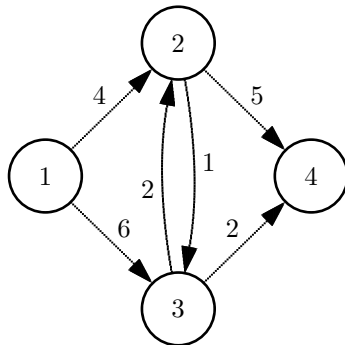


## 5 Dynamic and Integer Programming

### 5.1 Shortest routes

A *network* is a directed graph  $G = (N, A)$  where  $N$  is a set of nodes and  $A$  is a set of (directed) arcs  $ij$ , and where we have a 'length'  $a_{ij}$  attached to each arc  $ij$ .



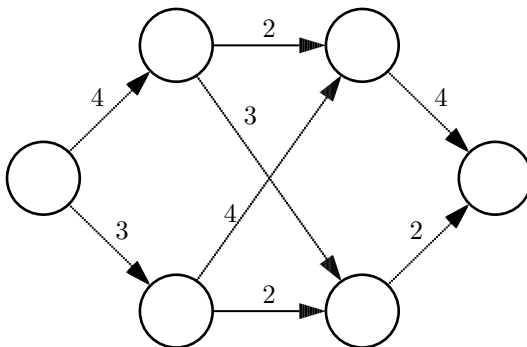
The nodes may correspond to cities and the arcs to one-way roads.

Suppose that we seek a shortest path from each node to node 4 say above. Clearly an optimal route is optimal from any intermediate node onwards. For example 1, 2, 3, 4 is a shortest path from node 1 to node 4, so 2, 3, 4 must be a shortest path from node 2 to node 4. In general, in sequential decision processes, the *optimality principle* states that optimal policies are optimal from any intermediate state onwards.

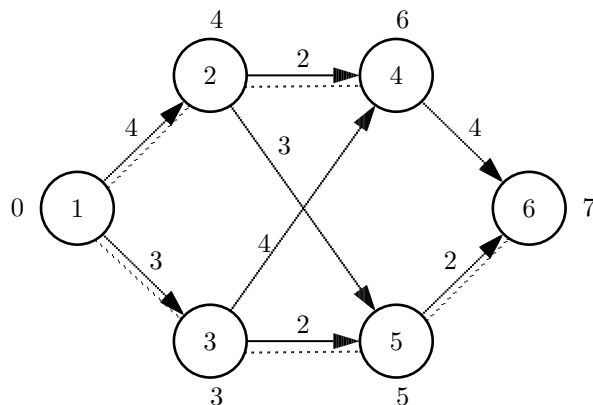
We next introduce two shortest path algorithms.

#### (a) Acyclic networks

Suppose that  $G$  contains no cycles (following the direction of the arcs). Then we can number the nodes  $1, 2, \dots, n$  such that arcs can only go from small numbers to big numbers.



We can do the numbering as follows. A *source* is a node with no incoming arcs, like the node on the left above. Since  $G$  is acyclic, if we keep tracing back along arcs we must reach a source. Label some source 1 and remove it from  $G$ . Now find a source in the remaining graph, label it 2 and continue. We can do this in  $O(n^2)$  steps, if we keep track of the number of arcs entering each node.



Suppose now that the nodes have been numbered as above. We may determine shortest paths from node 1 to each other node, in one pass.

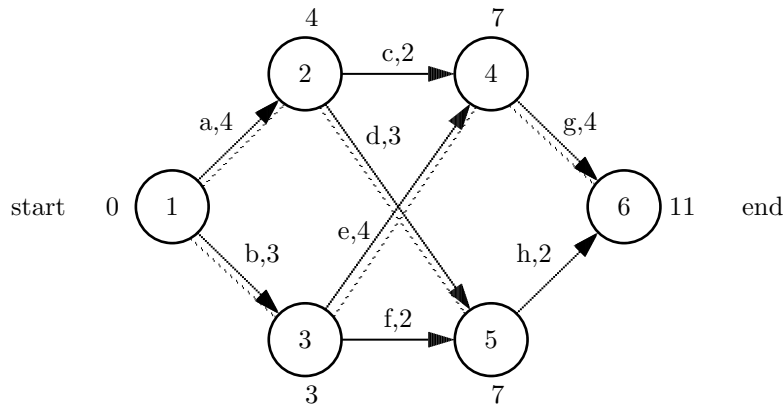
1. Let  $u_1 = 0$ .
2. For  $j = 2, \dots, n$  let  $u_j = \min\{u_k + a_{kj}\}$  where the minimum is over all  $k$  such that  $1 \leq k < j$  and  $kj$  is an arc.

This determines the shortest path lengths  $u_j$  in  $O(n^2)$  steps, and by keeping track of the calculations we also find a 'tree' of shortest paths from node 1. To determine **longest** paths we may proceed similarly with min replaced by max.

Suppose that a complicated project can be divided into a number of activities. Each activity takes a certain time, and cannot be started until certain other activities are completed (for example we cannot build the walls of a house until we have laid the foundations).

Activity	a	b	c	d	e	f	g	h
Immediate Predecessors	-	-	a	a	b	b	c,e	d,f
Duration (days)	4	3	2	3	4	2	4	2

We may represent this 8-activity project by the network below.



To determine a longest 1-6 path (start to end), let  $u_1 = 0$  and for  $j = 2, \dots, 6$  let  $u_j = \max\{u_k + a_{kj}\}$ , where the maximum is over all  $k$  such that  $1 \leq k < j$  and  $kj$  is an arc.

The numbers  $u_j$  are shown in the figure, together with arcs where the maxima are attained. We thus find that the minimum project duration is 11 days: for each node may be reached at time  $u_j$  after the start, and no earlier. Also, the 'critical path' is  $b, e, g$ , so that if any of these activities overruns then the whole project is delayed. Ideas such as these have been developed into the critical path method (CPM) or the project evaluation and review technique (PERT) much used in the planning and control of large projects.

(b) **General networks** (with all lengths  $a_{ij} \geq 0$ )

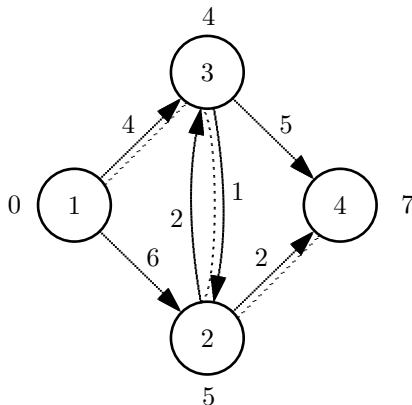
Suppose that again we wish to find all shortest paths from node 1. We are given an  $(n \times n)$  matrix of lengths  $a_{ij} \geq 0$ , where we set  $a_{ij} = \infty$  if there is no arc  $ij$ .

We shall present an efficient algorithm due to Dijkstra, which works in stages. At each stage we shall have a number  $u_j$  for each node  $j$  and a set  $P$  (for permanent) of nodes with  $T = N \setminus P$  (for temporary) such that (a) for each node  $j$  in  $P$ ,  $u_j$  is the minimum length of a  $1 - j$  path; and (b) for each node  $j$  in  $T$ ,  $u_j$  is the minimum length of a  $1 - j$  path with only its last node in  $T$ .

**Dijkstra's algorithm**

- Set  $u_1 = 0$ ,  $u_j = a_{1j}$  for  $j \in N \setminus \{1\}$ ,  $P = \{1\}$ ,  $T = N \setminus \{1\}$ .
- While  $T$  is non-empty do
  - find a node  $k$  in  $T$  with  $u_k = \min_{j \in T} \{u_j\}$
  - move  $k$  from  $T$  to  $P$
  - for each  $j$  in  $T$ , if  $u_j > u_k + a_{kj}$  then set  $u_j = u_k + a_{kj}$ .

**Example**



Iteration	Node				P
	1	2	3	4	
1	0	6	4	$\infty$	1
2		5	4	9	1,3
3		5		7	1,2,3
4				7	1,2,3,4

To see why this method works, let us check that when the label on node  $k$  is made permanent, the value  $u_k$  is indeed the minimum length of a  $1 - k$  path. Consider any  $1 - k$  path  $Q$ : we must check that the length of  $Q$  is at least  $u_k$ . But the length of  $Q$  is at least the length of the section  $Q'$  up to the first node  $j$  in  $T$ ; this must be at least  $u_j$  (since  $Q'$  has only its last node in  $T$ , see (b) above); and finally  $u_j \geq u_k$  by our choice of  $k$ .

**5.2 Transportation problems with fixed costs**

Many complex problems may be modelled if we allow integer-valued variables. Consider for example the transportation problem

$$\begin{aligned}
 & \min \sum_i \sum_j c_{ij} x_{ij} \\
 & \text{subject to} \\
 & \sum_j x_{ij} \leq s_i \quad \text{for each factory } F_i \\
 & \sum_i x_{ij} \geq d_j \quad \text{for each market } M_j \\
 & x_{ij} \geq 0 \quad \text{for each } i \text{ and } j.
 \end{aligned}$$

Here  $i$  runs from 1 to  $m$  and  $j$  from 1 to  $n$ . The unit cost  $c_{ij}$  includes the transportation cost from factory  $F_i$  to market  $M_j$  plus the variable production

cost at factory  $F_i$ . Now let each factory  $F_i$  incur a fixed overhead cost  $K_i \geq 0$  if it is operated. Also let there be a fixed set-up cost  $K_{ij} \geq 0$  for using the route from factory  $F_i$  to  $M_j$ . We wish to choose which factories to operate and which routes to use in order to minimise total (annual) costs.

We may formulate this problem as follows. Introduce a 0-1 variable  $y_i$  for each possible factory  $F_i$  and  $z_{ij}$  for each possible route  $F_i$  to  $M_j$ . Setting  $y_i = 1$  will mean that factory  $F_i$  will be operated. Similarly the  $z_{ij}$  will indicate which routes will be used. We obtain the mixed integer linear programme

$$\begin{aligned} \min \quad & \sum_i K_i y_i + \sum_i \sum_j (c_{ij} x_{ij} + K_{ij} z_{ij}) \\ \text{subject to} \quad & \sum_j x_{ij} - s_i y_i \leq 0 && \text{for each factory } F_i \\ & \sum_i x_{ij} \geq d_j && \text{for each market } M_j \\ & x_{ij} - b_{ij} z_{ij} \leq 0 && \text{for each } i, j \text{ where } b_{ij} \\ & && \text{is a large constant, for} \\ & && \text{example } b_{ij} = \min(s_i, d_j) \\ & x_{ij} \geq 0, z_{ij} = 0 \text{ or } 1 && \text{for each } i, j \\ & y_i = 0 \text{ or } 1 && \text{for each } i. \end{aligned}$$

If say we also insist that at least one of factories  $F_1, F_2, F_3$  be operated but not both  $F_1$  and  $F_2$  then we add the constraints

$$y_1 + y_2 + y_3 \geq 1, y_1 + y_2 \leq 1.$$

Suppose that say when factory  $F_1$  is operated it must output at least  $s'_1 (\leq s_1)$  units. Then we add the constraint

$$\sum_j x_{1j} - s'_1 y_1 \geq 0.$$

### 5.3 Branch-and-bound

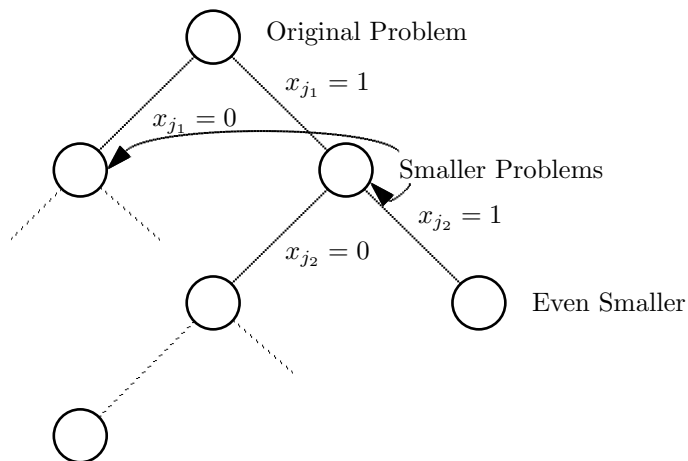
We have seen that allowing 0-1 valued variables is good for modelling. However, we may be unable to solve problems with many such variables. We may try a 'branch-and-bound' approach.

Consider a mixed integer linear programme

$$\begin{aligned} \max \quad & z = c'x \\ \text{subject to} \quad & Ax = b \\ & x_j = 0 \text{ or } 1 \quad \text{for } j = 1, \dots, p \\ & x_j \geq 0 \quad \text{for } j = p + 1, \dots, n. \end{aligned}$$

Here  $A$  is an  $(m \times n)$  matrix as usual.

Suppose that we know some feasible solution  $\hat{x}$  with corresponding value  $\hat{z} = c'\hat{x}$ , so that  $\hat{z}$  is a lower bound on the optimum value. We update the quantities  $\hat{x}$ ,  $\hat{z}$  as we proceed.



Consider a subproblem obtained by fixing various variables  $x_j$  at 0 or 1. Relax this problem to a linear programme (that is, replace any constraints  $x_j = 0$  or 1 (where  $x_j$  has not been fixed) by  $0 \leq x_j \leq 1$ ) and solve (quickly) yielding  $x^*$ ,  $z^*$ . There are three cases to consider.

- (a) If  $z^* \leq \hat{z}$  then 'prune' — we need not consider this subproblem further.
- (b) If  $z^* > \hat{z}$  and each  $x_j^*$  is 0 or 1 where required then update  $\hat{x}$ ,  $\hat{z}$ .
- (c) If neither (a) nor (b) hold then  $z^* > \hat{z}$  and  $0 < x_{j_o}^* < 1$  for some  $x_{j_o}$  that should be  $\{0, 1\}$ -valued. Create two subproblems (sons) one with  $x_{j_o}$  fixed = 0 and one with  $x_{j_o}$  fixed = 1.

When we have grown our tree until no more 'leaves' want to branch as in (c), then the final  $\hat{x}$  is an optimal solution, with value  $\hat{z}$ . We hope that the pruning in (a) will stop the search tree from growing too large.