

Shortest Path Problems

A digraph  $D = (N,A)$  consists of 2 sets:

$N$  = the set of nodes

$A \subseteq N \times N$  is the set of arcs

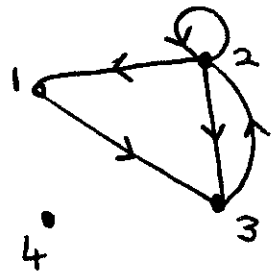


Fig 1

The above is a pictorial representation of the digraph with

$$N = \{1,2,3,4\}$$

$$A = \{(1,3), (2,1), (2,2), (2,3), (3,2)\}$$

We will only consider digraphs with  $N$  and  $A$  finite.

A walk  $W$  from a node  $i_1$  to a node  $i_p$  is a sequence of arcs  $((i_1, i_2), (i_2, i_3), \dots, (i_{p-1}, i_p))$  or equivalently a sequence of nodes  $(i_1, \dots, i_p)$  where  $(i_{t-1}, i_t) \in A$  for  $2 \leq t \leq p$ .

Although  $W$  is not a set we use the notation  $u \in W$  or  $i \in W$  to say arc  $u$  is in  $W$  or node  $i$  is in  $W$ .

Examples:  $(1,3,2)$  and  $(2,2,3,2,1)$  are walks in the digraph give in fig. 1.

If  $W = (i_1, \dots, i_p)$  and  $1 \leq a < b \leq p$  we use the notation

$$W[a,b] = (i_a, i_{a+1}, \dots, i_b)$$
 for the sub-walk of  $W$  from  $i_a$  to  $i_b$ .

Next given a walk  $W_1 = (i_1, \dots, i_p)$  and a walk  $W_2 = (i_p = j_1, j_2, \dots, j_p)$  we define the walk

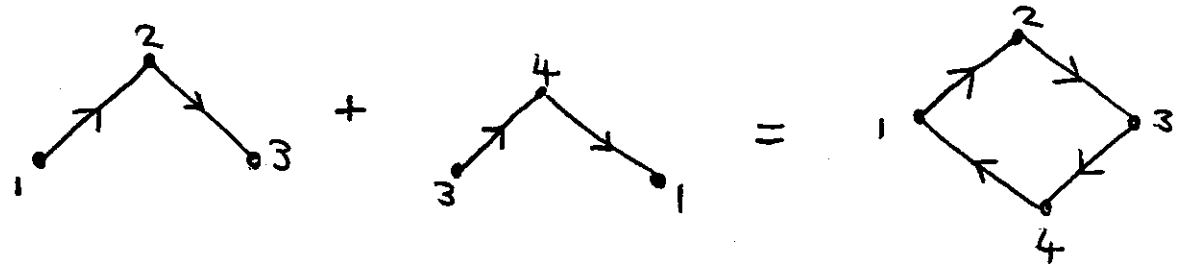


Fig 2

$$W_1 + W_2 = (i_1, \dots, i_p, j_2, \dots, j_p).$$

(We can only form  $W_1 + W_2$  if the terminal node of  $W_1 =$  the initial node of  $W_2$ ).

A Path  $P$  is a walk in which no node is visited more than once.

A circuit is a walk from a node to itself e.g.  $(1,2,3,4,1)$  of fig. 2 is a circuit.

### Length

We now assume that associated with each arc  $u \in A$  is a length  $\ell(u)$   
i.e.:  $\ell: A \rightarrow \mathbb{R}$ .

The length of a walk  $W$  is then defined by

$$\ell(W) = \sum_{u \in W} \ell(u)$$

i.e. the length of a walk is the sum of the lengths of the arcs in the walk.

We shall be concerned here with the following problem: given a node  $s \in N$ , find for each node  $j \neq s$  a minimum length path from  $s$  to  $j$ .

Because of an assumption we will make about the non-existence of negative circuits we will be able to show that a shortest path from  $s$  to  $j$  is also a shortest walk from  $s$  to  $j$ .

### Negative Circuits

If some arc lengths are negative it is possible that there is a circuit

$C$  such that  $\ell(C) < 0$ . We exclude this possibility for the following reason:

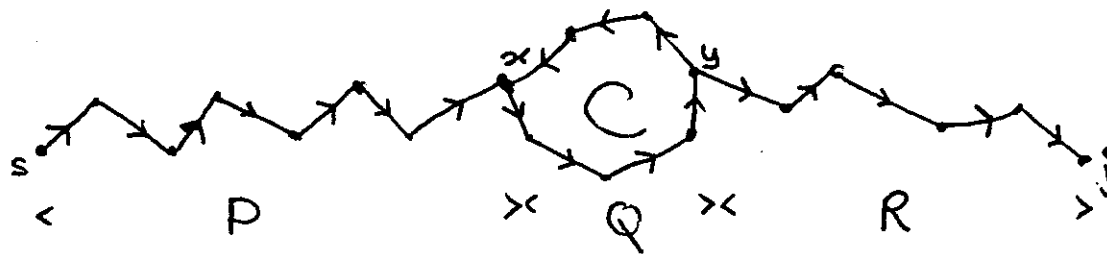


Fig 3

Suppose  $\ell(C) < 0$ . Define walk  $W_n$  to consist of (the path  $P$  from  $s$  to  $x$ ) + (the path  $Q$  from  $x$  to  $y$ ) + ( $n$  times round  $C$ ) + (the path  $R$  from  $y$  to  $j$ ). Then  $\ell(W_n) = \ell(P) + \ell(Q) + n\ell(C) + \ell(R)$

$$\rightarrow -\infty \text{ as } n \rightarrow \infty.$$

Thus there is no shortest walk from  $s$  to  $j$ .

Since the number of paths from  $s$  to  $t$  is finite ( $< 2^{|A|}$ ) there is always a shortest path from  $s$  to  $t$  but there are no known polynomial algorithms for finding shortest paths if there are negative circuits. This is essentially because they rely on

### Theorem 2.1

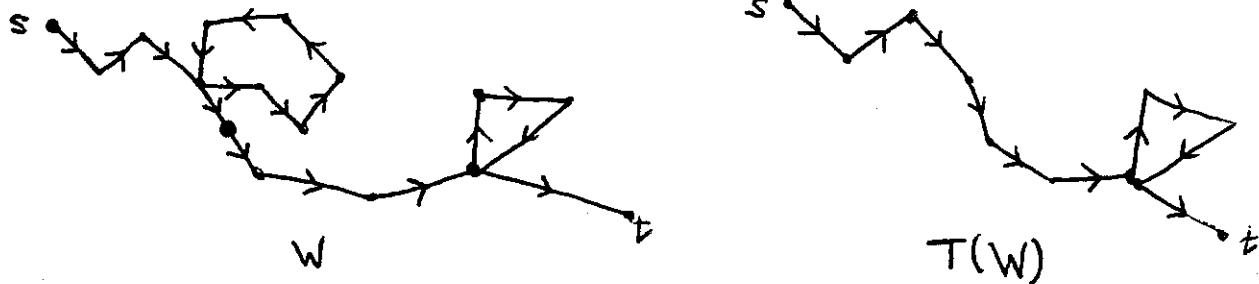
If  $D$  has no negative circuits then a shortest path from node  $s$  to node  $t \neq s$  is a shortest walk from  $s$  to  $t$ .

### Proof

Let  $W$  be any walk from  $s$  to  $t$  and let  $P^*$  be a shortest path from  $s$  to  $t$ . We can construct a path  $P$  from  $s$  to  $t$  such that  $\ell(W) \geq \ell(P)$ . As  $\ell(P) \geq \ell(P^*)$  we have  $\ell(W) \geq \ell(P^*)$  and our theorem.

Construction of P

If  $W$  is a path let  $P = W$ , otherwise suppose  $W = (s = i_1, \dots, i_p = t)$ .  
 Let  $i_a = i_b$  be the first repeated node. Let  $T(W)$  be the walk  $W[1,a] + W[b,p]$ .



Then  $T(W)$  is a walk from  $s$  to  $t$  with fewer edges than  $W$  and

$$\begin{aligned} \ell(T(W)) &= \ell(W) - \ell(W[a,b]) \\ &\leq \ell(W) \quad \text{as } W[a,b] \text{ is a circuit.} \end{aligned}$$

If  $T(W)$  is not a path we construct  $T^2(W)$  and so on. Thus there exists  $k \geq 0$  such that  $T^k(W)$  is a path. Let  $P = T^k(W)$ . Then  
 $\ell(P) \leq \ell(T^{k-1}(W)) \leq \dots \leq \ell(W)$ . □

Properties of Shortest PathsTheorem 3.1 (Optimality of sub-paths)

Let  $P = (i_1 = s, \dots, i_p = t)$  be a shortest path from  $s$  to  $t$ . Then for  $1 \leq a < b \leq t$   $P[a,b]$  is a shortest path from  $i_a$  to  $i_b$ .

Proof

Let  $R$  be any path from  $i_a$  to  $i_b$ . Let  $W = P[1,a] + R + P[b,t]$ .  $W$  is a walk from  $s$  to  $t$ . We know that

$$\begin{aligned} 0 &\geq \ell(W) - \ell(P) && [P \text{ is also a shortest walk}] \\ &= \ell(R) - \ell(P[a,b]). \end{aligned}$$

□

Suppose next that for each  $j \in N$  we have a path  $P_j$  from  $s$  to  $j$ . ( $P_s = (s)$ ) and that  $d(j) = \ell(P_j)$  ( $d(s) = 0$ ).

Theorem 3.2

$\{P_j; j \in N\}$  is a collection of shortest paths from  $s$  to each node of  $D$  if and only if  $\forall i,j \in N$  we have

$$(3.1) \quad d(j) \leq d(i) + \ell(i,j) \quad \forall (i,j) \in A.$$

Proof

only if: Suppose 3.1 does not hold and that there exist nodes  $x,y$  and arc  $(x,y)$  such that

$$d(y) > d(x) + \ell(x,y).$$

Consider  $W = P_x + (x,y)$ .  $W$  is a walk from  $s$  to  $y$  and  $\ell(W) = d(x) + \ell(x,y) < \ell(P_y)$ . This contradicts the fact that  $P_y$  is a shortest path.

If: Suppose 3.1 holds  $\forall j \in N$ . Let  $x \in N$  and  $P = (s = i_1, \dots, i_p = x)$  be a path from  $s$  to  $x$ . We show that  $\ell(P) \geq d(x)$ .

From 3.1 we have  $d(i_k) - d(i_{k-1}) < \ell(i_{k-1}, i_k)$   $2 \leq k \leq p$ . Thus

$$(3.2) \quad \sum_{k=2}^p (d(i_k) - d(i_{k-1})) \leq \sum_{k=2}^p \ell(i_{k-1}, i_k).$$

But the LHS of 3.2 "collapses" to  $d(i_k) - d(i_1) = d(x) - 0$  and the RHS of 3.2 is  $\ell(P)$ . □

The aim of all shortest path algorithms is to find a set of paths satisfying 3.1.

Ford's Algorithm

Suppose that  $D$  is given as a list of arcs and  $A = \{u_1, \dots, u_m\}$  where  $u_i = (x_i, y_i)$ .

We will first consider how to compute the lengths of shortest paths and then show how to produce paths.

At a general stage of the algorithm we will have estimates  $d(j)$  for the shortest path length to  $j \in N$ . Initially  $d(s) = 0$  and  $d(j) = \infty$  for  $j \in N \setminus \{s\}$ .

The idea behind Ford's algorithm stems from theorem 3.2: if we have an arc  $(x, y)$  such that  $d(y) > d(x) + \ell(x, y)$  then replace  $d(y)$  by  $d(x) + \ell(x, y)$ .

Ford's Algorithm (ignore statements on first reading)

begin

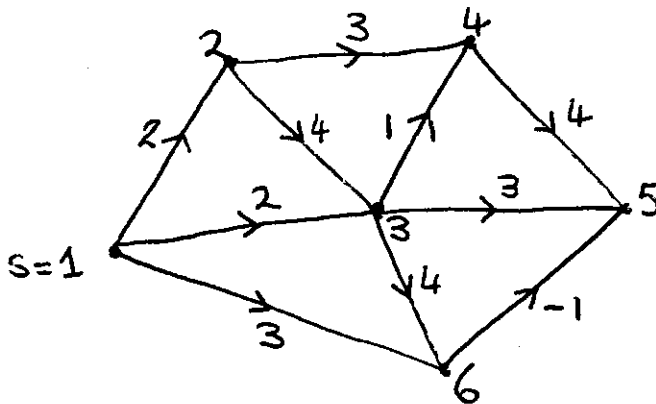
{Initialization: assume  $s = 1$  and  $|N| = n$ ,  $|A| = m$ }

$d(1) := 0$ ; for  $j := 2$  to  $n$  do [ $d(j) := \infty$ ;  $\pi(j) := s$ ];

```

repeat {main loop}
  flag := false
  for a := 1 to m do
    begin {process arc (xa, ya)}
      if d(ya) > d(xa) + l(xa, ya) then
        begin
          d(ya) := d(xa) + l(xa, ya); π(ya) := xa;
        end
      end
    end
  until flag = false
end

```

Example

Arc	d(1)	d(2)	d(3)	d(4)	d(5)	d(6)
-	0	∞	∞	∞	∞	∞
(1,2:2)	0	2	∞	∞	∞	∞
(1,3:2)	0	2	2	∞	∞	∞
(1,6:3)	0	2	2	∞	∞	∞
(2,3:4)	0	2	2	∞	∞	∞
(2,4:3)	0	2	2	5	∞	3
(3,4:1)	0	2	2	3	∞	3
(3,5:3)	0	2	2	3	5	3
(3,6:4)	0	2	2	3	5	3
(4,5:4)	0	2	2	3	5	3
(6,5:1)	0	2	2	3	2	3

---

 END OF 1ST PASS

There were no changes during second pass

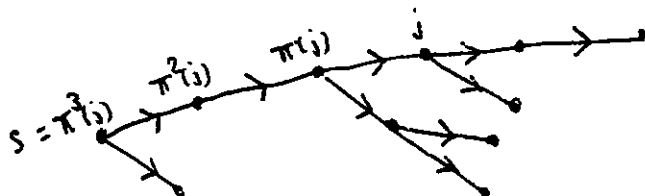
---

 END OF 2ND PASS



### Recording the shortest paths

We will show that the set of paths produced by the algorithm forms a directed tree rooted at s.



i.e. for each  $j \in N$  we record a predecessor node  $\pi(j)$  so that on termination of the algorithm we have the following:

$$(5.1a) \quad \pi(s) = s$$

$$(5.1b) \quad \text{for } j \neq s \exists k > 0 \text{ such that } s = \pi^k(j) \text{ and}$$

$$P_j = (s = \pi^k(j), \pi^{k-1}(j), \dots, \pi^2(j), \pi(j), j)$$

is a shortest path from  $s$  to  $j$ .

The statements of Ford's algorithm actually find shortest paths.

#### Lemma 5.1

Throughout the algorithm if  $d(j) \neq \infty$  then  $d(j)$  is the length of some walk from  $s$  to  $j$ .

Proof

The statement of true initially. We show that processing an arc does not alter the statements truth. Suppose that immediately prior to processing arc  $(x,y)$  that  $d(x) = \ell(W_x)$  or  $\infty$  and  $d(y) = \ell(W_y)$  or  $\infty$  for walks  $W_x, W_y$ . After processing  $(x,y)$  only  $d(y)$  can be altered and then  $d(y) = \ell(W_y)$  or  $\ell(W_x + (x,y))$  or  $\infty$ .  $\square$

During execution of the algorithm let label  $d(j)$  be correct if  $d(j) =$  the length of a shortest path from  $s$  to  $j$ .

Note that once  $d(j)$  is correct it does not change anymore. If it did change it would be reduced and then lemma 5.1 would imply the existence of a walk shorter than the shorest path which contradicts theorem 2.1.

For  $0 \leq k \leq |V| - 1$  let

$$H(k) = \{j \in N : \exists \text{ a shortest path from } s \text{ to } j \text{ which has } k \text{ arcs or less}\}$$

Lemma 5.2

After  $k$  passes through the main loop all nodes in  $H(k)$  have correct  $d$  labels.

Proof

By induction on  $k$ . For  $k = 0$  the result is true because  $H(0) = \{s\}$ . Suppose this result is true for all  $k < K$ . Let  $j \in H(K) \setminus H(K-1)$  and let  $P = (s = i_1, \dots, i_k = j)$  be a shortest path from  $s$  to  $j$ . By theorem 3.1  $i_{k-1} \in H(K-1)$  and so its label is correct at the beginning of the  $K^{\text{th}}$  main loop. During execution of the  $K^{\text{th}}$  main loop arc  $(i_{K-1}, j)$  will be processed and so at the end of the  $K^{\text{th}}$  main loop  $d(j) \leq d(i_{K-1}) + \ell(i_{K-1}, j) = \ell(P)$  and then

lemma 5.1 and theorem 2.1 imply  $d(j) = \ell(P)$ .

Now  $H(|N| - 1) = \{j : \exists \text{ a path from } s \text{ to } j\}$ . We therefore have

### Theorem 5.1

Ford's algorithm terminates after at most  $|N| - 1$  iterations, having computed the shortest distance from  $s$  to each  $j \in N$ .  $\square$

It is easily seen that the computation time for the main loop is bounded by some multiple of  $|A|$ . Thus the overall computation time of Ford's algorithm is  $O(|N| \times |A|)$ .

We have still to verify that on completion  $\pi$  provides shortest paths.

### Lemma 5.3

On completion of the algorithm the  $\pi$  labels are correct.

### Proof

On completion we have

$$(5.2) \quad d(j) = d(\pi(j)) + \ell(\pi(j), j) \quad \forall j \in N.$$

This is because there will have been no further reduction in  $d(\pi(j))$  after the last assignment to  $d(j)$ .

If we can show that (5.1) holds it will follow from (5.2) that  $d(j) = \ell(P_j)$  and we are through.

Fix  $j \in N$  and consider the sequence  $j, \pi(j), \pi^2(j), \dots$ . We have to show that  $\exists k$  such that  $\pi^k(j) = s$ . If this is not true then  $\exists \ell < m$  such that  $\pi^\ell(j) = \pi^m(j) \neq s$ .

Now for  $i \in N \setminus \{s\}$  let  $T(i)$  be the number of arc processings up to and

including the processing of arc  $(\pi(i), i)$  that gave  $d(i)$  its final value. Let  $T(s) = 0$ . Now  $i \neq s$  implies  $T(i) > T(\pi(i))$  because  $d(i)$  is not made correct until after  $d(\pi(i))$  is made correct.

Thus  $T(\pi^\ell(j)) > T(\pi^{\ell+1}(j)) > \dots > T(\pi^m(j))$  which contradicts  $\pi^\ell(j) = \pi^m(j)$ . □

### A Computational Improvement

Consider the following situations we are about to process arc  $(x, y)$ . This arc has been processed before but  $d(x)$  has not been reduced since  $(x, y)$  was last processed. Thus we know that prior to processing  $(x, y)$  that  $d(y) \leq d(x) + \ell(x, y)$  and so in fact there is no point in processing  $(x, y)$ .

We can speed up execution of the algorithm if we obey the following rule: arc  $(x, y)$  is processed only if  $d(x)$  has been reduced since arc  $(x, y)$  was last processed.

In the main loop arcs are processed in blocks. A block consists of all the arcs leaving a specific node.

The search for nodes whose  $d$  labels have been reduced since their blocks were last processed is speeded up by keeping them in a queue  $Q$ .

A queue is a linked list of nodes where insertions are made at the "back end" and deletions are from the "front end" only.

Ford's algorithm (final version)

procedure process (node : x); {process all the arcs leaving x}

begin

for (x,y)  $\in$  A do

begin

if  $d(x) + \ell(x,y) < d(y)$  then

begin

$d(y) := d(x) + \ell(x,y)$ ;  $\mathcal{P}(y) := x$ ;

if y is not in Q then insert y into Q

end

end

end;

{initialization}

begin

$d(1) = 0$ ;  $\pi(1) := 1$ ; for j = 2 to n do [ $d(j) := \infty$ ;  $\pi(j) := 1$ ]

Q :=  $\{\cdot, 1\}$  ;

while Q  $\neq \phi$  do

begin

delete x = front(Q) from Q.

process (x)

end

end

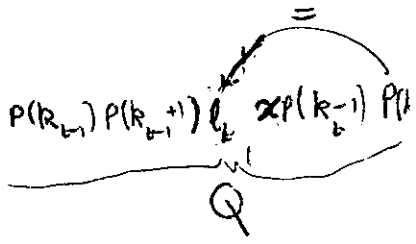
Example (see digraph on p8)

		$\alpha(j) \overset{j}{\dashv} \pi(j)$												
		1	2	3	4	5	6	1	2	3	4	5	6	Q
0	1	$\infty$	1	$\infty$	1	$\infty$	1	$\infty$	1	$\infty$	1	$\infty$	1	1
0	1	2	1	2	1	$\infty$	1	$\infty$	1	3	1	3	1	2-3-6
0	1	2	1	2	1	5	2	$\infty$	1	3	1	3	1	3-6-4
0	1	2	1	2	1	3	3	5	3	3	1	3	1	6-4-5
0	1	2	1	2	1	3	3	4	3	3	1	3	1	4-5
0	1	2	1	2	1	3	3	4	3	3	1	3	1	5
0	1	2	1	2	1	3	3	4	3	3	1	3	1	$\phi$

To prove that the modification is valid we relate the new algorithm to the old.

Let  $p(1), p(2), \dots$  be the sequence of nodes processed by the new algorithm. Define  $k_0 = 0$  and  $k_1, \ell_1, \dots, k_t, \ell_t, \dots$  as follows:  
 $p(1), \dots, p(k_1-1)$  are all distinct but  $p(k_1) = p(\ell_1)$  where  $1 \leq \ell_1 \leq k_1 - 1$   
 $p(k_1), \dots, p(k_2-1)$  are all distinct but  $p(k_2) = p(\ell_2)$  where  $k_1 \leq \ell_2 \leq k_2-1$   
 and so on.

Let  $X_t = \{p(k_{t-1}), \dots, p(k_t-1)\}$ ,  $Y_t = \{x \in X_t \text{ and } x \text{ is not in } Q \text{ immediately prior to the } \ell_t \text{th node processing}\}$ .

Exercise: show that  $N = X_t \cup Y_t$ . Suppose  $x \in \overline{X_t} \cap Q$  - 

[Hint: it is a simple direct consequence of Q being a queue]

Suppose now we re-ran the new algorithm but just before we process  $p(\ell_t)$  we process all the nodes in  $Y_t$ . Nothing will happen to  $d$  or  $\pi$  because none of  $Y_t$  are in  $Q$ . But the exercise shows that now between processing  $p(k_{t-1})$  and  $p(k_t-1)$  all nodes and hence all arcs are processed, i.e. we have gone through a main loop of the old algorithm (the order in which we process arcs in a main loop is irrelevant). Thus convergence in  $O(|N| \times |A|)$  time is assured.

Exercise: show that we do one less main loop in the new method.

The algorithm above is very efficient. We give a table of reported results for some random problems. The run time  $t$  is in seconds on a COC 6600

Problem	N	A	t	Problem	N	A	t
1	500	12500	.267	9	1000	4000	.139
2	500	7500	.177	10	1000	3000	.123
3	500	2500	.089	11	1000	2000	.088
4	500	1250	.050	12	2000	16000	.476
5	500	1000	.045	13	2000	12000	.371
6	500	750	.039	14	2000	8000	.315
7	1000	10000	.288	15	2000	4000	.191
8	1000	5000	.165				

Exercise: the initialization step wastes a little time, what should it be?

Department of Mathematics  
 CARNEGIE MELLON UNIVERSITY

Dijkstra's algorithm

When arc lengths are all non-negative the following algorithm is applicable.

For  $k \in N$  let  $\Gamma_k = \{v: (k,v) \in A\}$  = set of out-neighbours of  $k$ .

Dijkstra's algorithm

begin

A:  $d(1) := 0$ ;  $\pi(1) := 1$ ; for  $j := 2$  to  $n$  do [ $d(j) := \ell(s,j)$ ;  $\pi(j) := s$ ]

$S := \{1\}$  [ $S = \{\text{nodes that have been processed}\}$ ]

for  $i := 1$  to  $n-2$  do

begin

B: let  $d(k) = \min (d(j): j \in N \setminus S)$  ;

$S := S \cup \{k\}$  ;

{process  $k$ }

C: for  $v \in \Gamma_k - S$  do

if  $d(v) > d(k) + \ell(k,v)$  then

begin

$d(v) := d(k) + \ell(k,v)$ ,

$\pi(v) := k$

end

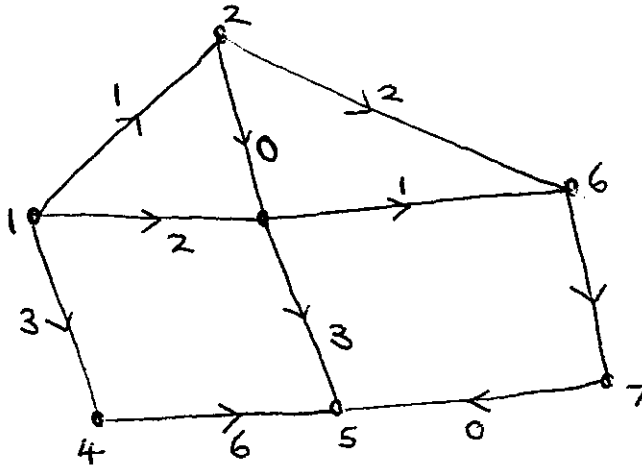
end

end

A nice feature of Dijkstra's algorithm is that each node is processed exactly once (except for last node which need not be processed).

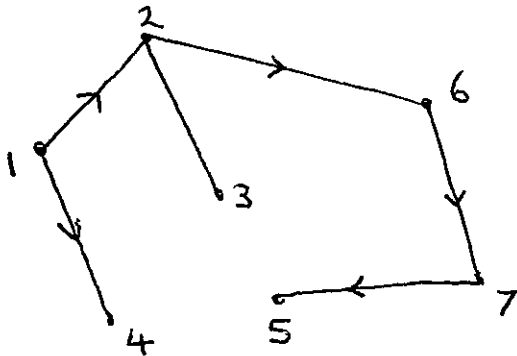


Example



	1	2	3	4	5	6	7	k
0	1	1 1	2 1 1 2	3 1 3 1 3 1 3 1	$\infty$ 1 $\infty$ 1 4 3 4 3 4 3 3 7	$\infty$ 1 3 2 2 3	$\infty$ 1 $\infty$ 1 $\infty$ 1 3 6 3 6	2 3 6 4 7

shortest path tree



Theorem

Assuming that all arc lengths are non-negative, Dijkstra's algorithm terminates with a shortest path from  $s$  to each node of  $D$ .

Proof

At each stage the digraph  $(S, X_S)$  where  $X_S = \{(\pi(j), i) : j \in S - \{s\}\}$  is a directed tree rooted at  $s$  and for  $j \in S$ ,  $d(j)$  is the length of the path from  $s$  to  $j$  in this tree. Furthermore if  $j \notin S$  then  $d(j)$  is the minimum length of a path from  $s$  to  $j$ , which follows a tree path in  $S$  and then jumps to  $j$ .

We prove by induction on  $|S|$  that throughout the algorithm  $j \in S$  implies  $d(j)$  is the length of a shortest path from  $s$  to  $j$ .

This is clearly true when  $|S| = 1$  and  $S = \{s\}$ . Suppose it is true for  $|S| < q$  and suppose  $k$  is the  $q^{\text{th}}$  node added to  $S$ . Let  $P = (s = i_1, i_2, \dots, i_a = k)$  be any path from  $s$  to  $k$  and let  $i_b$  be the first node of  $P$  that is not in  $S - \{k\}$ . Then

$$\begin{aligned} \ell(P) &\geq \ell(P[1, b]) && \text{[arc lengths non-negative]} \\ &\geq d(i_{b-1}) + \ell(i_{b-1}, i_b) \\ &\geq d(i_b) && \text{[See first paragraph]} \\ &\geq d(k) && \text{[definition of } k \end{aligned}$$

□

Execution Time

A naive implementation is  $O(n^2)$ .

Line A is  $O(n)$  and is executed once

Line B is  $O(n)$  and is executed  $n-1$  times

Line C is  $O(|\Gamma_k|)$  and total execution time is  $O(\sum_k |\Gamma_k|) = O(|A|) = O(n^2)$ .

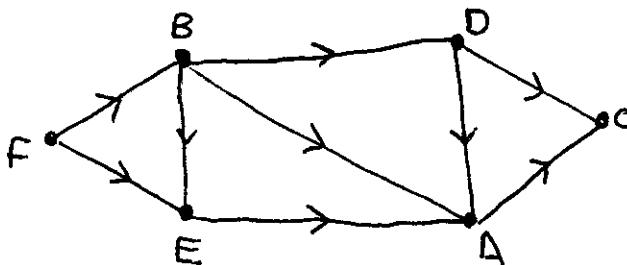
### Digraphs without circuits

These are important not least because they occur in critical path analysis. Their application in this area involves computing longest paths. For this problem inequalities are reversed throughout the previous sections but most important the optimality conditions (3.1) becomes

$$(7.1) \quad d(j) = \max_j (d(i) + \ell(i,j)).$$

### topological Ordering

Let the nodes of digraph  $G = (N,A)$  be ordered or numbered  $1,2,\dots,n$ . This ordering is topological if  $(i,j) \in A \rightarrow i < j$



F B E D A C is a topological ordering for the above digraph.

### Theorem 7.1

There exists a topological ordering for a digraph  $G$  if and only if the digraph does not contain any circuits.

### Proof

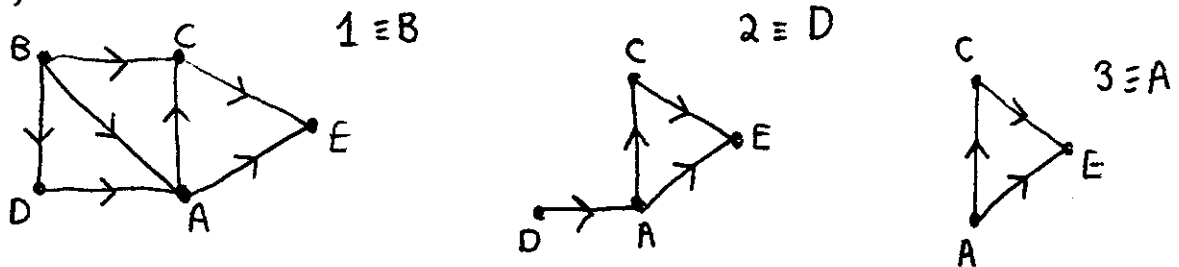
Suppose first that the nodes of  $G$  have been topologically ordered  $1,2,\dots,n$ . Suppose  $G$  has a circuit  $(i_1, i_2, \dots, i_p = i_1)$  then we have

$$i_1 < i_2 < \dots < i_p = i_1 \quad \text{contradiction.}$$

Conversely assume  $G$  has no circuits. We describe an algorithm for numbering the nodes of  $G$ . It's general step is: if nodes  $1, 2, \dots, k$  have been chosen define  $G_k = (N_k, A_k)$  where  $N_k = N - \{1, 2, \dots, k\}$

$$A_k = A \cap N_k \times N_k.$$

Note  $G_0 = G$  and  $A_k$  consists of these arcs not involving  $1, 2, \dots, k$ . Now since  $G$  has no circuits  $G_k$  will not have any for  $k = 0, 1, \dots, n-1$ . Thus (Lemma 7.1 below)  $G_k$  has at least one node without predecessors. Let  $k+1$  be such a node (a predecessor of a node  $j$  is a node  $i$  such that  $(i, j) \in A$ ).



The algorithm above will number the nodes  $1, 2, \dots, n$ . Now let  $(i, j) \in A$ . From the way  $j$  was chosen from  $G_{j-1}$  we see that  $(i, j) \in A_{j-1}$  which implies that  $i \in \{1, 2, \dots, j-1\}$  or  $i < j$ .  $\square$

### Lemma 7.1

Let  $G = (N, A)$  be a digraph without circuits. Then  $G$  contains a node without predecessors.

Proof

Let  $x_1 \in N$ . If  $x_1$  has no predecessors we are finished otherwise there is an arc  $(x_2, x_1)$ . If  $x_2$  has no predecessors we are finished, otherwise there is an arc  $(x_3, x_2)$ . Continuing thus we generate a sequence  $x_1, x_2, \dots$  where  $(x_{k+1}, x_k) \in A$  for  $k \geq 1$ . This sequence must terminate with a node without predecessors or repeat a node because  $G$  is finite. But repeating a node means there is a circuit in  $G$ .  $\square$

Assume now that we have topologically ordered the nodes of a digraph  $G$ . Then (1.9) can be replaced by

$$(7.2) \quad d(j) = \max_{i < j} (d(i) + \ell(i, j)) \quad j = 1, \dots, n$$

where it is assumed that  $s = 1$  (what if  $s \neq 1$ ?) Given that  $d(1) = 0$  we can use (7.2) to compute  $d(2), \dots$  in a manner analogous to back-substitution for solving a lower triangular set of linear equations

begin

$d(1) := 0;$

for  $j := 2$  to  $n$  do

$d(j) := \max (d(i) + \ell(i, j): (i, j) \in A)$  \*

end

(Note that because  $(i, j) \in A$  implies  $i < j$  \* can be carried out validly.)

Since (7.1) holds on completion of the algorithm a proof of validity is obvious.

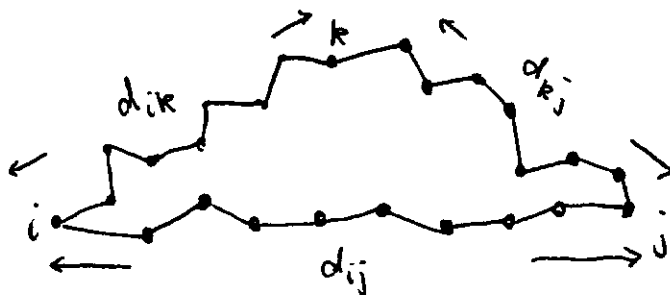
#### Computational Considerations

The complexity of the algorithm is  $O(|A|)$  and there exists an  $O(|A|)$  method for topologically ordering the nodes.

### Shortest paths between all pairs of nodes

We complete the material on shortest paths by describing an algorithm for finding shortest paths between all pairs of nodes.

We use the two matrices  $D = \|d_{ij}\|$ ,  $N = \|n_{ij}\|$  where  $d_{ij}$  is the length of the current best known path from  $i$  to  $j$  and  $n_{ij}$  is the second node after  $i$  on this path. the algorithm tries to improve paths as follows.



If in figure 12  $d_{ik} + d_{kj} < d_{ij}$  then the known path from  $i$  to  $j$  should be replaced by the known path from  $i$  to  $k$  followed by the one from  $k$  to  $j$ .

### Floyd's Algorithm

(Initialization)

begin

for  $i := 1$  to  $n$  do

for  $j :=$  to  $n$  do

$(d(i,j) := \ell(i,j);$

$(n(i,j) := j)$

(main algorithm)

for k := 1 to n do

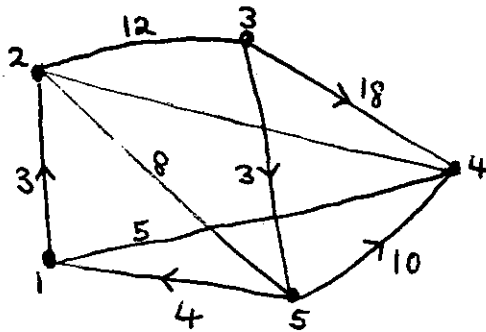
for i := 1 to n do

for j := 1 to n do

if  $d(i,j) > d(i,k) + d(k,j)$  then

$d(i,j) := d(i,k) + d(k,j)$ ;  $n(i,j) := n(i,k)$

end



Initially

D =	0	3	∞	5	∞	N =	1	2	3	4	5
	∞	0	12	∞	8		1	2	3	4	5
	∞	12	0	18	3		1	2	3	4	5
	5	4	∞	0	∞		1	2	3	4	5
	4	8	∞	10	0		1	2	3	4	5

Stage 1

D =	0	3	∞	5	∞	N =	1	2	3	4	5
	∞	0	12	∞	8		1	2	3	4	5
	∞	12	0	18	3		1	2	3	4	5
	5	4	∞	0	∞		1	2	3	4	5
	4	7	∞	9	0		1	1	3	1	0

Stage 2

D =	0	3	15	5	11	N =	1	2	2	4	2
	∞	0	12	∞	8		1	2	3	4	5
	∞	12	0	18	3		1	2	3	4	5
	5	4	16	0	12		1	2	2	4	2
	4	7	19	9	0		1	1	1	4	5



Stage 3

D =	0	3	15	5	11	N =	1	2	2	4	2
	$\infty$	0	12	30	8		1	2	3	3	5
	$\infty$	12	0	18	3		1	2	3	4	5
	5	4	16	0	12		1	2	2	4	2
	4	7	19	9	0		1	1	1	4	5

Stage 4

D =	0	3	15	5	11	N =	1	2	2	4	2
	35	0	12	30	8		3	2	3	3	5
	23	12	0	18	3		4	2	2	4	5
	5	4	16	0	12		1	2	2	4	2
	4	7	19	9	0		1	1	1	4	5

Stage 5

D =	0	3	15	5	11	N =	1	2	2	4	2
	12	0	12	17	8		5	2	3	5	5
	7	10	0	12	3		5	5	3	5	5
	5	4	16	0	12		1	2	2	4	2
	4	7	19	9	0		1	1	1	4	5

Ex

$$d_{32} = 10 \quad n_{32} = 5, \quad n_{52} = 1, \quad n_{12} = 2$$

Shortest path from 3 to 2 is (3,5,1,2)

Theorem 7.1

Floyds algorithm finds a shortest path between any pair of nodes.

Proof

We shall prove this by induction, the hypothesis being that at the beginning of the  $k$ th stage  $d_{ij}$  is the minimum length of a path from  $i$  to  $j$  with intermediate nodes taken from  $1, 2, \dots, k-1$ . This is true for  $k = 1$  and so assume it to be true for a general  $k$ . Now paths from  $i$  to  $j$  which only use  $1, 2, \dots, k$  as intermediate nodes either use node  $k$  or they do not. The minimum length of such paths which do not use  $k$  is the value of  $d_{ij}$  at the beginning of the  $k$ 'th stage. A path from  $i$  to  $j$  which uses  $k$

is the catenation of a path from  $i$  to  $k$  and a path from  $k$  to  $j$  and both these paths use only  $1, 2, \dots, k-1$  as intermediate nodes. Thus the minimum length of a path from  $i$  to  $j$  which only uses  $1, 2, \dots, k$  as intermediate nodes is  $\min \{d_{ij}, d_{ik} + d_{kj}\}$  which is the value of  $d_{ij}$  at the beginning of stage  $k+1$ . □

Department of Mathematics  
CARNEGIE MELLON UNIVERSITY

Operations Research II

Notation

$$x \oplus y = \min \{x, y\}, \quad \bigoplus_{i=1}^n x_i = \min \{x_1, x_2, \dots, x_n\}$$

$$x \otimes y = x + y$$

If  $A = \|a_{ij}\|$ ,  $B = \|b_{ij}\|$  are  $n \times n$  matrices then  $A \oplus B = \|c_{ij}\|$  where

$$c_{ij} = \bigoplus_{k=1}^n a_{ik} \otimes b_{kj} = \min \{a_{ik} + b_{kj} \mid 1 \leq k \leq n\}$$

e.g.

$$\begin{bmatrix} 4 & 2 \\ -1 & 3 \end{bmatrix} \oplus \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 1 & 0 \end{bmatrix}$$

Now let  $D$  be a digraph and  $L = \|\ell_{ij}\|$  where  $\ell_{ij}$  = length of arc  $(i, j)$ .

Claim

$$L^t = L \otimes L \otimes \dots \otimes L = \|\ell_{ij}^{(t)}\|$$

satisfies

$$\ell_{ij}^{(t)} = \min \text{ length of a walk from } i \text{ to } j \text{ using } t \text{ arcs or less.}$$

Proof

By induction on  $t$ . Obvious for  $t = 1$ . □

Let  $\lambda_{ij}^{(u)}$  = minimum length of a walk from  $i$  to  $j$  using  $u$  arcs or less. Now

$$(1) \quad \lambda_{ij}^{(u+1)} = \min \{ \lambda_{ik}^{(u)} + e_{kj} : k = 1, 2, \dots, n \}.$$

To see this note that for any  $u \geq 1$

$$(a) \quad \lambda_{ij}^{(u+1)} \leq \min \{ \lambda_{ik}^{(u)} + e_{kj} : k = 1, 2, \dots, n \}.$$

Since each term in the minimisation is the length of some walk from  $i$  to  $j$  using  $\leq u + 1$  arcs

and

(b) If  $W = (i = i_1, i_2, \dots, i_v = j)$ ,  $v \leq u + 1$  is a shortest walk using  $\leq u + 1$  arcs then

$$\begin{aligned} \lambda_{ij}^{(u+1)} &= e_{(i_1, i_2, \dots, i_{v-1})} + e_{i_{v-1}j} \geq \lambda_{ii_{v-1}}^{(u)} + e_{i_{v-1}j} \\ &\geq \min \{ \lambda_{ik}^{(u)} + e_{kj} : k = 1, 2, \dots, n \}. \end{aligned}$$

(a) and (b) imply (1).

We can now use induction on  $t$  to prove the claim.

Now

$$\begin{aligned} e_{ij}^{(t+1)} &= \bigoplus_{k=1}^n (e_{ik}^{(t)} \otimes j_{kj}) \\ &= \min \{ \lambda_{ik}^{(t)} + e_{kj} : k = 1, 2, \dots, n \} \\ &= \min \{ \lambda_{ik}^{(t)} + e_{kj} : k = 1, 2, \dots, n \} \text{ induction} \\ &= \lambda_{ij}^{(t+1)} \quad \text{by (1)}. \end{aligned}$$

□

Now if  $D$  has no negative circuits then  $\lambda_{ij}^{(n-1)}$  = minimum length of a path from  $i$  to  $j$  (why?) and so

$D^{n-1}$  is the matrix of shortest path lengths.

Question: if there are no negative cycles, why is

$$D^r = D^s \text{ for all } r, s, \geq n-1 ?$$

## Operations Research II

### Assignment Problems

Suppose that there are  $m$  jobs available and  $m$  people are to be considered for filling these jobs. Suppose it has been estimated that if person  $i$  is given job  $j$  then the cost of the job will be  $c_{ij}$ . The problem is how to assign people to jobs in such a way as to minimise the total cost.

We show how this problem can be solved as a sequence of shortest path problems.

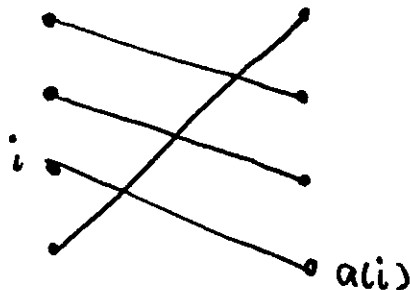
### Notation

An assignment  $a$  is a permutation of  $[m] = \{1, \dots, m\}$ .

A solution to the assignment problem can be expressed as person  $i$  does job  $a(i)$  for some assignment  $a$ . Let  $A_m = \{\text{assignments}\}$ . Then the problem can be expressed

$$\begin{aligned} & \text{minimise } \sum_{i=1}^m c_{ia(i)} \\ & \text{subject to } a \in A_m \end{aligned}$$

Another, fruitful, way of viewing this problem is that we seek a minimum weight perfect matching of the complete bipartite graph  $K_{m,m}$ .



$c_{ij}$  = 'weight' of edge  $(i,j)$

A k-assignment  $a$  is a permutation of  $[k]$ . For  $a \in A_k$  we let

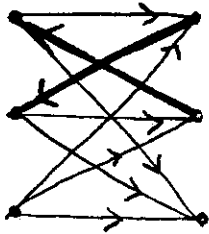
$$c(a) = \sum_{i=1}^k c_{ia(i)}$$

For  $a \in A_k$  define the bipartite digraph  $G(a)$  with nodes  $V_m = \{v_1, v_2, \dots, v_m\}$ ,  $W_m = \{w_1, w_2, \dots, w_m\}$  and arcs  $B(a) \cup F(a)$  where

$$B(a) = \{(w_{a(i)}, v_i) : 1 \leq i \leq k\} \quad \text{Backward Arcs}$$

$$F(a) = \{(v_i, w_j) : i > k \text{ or } i \leq k \text{ and } i \neq a(i)\} \quad \text{Forward Arcs}$$

Example  $m = 3, k = 2, a(1) = 2, a(2) = 1$



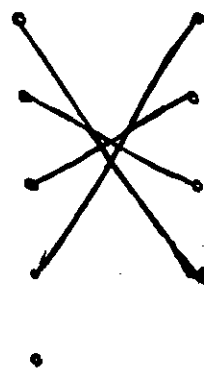
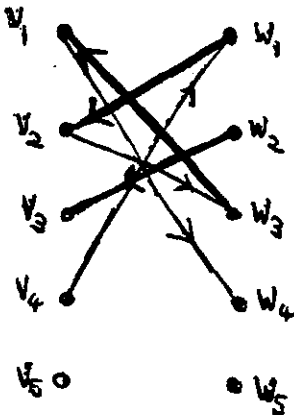
Arc lengths in  $G(a)$  are defined as follows:

$$\text{if } (v_i, w_j) \in F(a) \quad \ell(i, j) = c_{ij}$$

$$\text{if } (w_j, v_i) \in B(a) \quad \ell(j, i) = -c_{ij}$$

Suppose next that  $a \in A_k, k < m$  and  $P = (v_{k+1} = v_{i_1}, w_{j_1}, \dots, v_{i_p}, w_{j_p} = w_{k+1})$  is a path in  $G(a)$  from  $v_{k+1} \in V_m$  to  $w_{k+1} \in W_m$

- $a(1) = 3$
- $a(2) = 1$
- $a(3) = 2$



- $\hat{a} = a * P$
- add  $v_4 w_1$
- drop  $v_2 w_1$
- add  $v_2 w_3$
- drop  $v_1 w_3$
- add  $v_1 w_4$

$$P = v_4, w_1, v_2, w_3, v_1, w_4$$

We can construct a new assignment  $\hat{a}$  (denoted by  $a * P$ ) where

$$\begin{aligned}\hat{a}(i_t) &:= j_t & t = 1, \dots, p \\ \hat{a}(i) &= a(i) & \text{otherwise}\end{aligned}$$

It is not too difficult to see that  $a * P \in A_{k+1}$  and further that

$$\begin{aligned}c(a * P) &= c(a) + c_{i_1, j_1} - c_{i_2, j_1} + c_{i_2, j_2} - \dots + c_{i_p, j_p} \\ &= c(a) + \ell(P).\end{aligned}$$

We have a similar result if  $C = (v_{i_1}, w_{j_1}, \dots, v_{i_p}, w_{j_p}, v_{i_{p+1}} = v_{i_1})$  is a cycle in  $G(a)$  - assuming  $i_1 \leq k$ .

We again define  $a * C$  and this time  $c(a * C) = c(a) + \ell(C)$  and  $a * C \in A_k$

We need the following lemma: for a digraph  $G = (V, E)$  we define for each  $v \in V$

$$\begin{aligned}d^+(v) &= |\{(v, j) \in E\}| & - \text{outdegree of } v \\ d^-(v) &= |\{(i, v) \in E\}| & - \text{indegree of } v\end{aligned}$$

### Lemma 1

Suppose that for digraph  $G = (V, E)$  we have  $d^+(v), d^-(v) \leq 1$  for all  $v \in V$ . Then  $G$  is a collection of isolated nodes ( $d^+(v) = d^-(v) = 0$ ) plus a collection of node disjoint cycles  $C_1, \dots, C_p$  plus a collection of node disjoint paths  $P_1, \dots, P_q$  where

$$\begin{aligned}q &= |\{v : d^+(v) = 1, d^-(v) = 0\}| \\ &= |\{v : d^+(v) = 0, d^-(v) = 1\}| \end{aligned}$$



Proof (Outline)

By induction on  $|E|$ . Trivially true for  $|E| = 0$  and so assuming it is true for all digraphs with  $|E| < m$  and suppose we have a digraph with  $|E| = m > 0$ . Delete an arc  $(x,y)$ , apply the induction hypothesis and then put  $(x,y)$  back - which either turns a path into a cycle or lengthens a path or joins 2 isolated nodes. All cases maintain the result.  $\square$

A  $k$ -assignment  $a$  will be called  $\sigma$ -optimal if for any  $a' \in A_k$  with we have  $c(a) \leq c(a')$ .

Theorem

Let  $a \in A_k$  be  $\sigma$ -optimal

- (i) If  $C$  is a cycle in  $G(a)$  then  $\ell(C) \geq 0$
- (ii) Let  $P$  be a shortest path from  $v_{k+1}$  to  $w_{k+1}$ . Then  $a * P$  is also  $\sigma$ -optimal.

Proof

(i) Now  $a * C \in A_k$  and so  $c(a * C) = c(a) + \ell(C) \geq c(a)$  by assumption.

Hence  $\ell(C) \geq 0$ .

(ii) Let  $\hat{a}$  be any  $k+1$  - assignment. We show that there exist node disjoint cycles  $C_1, \dots, C_q$   $q \geq 0$  and a path  $Q$  from  $v_{k+1}$  to  $w_{k+1}$  such that

$$\hat{a} = a * C_1 * \dots * C_q * Q$$

(strictly speaking we need brackets on RHS of the above). Then

$$c(\hat{a}) = c(a) + \sum_{i=1}^q \ell(C_i) + \ell(Q)$$

$$\geq c(a) + \ell(Q) \quad \text{by (i)}$$

$$\geq c(a) + \ell(P) \quad \text{by assumption}$$

$$= c(a * P)$$

To construct  $C_1, \dots, C_q$  and  $Q$  consider the subset  $X$  of arcs of  $G(a)$  defined by

$$X = \{(v_i, w_{\hat{a}(i)}) : i = k+1 \text{ or } i \leq k \text{ and } a(i) \neq \hat{a}(i)\} \\ \cup \{(w_{a(i)}, v_i) : i \leq k \text{ and } a(i) \neq \hat{a}(i)\}$$

One then checks that the digraph  $(V_{k+1} \cup W_{k+1}, X)$  satisfies the conditions of the lemma, that  $d^+(v_{k+1}) = 1$ ,  $d^-(v_{k+1}) = 0$  and  $d^+(w_{k+1}) = 0$ ,  $d^-(w_{k+1}) = 1$  and that all other vertices are isolated or satisfy  $d^+(v) = d^-(v) = 1$ . The cycles and path of the lemma are what we need.  $\square$

Assignment Algorithm

(Initial Description)

```

begin
  a(1) = 1.
  for k = 2 to m do
    begin
      construct G(a);
      find a shortest path P from vk to wk;
      a = a*P
    end
  end
end

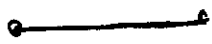
```

By Theorem 1 a remains  $\sigma$ -optimal and after  $m$ -iterations we will have solved the problem. As arc lengths could be negative, each shortest path problem could take  $O(m^3)$  steps and so the time complexity of the whole algorithm is  $O(m^4)$ . ~~We will subsequently show how to reduce this to  $O(m^3)$ .~~

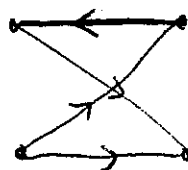
Example

$$n = 3 \begin{bmatrix} 6 & 4 & 5 \\ 4 & 2 & 4 \\ 7 & 2 & 5 \end{bmatrix} \leftarrow \|c_{ij}\|$$

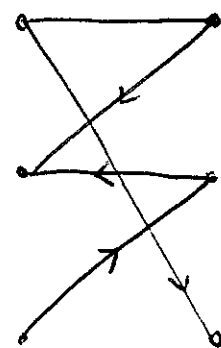
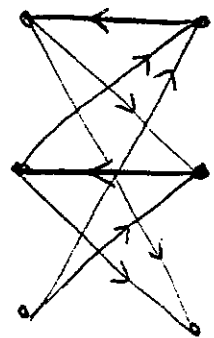
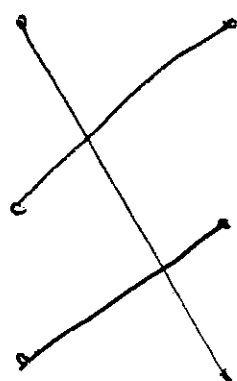
$\alpha$



$G(\alpha)$



$P$



Linear Programming Formulation

$$\text{ALP} = \text{minimise } \sum_{i=1}^m \sum_{j=1}^m c_{ij} x_{ij}$$

subject to

$$(1) \quad \sum_{i=1}^m x_{ij} = 1 \quad j = 1, 2, \dots, m$$

$$(2) \quad \sum_{j=1}^m x_{ij} = 1 \quad i = 1, 2, \dots, m$$

$$(3) \quad x_{ij} \geq 0 \quad \forall i, j.$$

Let  $X_F = \{\underline{x}: (1) - (3) \text{ hold}\}$

and

$$X_I = \{\underline{x} \in X_F: x_{ij} = 0 \text{ or } 1, \forall i, j\}.$$

Assignment Problem is equivalent to minimise  $c^T x$  subject to  $x \in X_I$ .

Proof

If  $\mathbf{a}$  is an assignment then putting

$$x_{i \mathbf{a}(i)} = 1 \quad i = 1, 2, \dots, m$$

$$x_{ij} = 0 \quad j \neq \mathbf{a}(i)$$

gives a solution to ALP with value  $c(\mathbf{a})$ .

Conversely, if  $\mathbf{x} \in X_I$  define  $\mathbf{a}$  by

$$\mathbf{a}(i) = \underline{\text{unique}} \ j \ \text{such that} \ x_{ij} = 1$$

(uniqueness follows from (1))

$\mathbf{a}$  is a permutation for if  $\mathbf{a}(r) = \mathbf{a}(s) = k$  then

$$\sum_{i=1}^m x_{ik} \geq x_{rk} + x_{sk} = 2, \quad \text{contradiction.} \quad \square$$

We show that

$$\begin{aligned} z_I &= \min \mathbf{c}^T \mathbf{x} \quad \text{subject to} \quad \mathbf{x} \in X_I = [\min\{c(\mathbf{a}) : \mathbf{a} \in A\}] \\ &= \min \mathbf{c}^T \mathbf{x} \quad \text{subject to} \quad \mathbf{x} \in X_F = z_F, \text{ say.} \end{aligned}$$

### Proof

Consider dual problem

$$\text{DALP} = \text{maximise} \quad \sum_{i=1}^m y_i + \sum_{j=1}^m z_j$$

$$\text{subject to} \quad y_i + z_j \leq c_{ij} \quad \forall i, j$$

We show that if  $\mathbf{a}^*$  is the assignment constructed by our algorithm then there exists  $(\mathbf{y}^*, \mathbf{z}^*)$  satisfying

$$(4) \quad y_i^* + z_j^* \leq c_{ij} \quad 1 \leq i, j \leq m$$

$$(5) \quad y_i^* + z_{a^*(i)}^* = c_{ia^*(i)} \quad 1 \leq i \leq m$$

Now (4) implies  $y^*, z^*$  is a dual solution and so

$$(6) \quad z_F \geq \sum_{i=1}^m y_i^* + \sum_{j=1}^m z_j^* \quad \begin{array}{l} [z_F \text{ is max. dual} \\ \text{[value}} \end{array}$$

But, from (5)

$$(7) \quad \begin{aligned} \sum_{i=1}^m c_{ia^*(i)} &= \sum_{i=1}^m y_i^* + \sum_{i=1}^m z_{a^*(i)}^* \\ &= \sum_{i=1}^m y_i^* + \sum_{j=1}^m z_j^* \end{aligned}$$

since  $a^*$  is a permutation.

Hence  $z_F \geq c(a^*) \geq z_F$ , where the first inequality follows from (6), (7) and the second from duality.

To construct  $y^*, z^*$  let  $\hat{a}$  be the  $(m-1)$ -assignment produced just before  $a^*$ . For  $x \in V \cup W$  let  $d(x) =$  length of shortest path from  $v_m$  to  $x$  in  $G(\hat{a})$ .

Let

$$y_i^* = -d(v_i) \quad i = 1, 2, \dots, m$$

$$z_j^* = d(w_j) \quad j = 1, 2, \dots, m.$$

Observe first that

$$d(v_i) = d(w_{\hat{a}(i)}) + \ell(w_{\hat{a}(i)}, v_i) \quad 1 \leq i \leq m-1$$

since  $(w_{\hat{a}(i)}, v_i)$  is the unique arc entering  $v_i$  in  $G(\hat{a})$ . Equivalently

$$(8) \quad -y_i^* = z_{\hat{a}(i)}^* - c_{i, \hat{a}(i)}.$$

This verifies (5) whenever  $\hat{a}(i) = a^*(i)$ , and also (4) for all  $i, j = \hat{a}(i)$ .

But if  $a^*(i) \neq \hat{a}(i)$  then the last path  $P$  found contains the arc

$(v_i, w_{a^*(i)})$  and for an arc of  $P$  we have

$$d(w_{a^*(i)}) = d(v_i) + \ell(v_i, w_{a^*(i)})$$

or

$$z_{a^*(i)}^* = -y_i + c_{i, a^*(i)}$$

and (5) is verified for all  $i$ .

We only now need check (4) for arcs of  $F(\hat{a})$  which are not on  $P$ . But for such arcs we have

$$d(w_j) \leq d(v_i) + \ell(v_i, w_j).$$

□