Shortest path problem

Properties and algorithms

Michel Bierlaire

Introduction to optimization and operations research

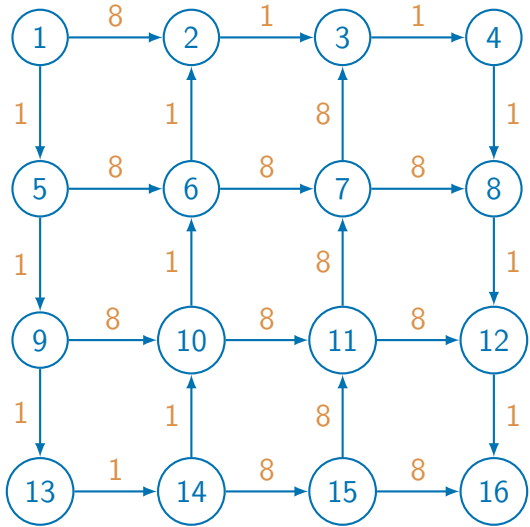
EPFL

Motivation

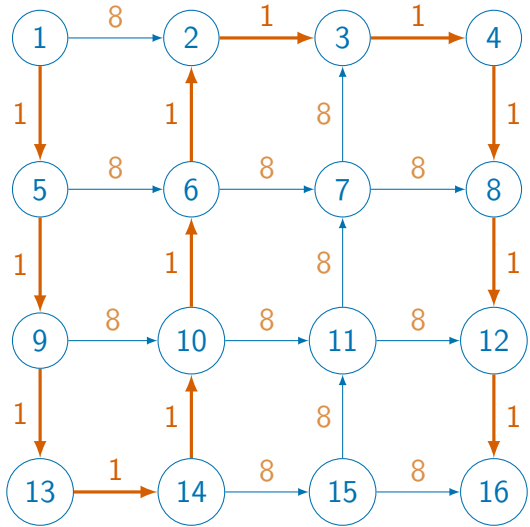
Shortest path problem

- The shortest path problem is a transhipment problem.
- It could therefore be solved with the simplex algorithm.
- However, it does not exploit the specific structure of the problem.
- We identify here some useful properties of the shortest path problem, that will be exploited by a dedicated algorithm.

Example: what is the shortest path from 1 to 16?



Example

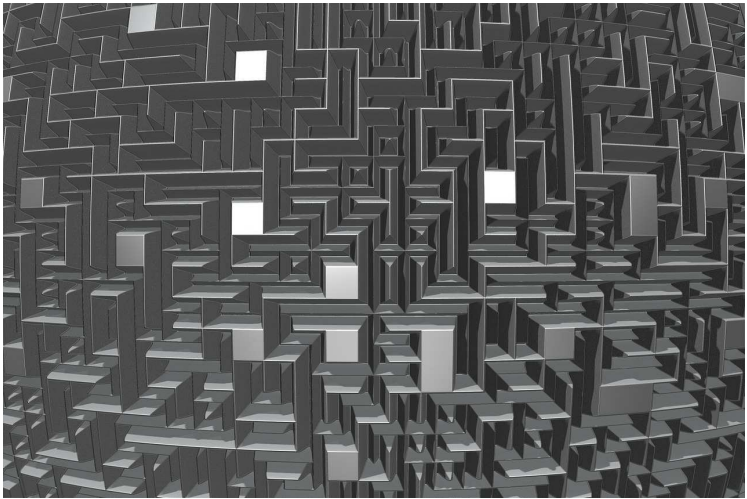


Comments

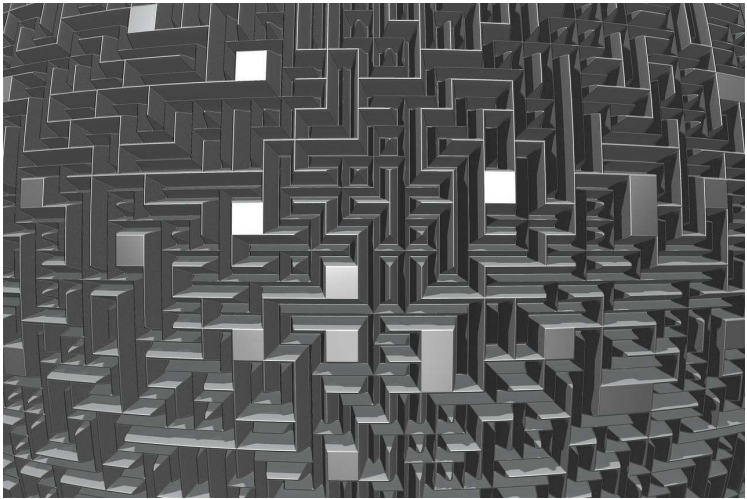
Map-based intuition

- When we look at a map, we have some intuition about what the shortest path could be.
- But, in general networks, the triangle inequality is not always valid, like in the example.
- Moreover, the computer has no bird eye's view on the network.

Main idea of the method



Main idea of the method



Complementarity slackness for the transhipment problem

For all (i, j)

$$c_{ij} + \lambda_i - \lambda_j \geq 0.$$

For all (i, j) such that $x_{ij} > 0$

$$c_{ij} + \lambda_i - \lambda_j = 0.$$

Sufficient and necessary optimality conditions.

Optimality conditions

Theorem 23.1

Consider

- a network $(\mathcal{N}, \mathcal{A})$,$n$ arcs, m nodes,
- a cost vector $c \in \mathbb{R}^n$,
- a vector of labels $\lambda \in \mathbb{R}^m$,
- a path P between node o and node d.

$$\lambda_j \leq \lambda_i + c_{ij}, \quad \forall (i,j) \in \mathcal{A}.$$

$$\lambda_j = \lambda_i + c_{ij}, \quad \forall (i,j) \in P.$$

then P is a shortest path from o to d.

Proof

$$\lambda_j \leq \lambda_i + c_{ij}, \quad \forall (i,j) \in \mathcal{A}.$$
$$\lambda_j = \lambda_i + c_{ij}, \quad \forall (i,j) \in P.$$

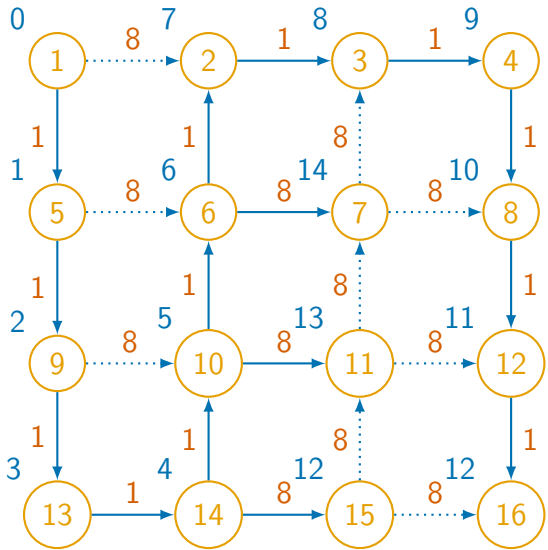
Consider any path Q

$$Q = o \to j_1 \to j_2 \ldots j_\ell \to d.$$
$$C(Q) = c_{oj_1} + c_{j_1 j_2} + \ldots + c_{j_\ell d}.$$
$$C(Q) \geq (\lambda_{j_1} - \lambda_o) + (\lambda_{j_2} - \lambda_{j_1}) + \ldots + \lambda_d - \lambda_{j_\ell} = \lambda_d - \lambda_o,$$
$$P = o \to i_1 \to i_2 \ldots i_k \to d.$$
$$C(P) = c_{oi_1} + c_{i_1 i_2} + \ldots + c_{i_k d}.$$
$$C(P) = (\lambda_{i_1} - \lambda_o) + (\lambda_{i_2} - \lambda_{i_1}) + \ldots + \lambda_d - \lambda_{i_k} = \lambda_d - \lambda_o.$$

Example

Show that the condition is verified with equality on the plain arcs
and with strict inequality on the dotted arcs
Show on an example that it corresponds to shortest paths
Mention that the subnetwork of plain arcs for a spanning tree.
Remind that, in a tree, there is exactly one path between two nodes.

Example: the shortest path tree



- Arcs in the tree: $\lambda_2 + c_{23} = 7 + 1 = 8 = \lambda_3$.
- Arcs not in the tree: $\lambda_5 + c_{56} = 1 + 8 = 9 > \lambda_6 = 6$.
- The plain arcs form a spanning tree.
- Therefore, there is exactly one path between two nodes: the shortest.

Labels

Optimality conditions

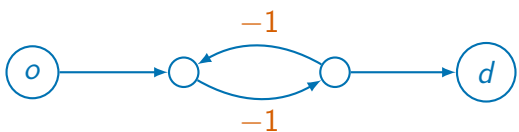
$$\lambda_j - \lambda_i \leq c_{ij}, \quad \forall (i,j) \in \mathcal{A}.$$
$$\lambda_j - \lambda_i = c_{ij}, \quad \forall (i,j) \in P.$$

Only differences matter

Common practice: normalize $\lambda_o = 0$.

Negative cost cycle

If there exists a forward path from o to d containing a negative cost cycle, no forward path is the shortest path from o to d.



Simple paths

- Consider a network $(\mathcal{N}, \mathcal{A})$, and
- two nodes o and d.
- Consider a forward path P between o and d
- that does not contain a negative cost cycle.
- Then there exists a simple forward path Q from o to d such that $C(Q) \leq C(P)$.
- Example: $o \to 1 \to 2 \to 1 \to 2 \to d$.
- It contains a cycle, with a non negative cost.
- Removing the cycle cannot increase the length of the path.

Conclusion: if there is a shortest path, there is a shortest path that is simple.

Lower bound on the length of shortest paths

Corollary

- If $c \geq 0$, then $C(P) \geq 0$.
- Otherwise,

$$C(P) \geq (m-1) \min_{(i,j) \in \mathcal{A}} c_{ij}.$$

Principle of optimality

- Consider a network $(\mathcal{N}, \mathcal{A})$,
- and two nodes o and d.
- Let $P = o \to i_1 \to i_2 \dots i_k \to d$ be a shortest path from o to d.
- Then, for any $\ell = 1, \dots, k$, the subpath $P_{o\ell} = o \to \dots \to i_\ell$ is a shortest path from o to i_ℓ
- and $P_{\ell d} = i_\ell \to \dots \to d$ is a shortest path from i_ℓ to d.

Principle of optimality

The shortest path algorithm

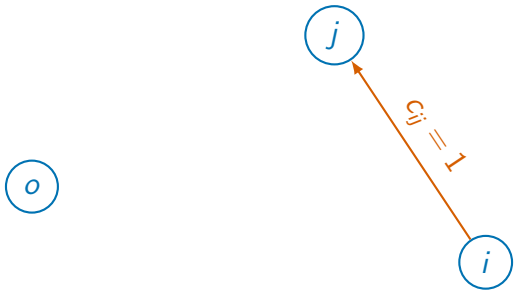
Motivation

- We consider here the single origin–all destinations shortest path problem.
- The optimality conditions are based on labels associated with each node.
- Idea of the algorithm: make sure that the labels verify the optimality conditions.
- The label of node j can be interpreted as the length of the shortest path identified so far from the origin to node j.

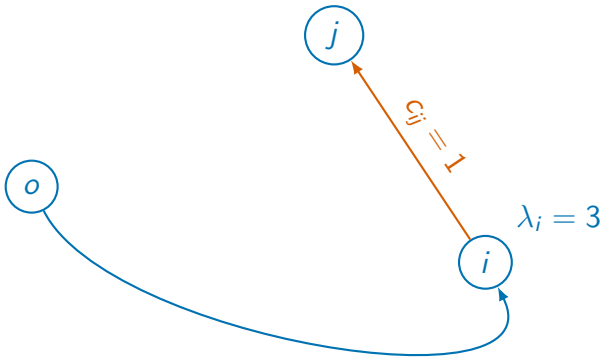
Optimality conditions

$$\lambda_j \leq \lambda_i + c_{ij}, \quad \forall (i,j) \in \mathcal{A}.$$
$$\lambda_j = \lambda_i + c_{ij}, \quad \forall (i,j) \in P.$$

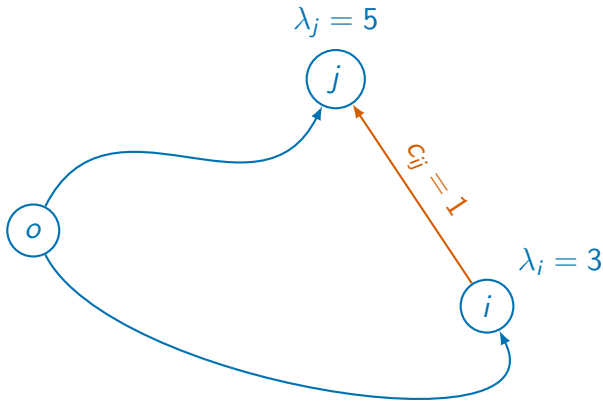
Labels



Labels

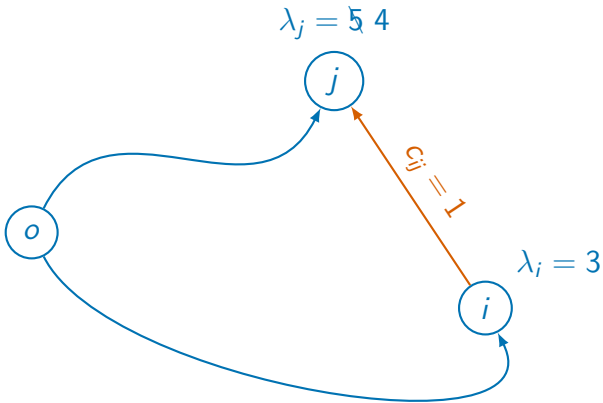


Labels



$$\lambda_j > \lambda_i + c_{ij}.$$

Labels



Idea of the algorithm

For each arc (i, j) in the network, if

$$\lambda_j > \lambda_i + c_{ij},$$

$$\lambda_j \leftarrow \lambda_i + c_{ij}.$$

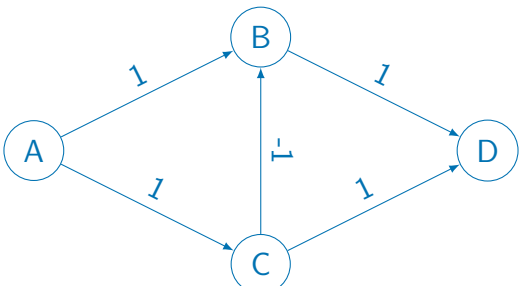
Second idea

Loop on the m nodes instead of the n arcs.

$\mathcal{S} \leftarrow \{o\}$, $\lambda_o = 0$, $\lambda_j = +\infty$.

- Select i in $\mathcal{S}$.
- For each (i, j):
  - If $\lambda_j > \lambda_i + c_{ij}$,
    - $\lambda_j \leftarrow \lambda_i + c_{ij}$.
    - If $\lambda_j < 0$ and $\lambda_j < (m-1) \min_{(i,j) \in \mathcal{A}} c_{ij}$: STOP.
    - $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.
- $\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$.
- If $\mathcal{S} = \emptyset$: STOP.
- Otherwise, start again.

Example



$\mathcal{S}$	i	λ_A	λ_B	λ_C	λ_D
$\{A\}$	A	0	∞	∞	∞
$\{B, C\}$	B	0	$1(A)$	$1(A)$	∞
$\{C, D\}$	C	0	$1(A)$	$1(A)$	$2(B)$
$\{B, D\}$	B	0	$0(C)$	$1(A)$	$2(B)$
$\{D\}$	D	0	$0(C)$	$1(A)$	$1(B)$
$\emptyset$	–	0	$0(C)$	$1(A)$	$1(B)$

Ingredients

Arcs

c_{ij}

Nodes

λ_i

Set of nodes

$\mathcal{S}$

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \; \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

At the end of each iteration

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \ \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If $i \in \mathcal{S}$, then $\lambda_i \neq \infty$.

- True at the very beginning: Initialization: node o.
- When treating node i, $\lambda_i \neq +\infty$.
- Any other node is inserted after the update of the label.

At the end of each iteration

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \ \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

For each node i, the value of λ_i does not increase during the iteration.

Direct consequence of the update condition

At the end of each iteration

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \ \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If $\lambda_i \neq \infty$, it is the length of one path from o to i.

- Initialization: $\lambda_o = 0$ can be interpreted that way.
- Iteration 1: labels are the length of the arc = path.
- By induction: suppose true before treating node k.
- As $k \in \mathcal{S}$, $\lambda_k \neq \infty$.
- Therefore, it is the length of a path (induction assumption).
- The iteration adds one arc, if necessary.

At the end of each iteration

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \ \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If $i \notin \mathcal{S}$, then

- $\lambda_i = \infty$, or
- $\lambda_j \leq \lambda_i + c_{ij}$, for all j such that $(i,j) \in \mathcal{A}$.

- Two reasons not to be in $\mathcal{S}$.
- 1. i has never been treated: $+\infty$.
- 2. i has been treated and removed from $\mathcal{S}$.
- Just after i has been treated, the inequality is verified.
- While i is out of $\mathcal{S}$, the other labels can only decrease, so that the inequality is verified.

Termination

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \ \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

Finite number of iterations.

- Termination: $\mathcal{S} = \emptyset$.
- If not, some nodes are added an infinite number of times.
- Each time, their label has a strictly lower value.
- $\lambda_i \to -\infty$.
- Impossible thanks to the other stopping criterion.

Termination

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \; \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If algorithm terminates with $\mathcal{S} = \emptyset$, then $\lambda_j = \infty$ if and only if there is no path from o to j.

- Sufficient condition.
- Assume by contradiction that there is a path from o to i.
- The algorithm will set all the labels along the path to a finite number as $\lambda_o = 0$.
- Necessary condition: contrapositive of the condition: If $\lambda_i \neq \infty$, it is the length of one path from o to i.

Termination

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \; \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If algorithm terminates with $\mathcal{S} = \emptyset$, then $\lambda_o = 0$.

- λ_o cannot increase: $\lambda_o \leq 0$.
- If $\lambda_o < 0$, there is a cycle with negative cost.
- The algorithm will not stop with $\mathcal{S} = \emptyset$.

Termination

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \ \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If algorithm terminates with $\mathcal{S} = \emptyset$, then if $\lambda_j \neq \infty$, then λ_j is the length of the shortest path from o and j.

- Consider ℓ. λ_ℓ is the length of a path P_ℓ.
- Consider any path Q from o to ℓ: $o \rightarrow j_1 \rightarrow j_2 \cdots \rightarrow \ell$.
- $C(Q) \geq \lambda_\ell - \lambda_o$.
- As $\mathcal{S} = \emptyset$, $\lambda_o = 0$, so that $C(Q) \geq \lambda_\ell = C(P_\ell)$.

Termination

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \; \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

If $\mathcal{S} \neq \emptyset$, choose $i \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- If $\lambda_j <$ lower bound: STOP,
- otherwise $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If algorithm terminates with $\mathcal{S} = \emptyset$, then for all $j \neq o$ such that $\lambda_j \neq \infty$

$$\lambda_j = \min_{(i,j) \in \mathcal{A}} (\lambda_i + c_{ij}).$$

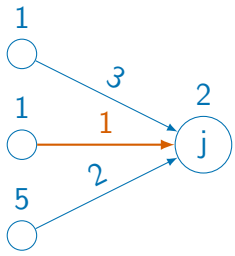
As the labels are the length of the shortest path, they must verify the optimality conditions:

$$\lambda_j - \lambda_i \leq c_{ij}, \quad \forall (i,j) \in \mathcal{A}.$$

$$\lambda_j - \lambda_i = c_{ij}, \quad \forall (i,j) \in P.$$

Bellman's equation

$$\lambda_j = \min_{(i,j)\in\mathcal{A}} (\lambda_i + c_{ij}).$$



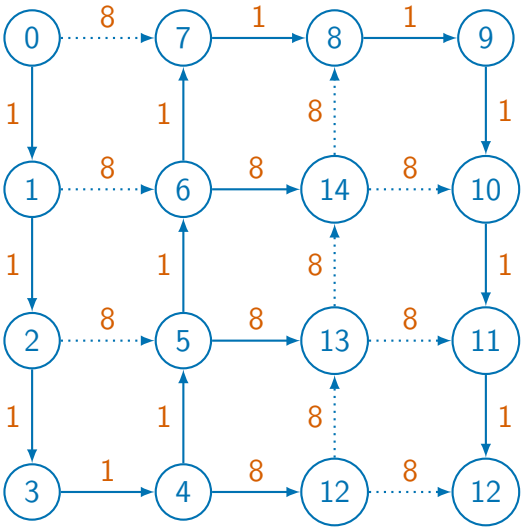
Bellman's subnetwork

- For each $j \neq o$, select one (i, j) verifying the equation.
- If several, choose just one.
- Because m nodes, there will be $m - 1$ arcs.
- Assume that every cycle in the network (if any) has positive length.
- Assume that the Bellman subnetwork contains a cycle. Its length is

$$c_{i_1 i_2} + c_{i_2 i_3} + \cdots + c_{i_{\ell-1} i_\ell} + c_{i_\ell i_1} = \lambda_{i_2} - \lambda_{i_1} + \lambda_{i_3} - \lambda_{i_2} + \cdots + \lambda_{i_\ell} - \lambda_{i_{\ell-1}} + \lambda_{i_1} - \lambda_\ell = 0,$$

- So there is no cycle and it is a tree.
- Therefore, there is only one path from o to any i.
- As the optimality conditions apply, it is the shortest path.
- The subnetwork is called the shortest path tree.

Bellman's subnetwork



Dijkstra's algorithm

Motivation

- In many applications, the cost on the arcs are all non negative.
- In that case, the shortest path algorithm can be designed to be efficient.
- In particular, it is possible to guarantee that each node will be treated at most once.
- This version of the algorithm is called Dijkstra's algorithm, from the name of a Dutch researcher.

Non negative costs

Cycles

- No cycle with negative cost.
- No need to check if the problem is unbounded.

Node selection

Select the node in $\mathcal{S}$ with the smallest label.

Algorithm

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \; \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

Choose $i \in \mathcal{S}$ such that $\lambda_i \leq \lambda_j$, for all $j \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

Permanent labels

$$\mathcal{T} = \{i | \lambda_i \neq \infty \text{ and } i \notin \mathcal{S}\}.$$

At the end of each iteration

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \; \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

Choose $i \in \mathcal{S}$ such that $\lambda_i \leq \lambda_j$, for all $j \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If $i \in \mathcal{T}$ and $j \notin \mathcal{T}$, then $\lambda_i \leq \lambda_j$.

- First iteration: o is the only node to be treated.
- For all j updated: $\lambda_j = c_{oj} \geq 0 = \lambda_o$.
- Iteration treating $\ell \notin \mathcal{T}$: assume property holds, and $\lambda_\ell \leq \lambda_j \forall j \notin \mathcal{T}$.
- Treat arc (ℓ, m), $m \in \mathcal{T}$: $\lambda_m \leq \lambda_\ell + c_{\ell m}$. No update (see next property).
- arc (ℓ, m) $m \notin \mathcal{T}$: if updated $\lambda_m = \lambda_\ell + c_{\ell m}$. As λ_ℓ was larger than any label in $\mathcal{T}$, the same holds

At the end of each iteration

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \ \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

Choose $i \in \mathcal{S}$ such that $\lambda_i \leq \lambda_j$, for all $j \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If $i \in \mathcal{T}$ at the beginning of the iteration, then the label λ_i is not modified during the iteration.

Was shown before

At the end of each iteration

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \; \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

Choose $i \in \mathcal{S}$ such that $\lambda_i \leq \lambda_j$, for all $j \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If $i \in \mathcal{T}$ at the beginning of the iteration, then $i \notin \mathcal{S}$ at the end of the iteration.

Corollary of the previous result

At the end of each iteration

Initialization

- $\lambda_o \leftarrow 0$,
- $\lambda_i \leftarrow +\infty \; \forall i \in \mathcal{N}, i \neq o$,
- $\mathcal{S} \leftarrow \{o\}$.

Choose $i \in \mathcal{S}$ such that $\lambda_i \leq \lambda_j$, for all $j \in \mathcal{S}$

$\forall (i,j)$, if $\lambda_j > \lambda_i + c_{ij}$

- $\lambda_j \leftarrow \lambda_i + c_{ij}$,
- $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$

If $i \in \mathcal{T}$, then λ_i is the length of the shortest path from o to i.

- Because the label is permanent.
- This is the interpretation at the end of the algorithm. The value will be the same.

Longest path

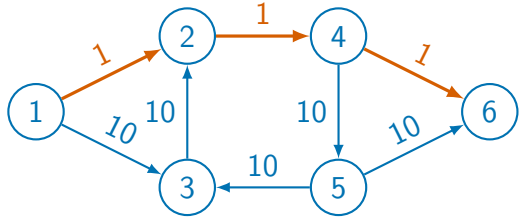
Motivation

- In optimization, we can change a minimization problem into a maximization problem.
- We analyze the implications for the longest path problem.

Definition

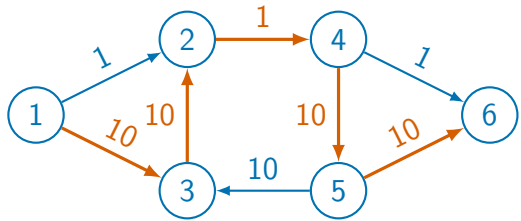
Shortest path problem

Find a simple forward path between o and d with the minimal cost.



Longest path problem

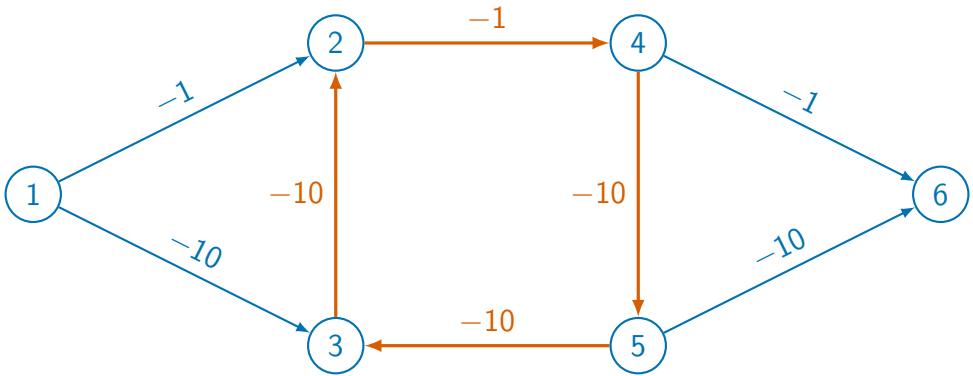
Find a simple forward path between o and d with the maximal cost.



Transhipment problem

$$\max_{x\in\mathbb{R}^n} \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \Longleftrightarrow \min_{x\in\mathbb{R}^n} -\sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} = \sum_{(i,j)\in\mathcal{A}} (-c_{ij})x_{ij}.$$

Risk of negative cycle



Program Evaluation and Review Technique

Motivation

- We consider an application that can be modeled as a longest path problem, with a network without any cycle.
- In that case, the problem can be solved using the shortest path algorithm on the network where the signs of the costs are changed.

PERT

Program Evaluation and Review Technique

- Project composed m tasks.
- Each task has a duration.
- Each task has a list of predecessors.
- What is the minimum duration of the project?
- What are the tasks that do not tolerate any delay without delaying the project?

PERT

Program Evaluation and Review Technique

- Project composed m tasks.
- Each task has a duration.
- Each task has a list of predecessors.

- What is the minimum duration of the project?
- What are the tasks that do not tolerate any delay without delaying the project?

Example: prepare a pizza

1. Buy the ingredients (30 minutes).
2. Prepare the sauce (20 minutes) [1].
3. Prepare the dough (4 hours) [1].
4. Bake (12 minutes) [2,3].

Network

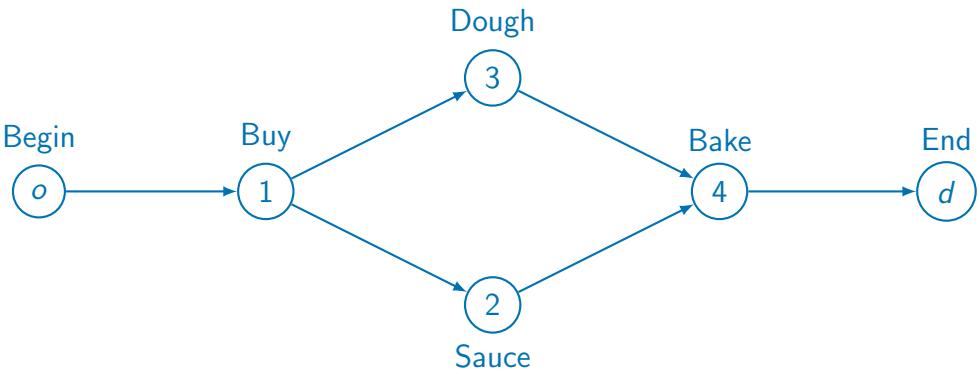
One node per task

Network

Begin and end

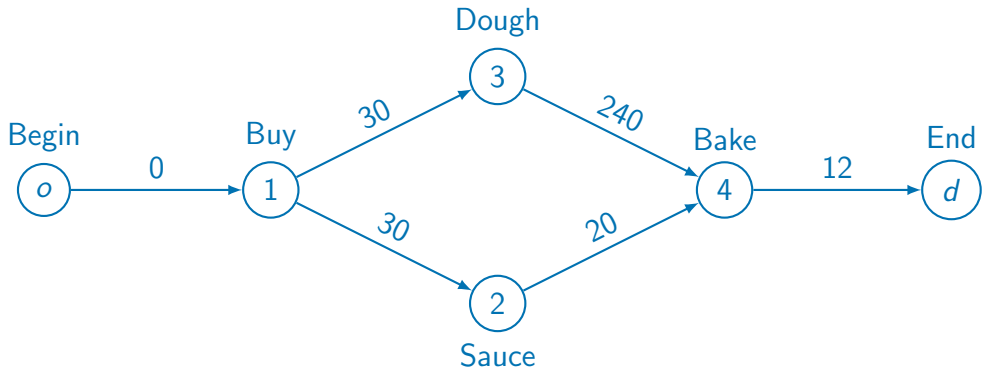
Network

One arc per precedence



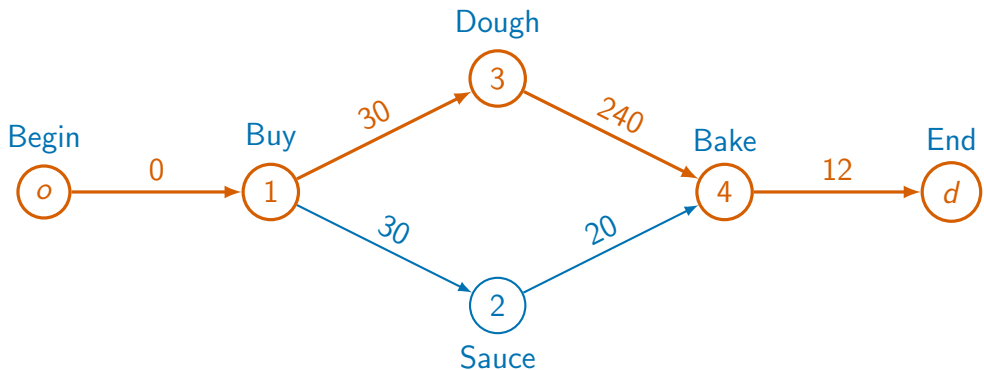
Network

Cost on arc: task duration



Longest path

- What is the minimum duration of the project?
- What are the tasks that do not tolerate any delay without delaying the project?



Summary

- Principle of optimality.
- Dual algorithm based on complementarity slackness conditions.
- Converges if there is no cycle with negative cost.
- Properties: interpretation of the dual variables.
- Bellman's equation.
- Bellman's subnetwork and the shortest path tree.
- Special case: Dijkstra.
- The longest path problem and PERT.