Foundations of Operations Research

Master of Science in Computer Engineering

Roberto Cordone
roberto.cordone@unimi.it

Tuesday 13.15 - 15.15
Thursday 10.15 - 13.15

http://homes.di.unimi.it/~cordone/courses/2014-for/2014-for.html

Lesson 6: Optimal paths | Como, Fall 2013

A shortest path model with negative costs

In a barter market, you have an apple and you want pears. Give your apple to someone in exchange for pears, or for something which can be exchanged for something else which... ending up with the maximum possible quantity of pears. Always exchange the whole amount of product (unnecessary, but simplifying assumption).

We model the market as a weighted directed graph $G = (N, A, r)$:

- a node for each item type owned by a merchant:
  s is your apple, t is a fictitious node reached by all pear nodes
- an arc for each possible exchange
- an exchange rate $r_{ij} : A \to \mathbb{R}^+$

A feasible solution is a chain of exchanges from your apple to any pears

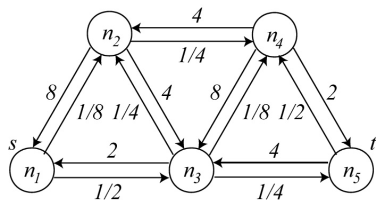
The gain in pears is the product of all exchange rates along the chain

$$f = \prod_{(i,j) \in X} r_{ij} \Leftrightarrow \max_X \sum_{(i,j) \in X} \log_2 r_{ij} \Leftrightarrow \min_X \sum_{e \in X} \log \frac{1}{r_{ij}}$$

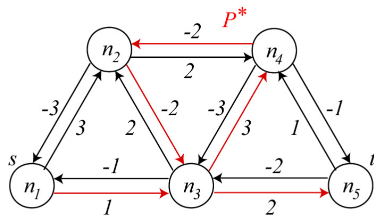
The costs $c_{ij} = \log \frac{1}{r_{ij}}$ are positive and negative

We are looking for a minimum cost path $P = (U, X)$ from s to t

A shortest path model with negative costs (2)



r_{ij}



$c_{ij} = \log_2 \frac{1}{r_{ij}}$

Since circuit (n_3, n_4, n_2) has a negative cost: $c_{342} = -1$

- $P_0 = (n_1, n_3, n_5)$ costs $c_0 = 3$: one apple for $1/8$ of pear
- $P_1 = (n_1, n_3, n_4, n_2, n_5)$ costs $c_1 = 2$: one apple for $1/4$ of pear
- $P_2 = (n_1, n_3, n_4, n_2, n_3, n_4, n_2, n_5)$ costs $c_3 = 1$: one apple for $1/2$ of pear
- ...

Economists call it a money pump: repeat the exchange circuit and get any amount of pears (*in practice, the market reacts adapting the prices*)

The minimum path problem with negative circuits has no optimum: it is an unbounded problem

A different model: shortest elementary path

A courier service must deliver mail to a part of the town, whose street network is known. Each delivery grants a known profit; visiting each street has a known cost, related to the postman's salary.

Compute the optimal delivery path from the office and back to it.

We model the town as a weighted directed graph $G = (N, A)$:

- a node for each crossing, plus node s for the office as a starting point and node t for the office as a destination point
- two opposite arcs for each street (simplifying assumptions: no road crossing during delivery, no partial service on a side of a street)
- a service net cost $c_{ij} : A \to \mathbb{R}$ (difference between salary and profit)

The profit is gained only at the first visit of the arc
Further visits are costly and useless: they should be avoided

We are looking for a minimum cost path $P = (U, X)$ from s to t
with a limit on the use of arcs (at most one visit)

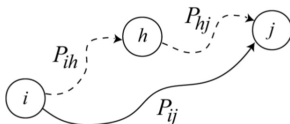
The problem is $\mathcal{NP}$-hard (no polynomial-complexity algorithm is known)

Triangular operation

We will describe an algorithm which

- detects whether the graph has circuits of negative cost
- if it has none, provides a minimum cost path between each pair of nodes

Triangular operation: given a path P_{ij} from i to j and a node $h \in N \setminus \{i, j\}$, replace it with a path P_{ih} from i to h plus a path P_{hj} from h to j



When $c_{P_{ij}} > c_{P_{ih}} + c_{P_{hj}}$, the resulting path is better than the original one

- How should we apply these operations, and when should we stop?
- Is the final solution certainly optimal?

A fundamental theorem

Let $N_h = \{n_i \in N : i \leq h\} = \{n_1, \dots, n_h\}$ be the subset of the first h nodes

Let $P^{*\,(h)}_{ij}$ be the minimum cost path from i to j in the subgraph induced by $\{n_i, n_j\} \cup N_h$ and $\ell^{(h)}_{ij}$ be the corresponding cost, for $h = 0, \dots, n-1$

Then

$$\ell^{(h+1)}_{ij} = \min \left(\ell^{(h)}_{ij}, \ell^{(h)}_{i,h+1} + \ell^{(h)}_{h+1,j} \right)$$

Proof: *Path* $P^{*\,(h+1)}_{ij}$ *either includes* n_{h+1} *or not*

If $n_{h+1} \notin P^{*\,(h+1)}_{ij}$*, then* $P^{*\,(h+1)}_{ij}$ *is also the minimum cost path from* i *to* j *in the subgraph induced by* $\{i, j\} \cup N_h$*, and its cost is* $\ell^{(h)}_{i,j}$

If $n_{h+1} \in P^{*\,(h+1)}_{ij}$*, then* $P^{*\,(h+1)}_{ij}$ *goes from* n_i *to* n_{h+1} *and from* n_{h+1} *to* n_j*.*
By the optimality principle

- *the former subpath is optimal in the graph induced by* $\{n_i, n_j\} \cup N_{h+1}$*; since it does not use* j*, it is also optimal in the graph induced by* $\{n_i\} \cup N_{h+1} = \{n_i, n_{h+1}\} \cup N_h$ *and its cost is* $\ell^{(h)}_{i,h+1}$
- *the latter subpath is optimal in the graph induced by* $\{n_i, n_j\} \cup N_{h+1}$*; since it does not use* i*, it is also optimal in the graph induced by* $\{n_j\} \cup N_{h+1} = \{n_{h+1}, n_j\} \cup N_h$ *and its cost is* $\ell^{(h)}_{h+1,j}$

Floyd-Warshall's algorithm (1962)

Represent all shortest paths through two matrices

- ℓ_{ij} is the cost of a minimum path from i to j
- π_{ij} is the predecessor node along the minimum path from i to j

Floyd-Warshall(N, A, c)

$\{\ N_0 = \emptyset$: the path in $\{i, j\} \cup N_0$ is arc $(i, j)\ \}$

For each $i \in N$ **do**

  For each $j \in N$ **do**

    $\ell_{ij} := c_{ij};\ \pi_j := i;$

For $h := 1$ **to** n **do**

  For each $i \in N \setminus \{h\}$ **do**

    For each $j \in N \setminus \{h\}$ **do**

      If $\ell_{ij} > \ell_{ih} + \ell_{hj}$ **then** { triangular operation }

        $\ell_{ij} := \ell_{ih} + \ell_{hj};\ \pi_{ij} := \pi_{hj};$

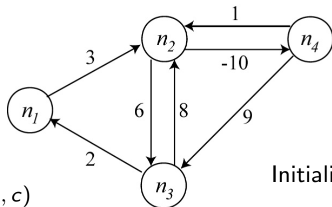
If $\exists i \in N : \ell_{ii} < 0$

  then Return "Negative circuit detected";

  else Return (ℓ, π);

The overall complexity is $O\left(n^3\right)$

Application of Floyd-Warshall's algorithm (1)



Floyd-Warshall(N, A, c)

For each $i \in N$ **do**

  For each $j \in N$ **do**

    $\ell_{ij} := c_{ij}$; $\pi_j := i$;

For $h := 1$ **to** n **do**

  For each $i \in N \setminus \{h\}$ **do**

    For each $j \in N \setminus \{h\}$ **do**

      If $\ell_{ij} > \ell_{ih} + \ell_{hj}$ **then**

        $\ell_{ij} := \ell_{ih} + \ell_{hj}$; $\pi_{ij} := \pi_{hj}$;

If $\exists i \in N : \ell_{ii} < 0$

  then Return Negative circuit;
  else Return (ℓ, π);

Initialization $(h = 0)$

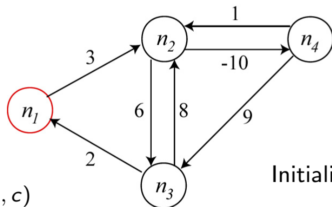
Distance matrix

ℓ	1	2	3	4
1	0	3	$+\infty$	$+\infty$
2	$+\infty$	0	6	-10
3	2	8	0	$+\infty$
4	$+\infty$	1	9	0

Predecessor matrix

π	1	2	3	4
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

Application of Floyd-Warshall's algorithm (2)



Floyd-Warshall(N, A, c)

For each $i \in N$ **do**

  For each $j \in N$ **do**

    $\ell_{ij} := c_{ij}$; $\pi_j := i$;

For $h := 1$ **to** n **do**

  For each $i \in N \setminus \{h\}$ **do**

    For each $j \in N \setminus \{h\}$ **do**

      If $\ell_{ij} > \ell_{ih} + \ell_{hj}$ **then**

        $\ell_{ij} := \ell_{ih} + \ell_{hj}$; $\pi_{ij} := \pi_{hj}$;

If $\exists i \in N : \ell_{ii} < 0$

  then Return Negative circuit;
  else Return (ℓ, π);

Initialization $(h = 1)$

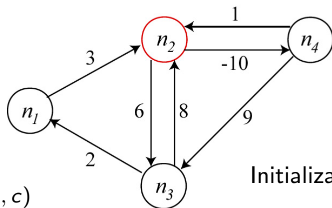
Distance matrix

ℓ	1	2	3	4
1	0	3	$+\infty$	$+\infty$
2	$+\infty$	0	6	-10
3	2	5	0	$+\infty$
4	$+\infty$	1	9	0

Predecessor matrix

π	1	2	3	4
1	1	1	1	1
2	2	2	2	2
3	3	1	3	3
4	4	4	4	4

Application of Floyd-Warshall's algorithm (3)



Floyd-Warshall(N, A, c)

For each $i \in N$ **do**

  For each $j \in N$ **do**

    $\ell_{ij} := c_{ij};\ \pi_j := i;$

For $h := 1$ **to** n **do**

  For each $i \in N \setminus \{h\}$ **do**

    For each $j \in N \setminus \{h\}$ **do**

      If $\ell_{ij} > \ell_{ih} + \ell_{hj}$ **then**

        $\ell_{ij} := \ell_{ih} + \ell_{hj};\ \pi_{ij} := \pi_{hj};$

If $\exists i \in N : \ell_{ii} < 0$

  then Return Negative circuit;
  else Return (ℓ, π);

Initialization ($h = 2$)

Distance matrix

ℓ	1	2	3	4
1	0	3	9	-7
2	$+\infty$	0	6	-10
3	2	5	0	-2
4	$+\infty$	1	7	-9

Predecessor matrix

π	1	2	3	4
1	1	1	2	2
2	2	2	2	2
3	3	1	3	2
4	4	4	2	2

There is a negative circuit: since the ℓ_{ij} always decrease, we can terminate

Dynamic programming

Dijkstra's and Floyd-Warshall's algorithms are examples of the dynamic programming approach

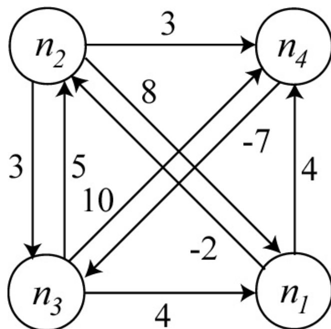
- decompose the problem into subsequent stages named restrictions
- prove that the optimal solution of stage $h+1$ depends only on the optimal solutions of stages $1, \dots, h$ (i. e. the optimality principle)
- build the optimal solution starting from those of the subproblems

This general approach was proposed by Richard Bellman in 1940

Of course, it works only if the problem enjoys the optimality principle

Exercise

Compute the minimum cost path between each pair of nodes



Optimal paths in directed acyclic graphs

In a graph with negative cost circuits, the minimum path problem is

- either unbounded
- or ill-posed, and requires additional constraints which make it hard

There are two important special cases in which negative cost circuits are certainly nonexistent:

1. when the cost function is nonnegative: $c_{ij} \geq 0$ for all $(i,j) \in A$

   Then, we apply Dijkstra's algorithm (*saving time and memory space*)

2. when the graph has no circuits: $\nexists P = \{(i_0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k)\}$ such that $(i_h, i_{h+1}) \in A$ for $h = 0, \ldots, k-1$ and $i_k = i_0$

   Then, we apply another dynamic programming algorithm (*saving time and memory space*)

   This algorithm can be adapted to compute maximum cost paths

Is this relevant in practice?

Of course, it is: do you remember the replacement plan problem?

A fundamental property (topological ordering)

The nodes of a directed acyclic graph $G = (N, A)$ can be topologically ordered, i. e. indexed so that

$$(n_i, n_j) \in A \Leftrightarrow i < j$$

The topological order can be exploited to decompose the problem and solve it very efficiently by dynamic programming

In a topologically ordered directed acyclic graph $G = (N, A)$,
for each arc the index of the head is larger than the index of the tail.

Then, in such a graph all paths visit nodes with increasing indices

An algorithm for topological ordering

1. Set $i := 1$
2. Select a node with $\Delta_v^- = \emptyset$
   Denote it as n_i
3. Remove n_i and all incident arcs from the graph
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

TopologicalOrdering(N, A)

For $i := 1$ **to** n **do**

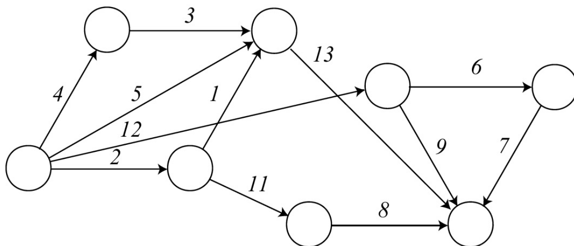
$Q := \{n \in N : \Delta_n^- = \emptyset\}$;
$n :=$ Extract(Q);
index$[n] := i$;
$A := A \setminus \Delta_n^+$;
$N := N \setminus \{n\}$;

Notice that there is always at least a node with no ingoing arcs:
by contradiction, if all nodes have ingoing arcs, the graph has circuits
(*follow the arcs: either you get back to a visited node or you proceed forever*)

The overall complexity is $O(m)$ because

- the set Q of nodes with zero indegree can be computed in $O(m)$ time and updated in constant time for each arc
- extracting a node and setting its index takes constant time for each node
- removing the arcs takes constant time for each arc

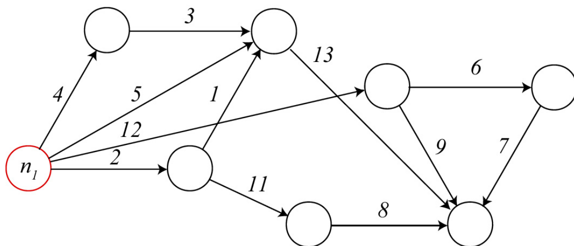
Application of the topological ordering algorithm (1)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_v^- = \emptyset$ and denote it as n_i
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

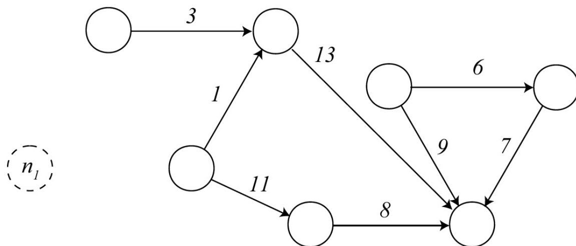
Application of the topological ordering algorithm (2)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i ($i = 1$)
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

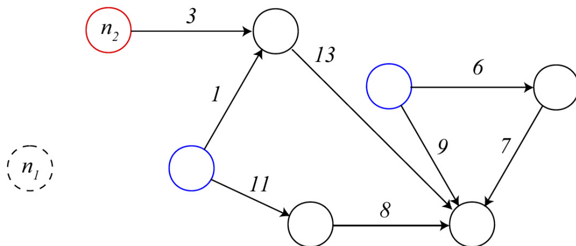
Application of the topological ordering algorithm (3)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i $(i = 1)$
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

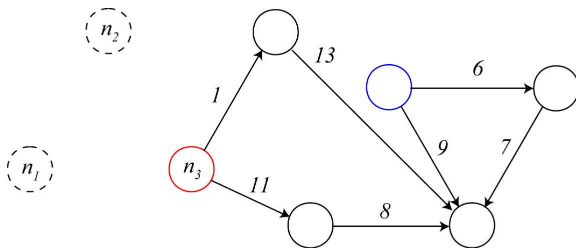
Application of the topological ordering algorithm (4)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i ($i = 2$)
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

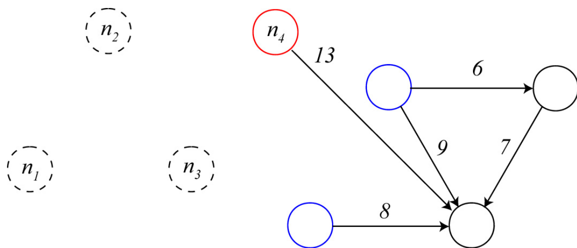
Application of the topological ordering algorithm (5)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i ($i = 3$)
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

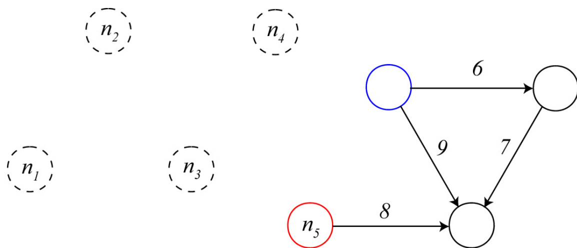
Application of the topological ordering algorithm (6)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i $(i = 4)$
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

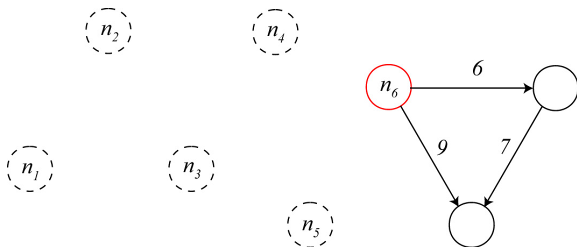
Application of the topological ordering algorithm (7)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i ($i = 5$)
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

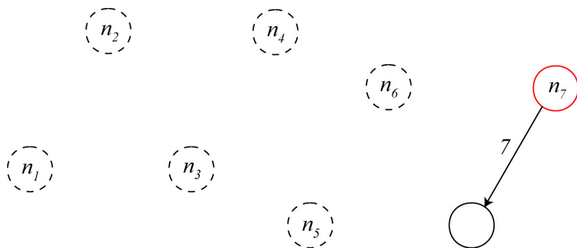
Application of the topological ordering algorithm (8)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i ($i = 6$)
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

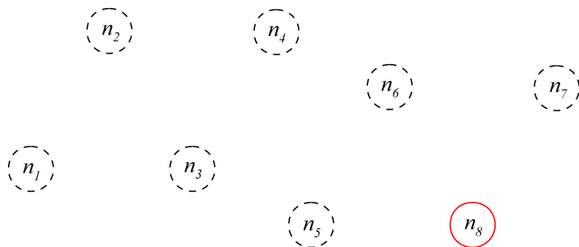
Application of the topological ordering algorithm (9)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i ($i = 7$)
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

Application of the topological ordering algorithm (10)



Algorithm for topological ordering:

1. $i := 1$
2. Select a node with $\Delta_n^- = \emptyset$ and denote it as n_i ($i = 8$)
3. Remove n_i and all incident arcs
4. If there are still nodes, then set $i := i + 1$ and go to step 2, else terminate

A fundamental theorem

Let ℓ_j be the minimum cost of a path from s to n_j in a topologically ordered acyclic directed graph

Then

$$\ell_j = \min_{(i,j)\in\Delta_j^-} (\ell_i + c_{ij})$$

Proof: *Nodes preceding s in the topological order cannot be reached from s. Therefore, let $s = n_1$.*

By induction:

- *for $h = 1$, the optimal path from $s = n_1$ to n_h is trivial (no arc) and its cost is $\ell_1 = 0$*
- *for $h > 1$, any path from s to n_h goes from n_1 to one of the nodes n_i $(i = 1, \ldots, h-1)$ and ends with arc (n_i, n_h)*

  Given the node $i^* \in \{1, \ldots, h-1\}$ which minimizes $\ell_i + c_{ih}$, the optimal path from n_1 to n_{i^*} plus arc (n_{i^*}, n_h) costs no more than any other path from n_1 to n_h: therefore it is optimal

Shortest path algorithm in topologically ordered graphs

DynamicProgramming(N, A, c, s)

$\ell_1 := 0$; $\pi_1 := n_1$;

For $h := 2$ **to** n **do**

$i^* := \arg \min_{i \in N : (i,h) \in \Delta_h^-} (\ell_i + c_{ih})$;

$\ell_h := \ell_{i^*} + c_{i^* h}$;

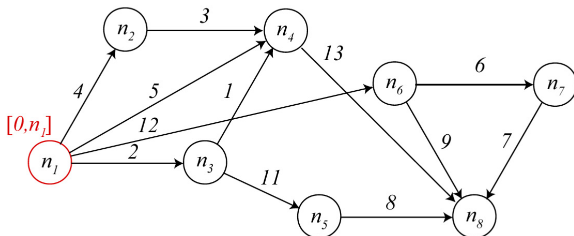
$\pi_h := i^*$;

Return (ℓ, π);

The overall complexity is $O(m)$ because

- the selection of i^* takes constant time for each arc
- the computation of ℓ and π takes constant time for each node

Application of the algorithm (1)



DynamicProgramming(N, A, c, s)

$\ell_1 := 0$; $\pi_1 := n_1$;

$\ell_1 := 0 \ \pi_1 := n_1$

For $h := 2$ **to** n **do**

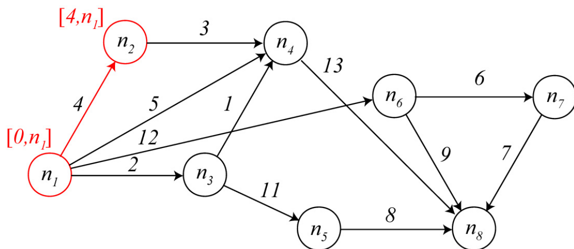
$$i^* := \arg \min_{i \in N : (i,h) \in \Delta_h^-} (\ell_i + c_{ih});$$

$\ell_h := \ell_{i^*} + c_{i^*h}$;

$\pi_h := i^*$;

Return (ℓ, π);

Application of the algorithm (2)



DynamicProgramming(N, A, c, s)

$\ell_1 := 0;\ \pi_1 := n_1;$

For $h := 2$ **to** n **do**

$i^* := \arg\min_{i \in N : (i,h) \in \Delta_h^-} (\ell_i + c_{ih});$

$\ell_h := \ell_{i^*} + c_{i^* h};$

$\pi_h := i^*;$

Return $(\ell, \pi);$

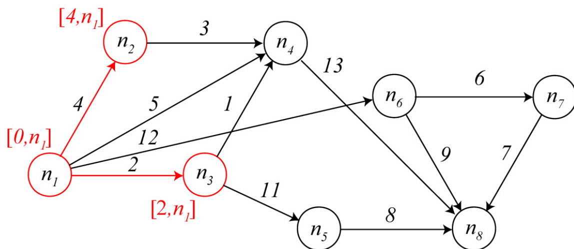
$h = 2$

$\Delta_h^- = \{(n_1, n_2)\} \Rightarrow i^* = 1$

$\ell_2 = \ell_1 + c_{12} = 4$

$\pi_2 = 1$

Application of the algorithm (3)



DynamicProgramming(N, A, c, s)

$\ell_1 := 0$; $\pi_1 := n_1$;

For $h := 2$ **to** n **do**

$i^* := \arg \min_{i \in N:(i,h) \in \Delta_h^-} (\ell_i + c_{ih})$;

$\ell_h := \ell_{i^*} + c_{i^* h}$;

$\pi_h := i^*$;

Return (ℓ, π);

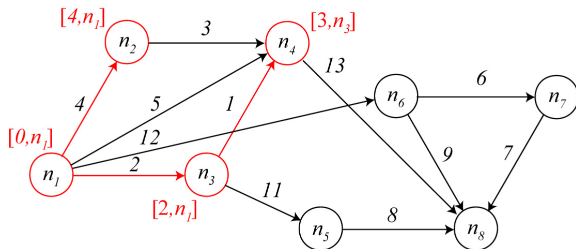
$h = 3$

$\Delta_h^- = \{(n_1, n_3)\} \Rightarrow i^* = 1$

$\ell_3 = \ell_1 + c_{13} = 2$

$\pi_3 = 1$

Application of the algorithm (4)



DynamicProgramming(N, A, c, s)

$\ell_1 := 0;\ \pi_1 := n_1;$

For $h := 2$ **to** n **do**

$i^* := \arg \min_{i \in N : (i,h) \in \Delta_h^-} (\ell_i + c_{ih});$

$\ell_h := \ell_{i^*} + c_{i^* h};$

$\pi_h := i^*;$

Return $(\ell, \pi);$

$h = 4$

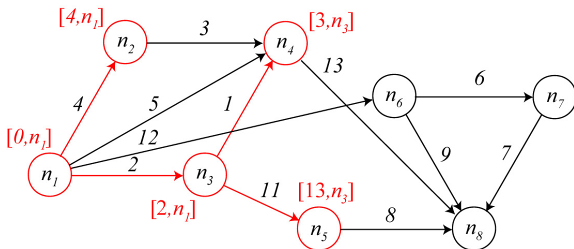
$\Delta_h^- = \{(n_1, n_4), (n_2, n_4), (n_3, n_4)\} \Rightarrow$

$i^* = 3$

$\ell_4 = \ell_3 + c_{34} = 3$

$\pi_4 = 3$

Application of the algorithm (5)



DynamicProgramming(N, A, c, s)

$\ell_1 := 0$; $\pi_1 := n_1$;

For $h := 2$ **to** n **do**

$i^* := \arg \min_{i \in N : (i,h) \in \Delta_h^-} (\ell_i + c_{ih})$;

$\ell_h := \ell_{i^*} + c_{i^* h}$;

$\pi_h := i^*$;

Return (ℓ, π);

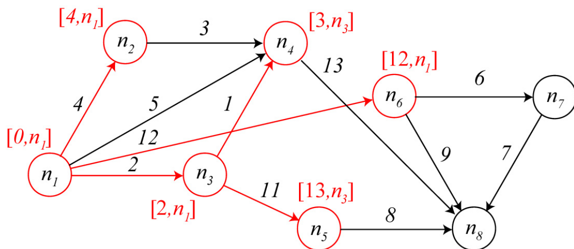
$h = 5$

$\Delta_h^- = \{(n_3, n_5)\} \Rightarrow i^* = 3$

$\ell_5 = \ell_3 + c_{35} = 13$

$\pi_5 = 3$

Application of the algorithm (6)



DynamicProgramming(N, A, c, s)

$\ell_1 := 0;\ \pi_1 := n_1;$

For $h := 2$ **to** n **do**

$\quad i^* := \arg \min_{i \in N : (i,h) \in \Delta_h^-} (\ell_i + c_{ih});$

$\quad \ell_h := \ell_{i^*} + c_{i^* h};$

$\quad \pi_h := i^*;$

Return $(\ell, \pi);$

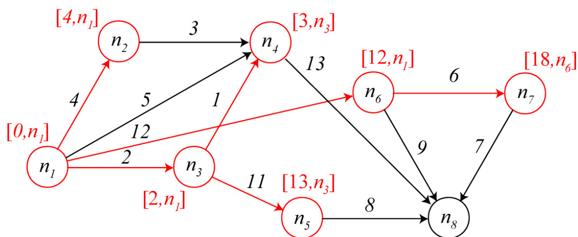
$h = 6$

$\Delta_h^- = \{(n_1, n_6)\} \Rightarrow i^* = 1$

$\ell_6 = \ell_1 + c_{16} = 12$

$\pi_6 = 1$

Application of the algorithm (7)



DynamicProgramming(N, A, c, s)

$\ell_1 := 0;\ \pi_1 := n_1;$

For $h := 2$ **to** n **do**

$i^* := \arg \min_{i \in N : (i,h) \in \Delta_h^-} (\ell_i + c_{ih});$

$\ell_h := \ell_{i^*} + c_{i^* h};$

$\pi_h := i^*;$

Return $(\ell, \pi);$

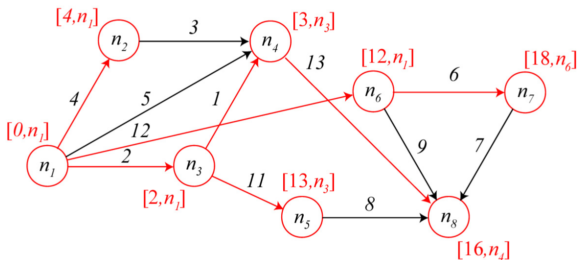
$h = 7$

$\Delta_h^- = \{(n_6, n_7)\} \Rightarrow i^* = 6$

$\ell_7 = \ell_6 + c_{67} = 18$

$\pi_7 = 3$

Application of the algorithm (8)



DynamicProgramming(N, A, c, s)

$\ell_1 := 0$; $\pi_1 := n_1$;

For $h := 2$ **to** n **do**

$i^* := \arg \min_{i \in N:(i,h) \in \Delta_h^-} (\ell_i + c_{ih})$;

$\ell_h := \ell_{i^*} + c_{i^* h}$;

$\pi_h := i^*$;

Return (ℓ, π);

$h = 8$

$\Delta_h^- = \{(n_4, n_8), (n_5, n_8), (n_6, n_8), (n_7, n_8)\}$

$\Rightarrow i^* = 4$

$\ell_8 = \ell_4 + c_{48} = 16$

$\pi_8 = 4$

Maximum cost path in directed acyclic graphs

Computing the maximum of an objective function is equivalent to computing the minimum of its opposite

$$\max f(x) \Leftrightarrow \min [-f(x)]$$

An algorithm which makes no assumption on the sign of the objective function can be used indifferently to minimize or maximize it

When computing the maximum path

- The problem is unbounded if the graph includes positive cost circuits
- If constraints are imposed on the number of visits to each arc or node, the problem is $\mathcal{NP}$-hard
- Floyd-Warshall's algorithm detects positive cost circuits; if none exists, it computes the maximum path between each pair of nodes
- Dijkstra's algorithm cannot be applied
- If the graph has no circuits, the dynamic programming algorithm computes the maximum path from each node to all the other ones