is possible, so the initial matching is maximal. Notice that although it is maximal it is not complete.

For an existing non-maximal matching, *M*, of a bipartite graph, *G*, the Matching Augmentation algorithm produces what is called an alternating path.

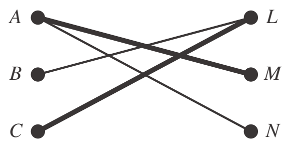
An **alternating path** for *M* in *G* is a path which consists alternately of arcs in *M* and arcs not in *M*.

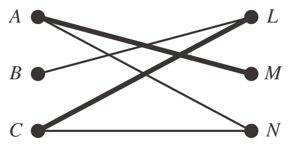
This definition makes it easier to see why the Matching Augmentation algorithm works. The following argument is not a proof, however.

Consider the two situations in which Step 4 of the algorithm stops. If Step 4 stops because no further labelling is possible, the algorithm has shown the initial matching to be maximal. If Step 4 stops because a right-node which isn't in the matching has been reached, Step 6 will improve the matching. In the second case the algorithm has constructed a path from an unmatched left-node to an unmatched right-node. The path is alternating because the only way to go from left to right is via an arc which is not in M, and the only way to go from right to left is via an arc which is in M. Because the path starts at the left and ends at the right it must contain one more arc not in M than in M, and so Step 6 of the algorithm produces a matching that contains one more arc than the initial matching.

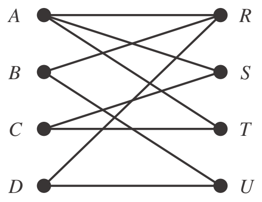
Exercise 1A

1 Apply the Matching Augmentation algorithm to the following bipartite graphs where the heavy lines represent matchings. In each case state what you can conclude.

(a)


(b)


2 Find the adjacency matrix corresponding to the bipartite graph in the figure.



Select four 1s in the matrix in such a way that no two 1s are in the same row or column. Hence find a maximal matching for the bipartite graph.

3 A large department store employs five students, Anita, Bruce, Chloe, Darminder and Errol, for the Christmas period.

The Hardware manager would be happy to use Anita, Chloe or Darminder.
The Bookshop manager would be happy to use Bruce or Errol.
The Sports manager would be happy to use Anita or Darminder.
The Electrical manager would be happy to use Darminder or Errol.
The Food hall manager would be happy to use Anita or Chloe.

(a) Draw a bipartite graph, G, to show which students are suitable for each department.

The managing director initially decides to place Anita in the Food hall, Chloe in Hardware, Darminder in Electrical and Errol in the Bookshop. However, he is then unable to place Bruce appropriately.

(b) Show the incomplete matching, M, that describes the managing director's attempted allocation.

(c) Use the Matching Augmentation algorithm to construct an alternating path for M in G, and hence find a complete matching.

4 A dance team consists of four men, Ahmed, Benny, Chris and Derek, paired with four women, Ann, Bala, Celine and Di. Ann is only prepared to dance with Ahmed, Bala will dance with Benny or Derek, Celine will dance with Chris, and Di will dance with any of the men. In their first competition, Di dances with Derek.

(a) Draw a bipartite graph, with the women on the left and the men on the right, to represent the only possible matching with Di and Derek paired.

Unfortunately, Di and Derek fall out and a different pairing has to be arranged for the second competition.

(b) Draw a bipartite graph to show the possible pairings and the incomplete matching, *M*, from the first competition.

(c) Apply the Matching Augmentation algorithm to obtain a complete matching for the second competition.

1.3* Maximal matching–minimum cover

This section contains extension material and may be omitted.

There is an important connection between the matching problem and the problem of finding sets of nodes which cover every arc, that is sets of nodes which contain at least one end-node of each arc of the bipartite graph. These are called **cover sets** of the graph.

Consider, for example, the graph drawn for Example 1.2.2, redrawn as Fig. 1.11.

Since *A*-*R*, *B*-*T* and *D*-*S* is a matching, a cover set must contain at least one of *A* or *R*, at least one of *B* or *T* and at least one of *D* or *S*.

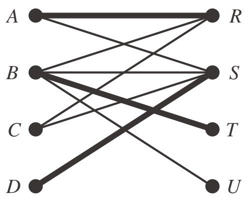


Fig. 1.11

In general it is clear that:

The number of arcs in any matching	⩽	the number of nodes in any cover set.

The smallest possible number of nodes in a cover set cannot, therefore, be less than the number of arcs in a maximal matching. In Example 1.2.2, this minimum is actually achieved for the cover set $\{B, R, S\}$. The interesting aspect of the connection between the matching problem and the cover set problem is that this result is always true. That is:

The number of arcs in a maximal matching	=	the minimum number of nodes in a cover set.

This is called the maximal matching–minimum cover result.

You can prove this result by considering the effect of applying the Matching Augmentation algorithm to bipartite graphs for which the maximal matching has already been achieved.

First, consider the following examples:

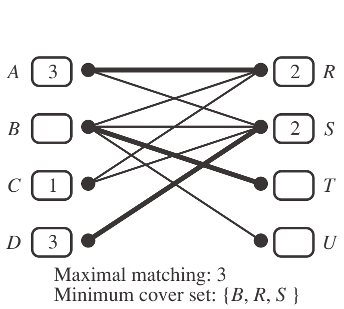


Fig. 1.12

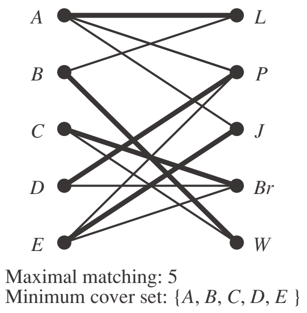


Fig. 1.13

The key to proving the maximal matching–minimum cover result is to note how the examples of cover sets in Figs. 1.12 to 1.14 are formed. In each case, they consist of the unlabelled left-nodes and labelled right-nodes.

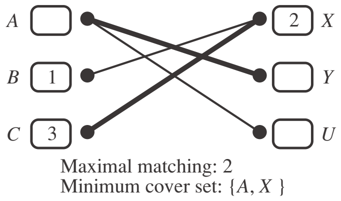


Fig. 1.14

To prove the maximal matching–minimum cover result it is therefore necessary to prove:

- the unlabelled left-nodes and labelled right-nodes form a cover set;
- the number of these nodes equals the number of arcs in a maximal matching.

These two facts follow from the following features of maximal matchings to which the Matching Augmentation algorithm has been applied. In each case, you should try to explain the reason for these features (see Miscellaneous exercise 1 Question 12).

- Each arc of the bipartite graph either has an unlabelled left-node or a labelled right-node.
- All unlabelled left-nodes are in the matching.
- All labelled right-nodes are in the matching.
- Each arc of the matching joins nodes which are either both labelled or both unlabelled.

Then:

Number of arcs of maximal matching

= number of right-nodes of matching

= number of unlabelled right-nodes of matching
+ number of labelled right-nodes of matching

= number of unlabelled left-nodes of matching
+ number of labelled right-nodes of matching

= number of unlabelled left-nodes + number of labelled right-nodes.

Each arc of the bipartite graph has either an unlabelled left-node or a labelled right-node, but not both. So the set of unlabelled left-nodes and labelled right-nodes is a cover set. It must be a minimum cover set since it has the same number of nodes as the number of arcs in a matching.

The maximal matching–minimum cover result can be useful for seeing quickly that you have found a maximal matching and need not apply the Matching Augmentation algorithm.

Example 1.3.1
For the bipartite graph shown in Fig. 1.15, find

(a) a cover set containing four nodes,

(b) a matching containing four arcs.

What can you conclude about your answers to (a) and (b)?

(a) $A,\ B,\ T,\ V$.

(b) $\{A, R\}, \{B, S\}, \{C, T\}, \{E, V\}$.

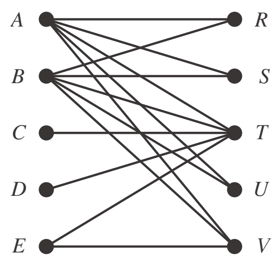


Fig. 1.15

A cover set and a matching of the same sizes have been found. The solution to part (a) is therefore a minimum cover set and the solution to part (b) is a maximal matching.

1.4 Allocation problems

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 1.16.

		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 1.16

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 1.18 is obtained from Table 1.17 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	–
9	6	9	6
10	–	10	7
9	5	9	8

Table 1.17

$-5 \rightarrow$
$-6 \rightarrow$
$-7 \rightarrow$
$-5 \rightarrow$

5	0	4	–
3	0	3	0
3	–	3	0
4	0	4	8

Table 1.18

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 1.18 in a similar way, in this case by subtracting 3 from the first and third columns, leads to the array in Table 1.19.

2	0	1	–
0	0	0	0
0	–	0	0
1	0	1	3

Table 1.19

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 1.20 shows a matching of cost 1 in bold type.

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

Table 1.20

So the original problem has an optimal allocation, of cost £30,000, shown in Table 1.21.

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

Table 1.21

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 1.4.1
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 1.22 each element has −3 subtracted from it, which is just the same as adding 3 to each element.

8	−3	4	7

$-(-3)$ or $+3 \rightarrow$

11	0	7	10

Table 1.22

To maximise an allocation you can therefore simply change the sign of every element and then minimise in the usual way, as in Table 1.23.

3	1	2

Change sign $\rightarrow$

−3	−1	−2

$-(-3) \rightarrow$

0	2	1

Table 1.23

The same effect is produced by subtracting each element from the largest value in the array, shown in Table 1.24.

3	1	2

Subtract from 3 $\rightarrow$

0	2	1

Table 1.24

The next example illustrates this process.

Example 1.4.2

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 $16 + 16 + 12 + 8 + 16 = 68$.

1.5 The Hungarian algorithm

The opening example of Section 1.4 led to the reduced array in Table 1.25.

2	0	1	–
0	0	0	0
0	–	0	0
1	0	1	3

Table 1.25

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, called the **Hungarian algorithm**, is based upon the ideas of Section 1.3 and depends upon covering the zero elements with the minimum number of vertical or horizontal lines or both.

In this case, three lines are needed, shown in Table 1.26.

2	0	1	–
~~0~~	~~0~~	~~0~~	~~0~~
~~0~~	~~–~~	~~0~~	~~0~~
1	0	1	3

Table 1.26

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 1.27 for the result of this process.

2	1	1	–
1	2	1	1
1	–	1	1
1	1	1	3

Table 1.27

Then subtract this element from every element of the array. The result is shown in Table 1.28.

1	0	0	–
0	1	0	0
0	–	0	0
0	0	0	2

Table 1.28

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

Table 1.29

It is now possible to allocate four zeros in several ways; one is shown by the numbers in bold type in Table 1.29. 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.

Example 1.5.1
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 array on the right shows the situation after Step 1, after row and column reductions. The numbers are tens of miles.

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

The array below on the left shows the situation after Step 2, where two lines are needed to cover the zeros. The array below on the right 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.

The dependence of the Hungarian algorithm upon the minimum number of lines covering zeros is explained in the starred Section 1.3. If the maximal matching is shown in the form of an adjacency matrix, zero-cost arcs are represented by entries of zero. A maximal matching is required, so the minimum cover is needed.

1.6 Non-square arrays

Suppose that five workers are available for four tasks. The times each worker would take at each task are given in Table 1.30. How can each task be allocated 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

Table 1.30

To be able to apply the methods of this chapter it is first necessary to create a square array by adding in 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 1.6.1
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 following array 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 1B

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 metres.

(a) Convert the array into a square one 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.