Shortest path problem

Properties and algorithms

Michel Bierlaire

Introduction to optimization and operations research

EPFL

The shortest path problem

Given:

- A directed network $(\mathcal{N}, \mathcal{A})$, where:
  - $\mathcal{N}$ is the set of nodes,
  - $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of arcs,
- a cost function $c : \mathcal{A} \to \mathbb{R}$,
- a source node $o \in \mathcal{N}$.

Find a shortest forward path from the source node o to every other node $v \in \mathcal{N}$, that is, for each v, find a path $P_{o \to v}$ minimizing the total cost:

$$\min_{P_{o \to v}} \sum_{(i,j) \in P_{o \to v}} c_{ij}.$$

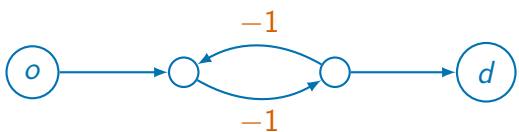
Motivation

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 introduce an algorithm based on duality that exploits the structure of the problem.
- Before, we clarify the difference between “shortest path” and “shortest simple path”.

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.



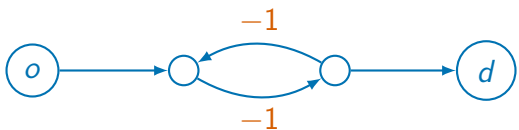
Simple paths

- 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.

Simple paths

The reverse is not true



- The number of simple paths is bounded. So there is always a shortest simple paths.
- The number of paths is not necessarily bounded. So there may be no shortest path.

Shortest path vs shortest simple path

- If the network is connected, the shortest path exists if and only if there is no negative cost cycle.
- Finding a shortest path, if it exists, is easy.
- This is what we cover in this lecture.
- And it is a simple path.
- If no shortest path exists, finding a shortest simple path is difficult.
- It requires algorithms covered in the Chapter "Discrete Optimization".
- We check the existence of a negative cost cycle by using the following lower bound.

Lower bound on the length of shortest simple paths

Corollary

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

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

Denote $\alpha = \min_{(i,j)\in\mathcal{A}} c_{ij}$.

$$C(P) = \sum_{(i,j)\in P} c_{ij} \geq \alpha \sum_{(i,j)\in P} 1 \geq \alpha(m-1),$$

as $\sum_{(i,j)\in P} 1$ is the number of arcs in path P.

Detection of negative cost cycle

Property

- If there is no negative cost cycle,
- there is a shortest simple path P such that $C(P) \geq (m-1)\min_{(i,j)\in\mathcal{A}} c_{ij}$.
- Therefore, every simple path Q is such that $C(Q) \geq C(P) \geq (m-1)\min_{(i,j)\in\mathcal{A}} c_{ij}$.
- Therefore, every path Q is such that $C(Q) \geq (m-1)\min_{(i,j)\in\mathcal{A}} c_{ij}$.

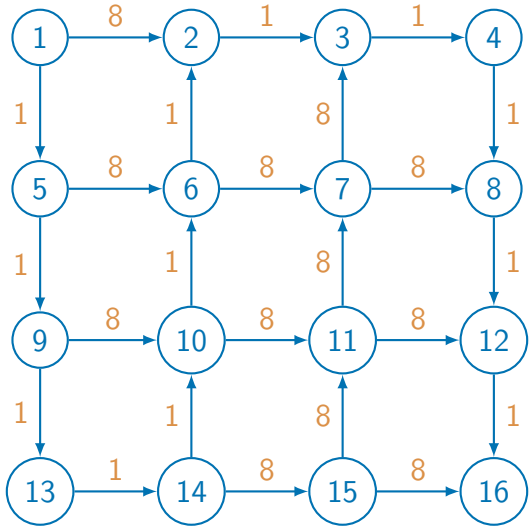
Contrapositive

- If there exists any path Q such that

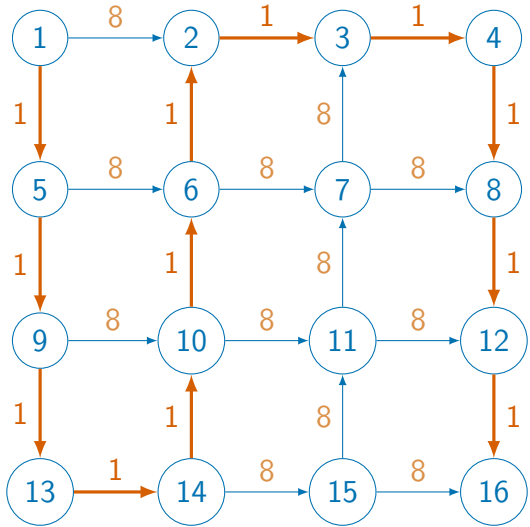
$$C(Q) < (m-1)\min_{(i,j)\in\mathcal{A}} c_{ij},$$

then there is a negative cost cycle.

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



Example

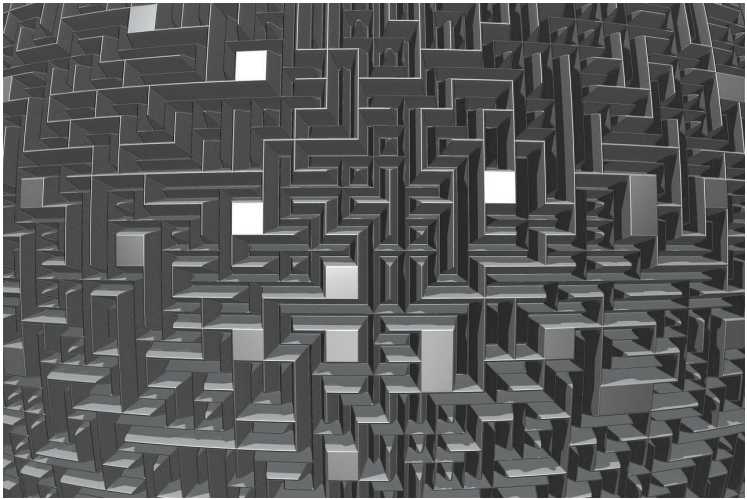


Comments

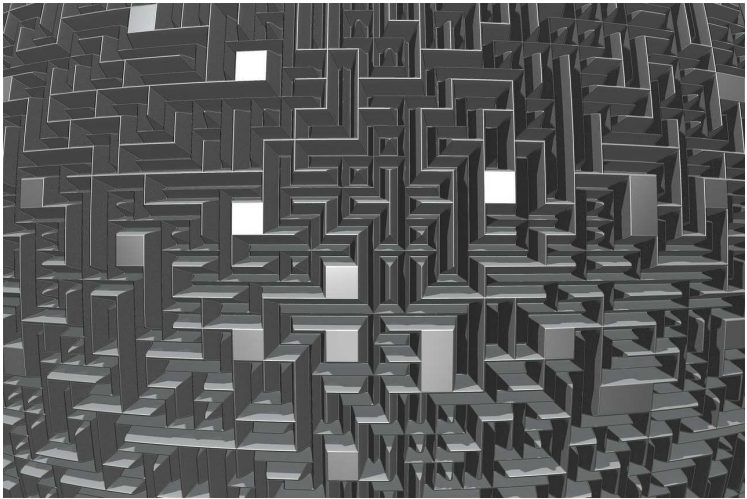
Map-based intuition

- 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

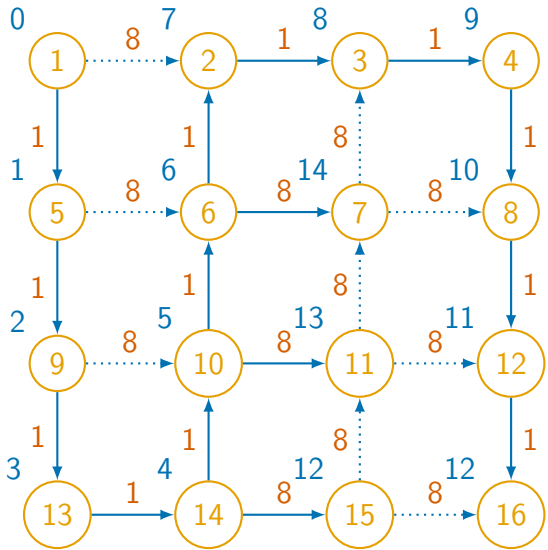
$$Q = o \to j_1 \to j_2 \ldots j_\ell \to d.$$
$$C(Q) = c_{oj_1} + c_{j_1 j_2} + \ldots + c_{j_\ell d}.$$
$$C(Q) \geq (\lambda_{j_1} - \lambda_o) + (\lambda_{j_2} - \lambda_{j_1}) + \ldots + \lambda_d - \lambda_{j_\ell} = \lambda_d - \lambda_o,$$
$$P = o \to i_1 \to i_2 \ldots i_k \to d.$$
$$C(P) = c_{oi_1}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

Example

On the next slide...

- The condition is verified with equality on the plain arcs,
- and with strict inequality on the dotted arcs.
- The plain arcs correspond to the shortest paths.
- The subnetwork of plain arcs form a spanning tree.
- 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 $\lambda_o = 0$.

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

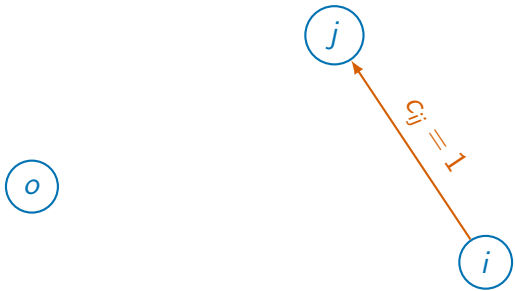
Main idea

- The optimality conditions are based on labels associated with each node.
- Idea of the algorithm: update labels so that they 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.

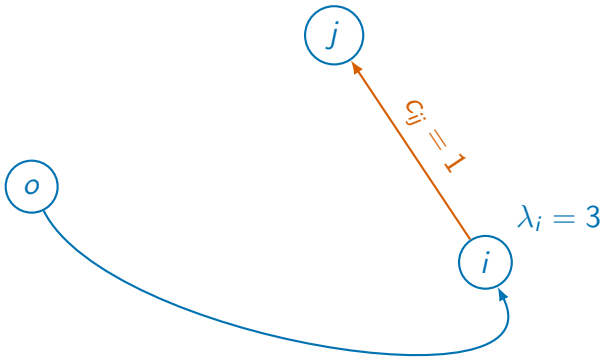
Optimality conditions

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

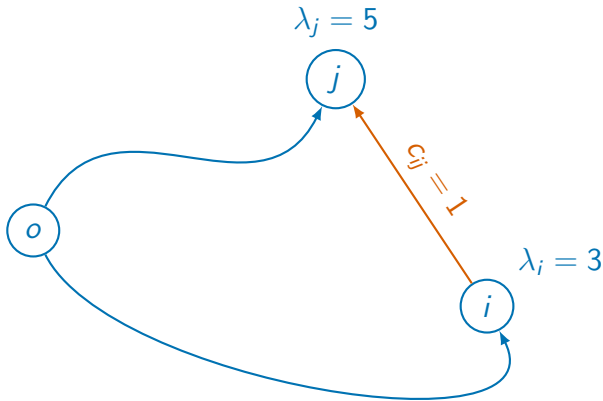
Labels



Labels

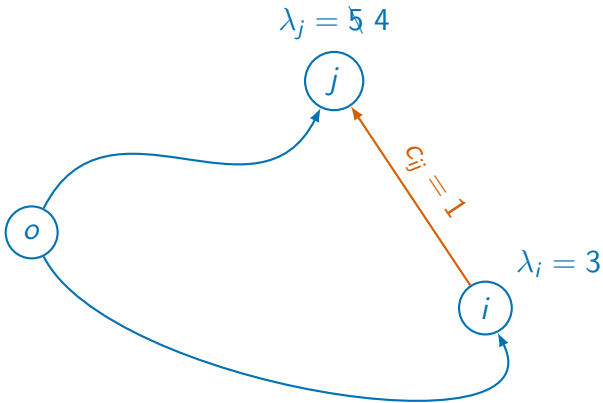


Labels



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

Labels



Idea of the algorithm

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

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

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

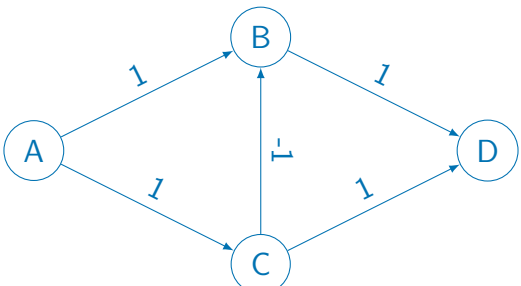
Second idea

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	$0(C)$	$1(A)$	$1(B)$

Summary

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 $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$.

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

Properties at the end of each iteration

- If $i \in \mathcal{S}$, then $\lambda_i \neq \infty$.
- For each node i, the value of λ_i does not increase during the iteration.
- If $\lambda_i \neq \infty$, it is the length of one path from o to i.
- If $i \notin \mathcal{S}$, then
  - $\lambda_i = \infty$, or
  - $\lambda_j \leq \lambda_i + c_{ij}$, for all j such that $(i, j) \in \mathcal{A}$.

Termination

Two stopping criteria

- $\mathcal{S} = \emptyset$, or
- $\lambda_j < 0$ and $\lambda_j < (m-1) \min_{(i,j) \in \mathcal{A}} c_{ij}$.

Properties

Finite number of iterations

- Suppose, for contradiction, that the algorithm performs an infinite number of iterations.
- Then the set $\mathcal{S}$ is never empty, so at least one node is added infinitely often.
- Each time a node j is added, its label λ_j is updated to a strictly smaller value.
- Thus, $\lambda_j \to -\infty$.
- However, the algorithm includes a stopping rule that prevents this:
- If $\lambda_j < 0$ and $\lambda_j < (m-1)\min_{(i,j)\in\mathcal{A}} c_{ij}$, the algorithm halts.
- Contradiction. Therefore, the algorithm terminates after a finite number of steps.

Properties upon termination

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

- (Sufficiency) Suppose, for contradiction, that there is a path from o to j, but $\lambda_j = \infty$.
- Since $\lambda_o = 0$, the algorithm will propagate finite labels along the path to j.
- This implies that λ_j must eventually receive a finite value, contradicting $\lambda_j = \infty$.
- (Necessity) If $\lambda_j \neq \infty$, then the algorithm has found a path from o to j with finite cost.
- Thus, there must exist at least one such path.
- By contrapositive: if no path from o to j exists, then $\lambda_j = \infty$.

Properties upon termination

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

- By initialization, $\lambda_o = 0$, and it can only decrease during the algorithm.
- If $\lambda_o < 0$, this implies the presence of a negative cost cycle reachable from o.
- In such a case, the algorithm would not terminate with $\mathcal{S} = \emptyset$, as labels would continue to decrease until the lower bound is violated.
- Therefore, when the algorithm terminates, it must be that $\lambda_o = 0$.

Properties upon termination

If the algorithm terminates with $\mathcal{S} = \emptyset$, then for any node j such that $\lambda_j \neq \infty$, λ_j is the length of the shortest path from o to j.

- Consider a node ℓ with $\lambda_\ell \neq \infty$.
- Let P_ℓ be the path from o to ℓ used to compute λ_ℓ during the algorithm. Then $\lambda_\ell = C(P_\ell)$, the cost of that path.
- Take any other path Q from o to ℓ, say $o \to j_1 \to j_2 \to \cdots \to \ell$.
- Since all label updates respect $\lambda_{j_k} \leq \lambda_{j_{k-1}} + c_{j_{k-1},j_k}$, we get: $\lambda_\ell \leq C(Q)$.
- Thus, λ_ℓ is less than or equal to the cost of any path from o to ℓ.
- Therefore, λ_ℓ equals the minimum possible cost: the length of the shortest path.

Properties upon termination

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

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

This condition reflects the fact that the labels satisfy the optimality conditions:

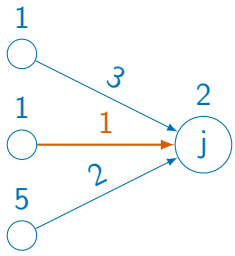
$$\lambda_j \leq \lambda_i + c_{ij}, \quad \forall (i,j) \in \mathcal{A}.$$

For any arc (i,j) that belongs to a shortest path, equality holds:

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

Bellman's equation

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



Bellman's subnetwork

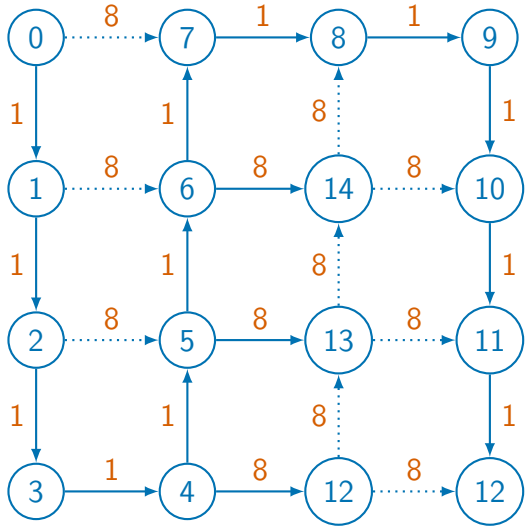
How to construct it?

- For each node $j \neq o$, select one arc (i, j) satisfying:

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

- If multiple arcs satisfy the condition, choose any one of them.

Bellman's subnetwork



Bellman's Subnetwork

Property

If all cycles in the original network have strictly positive cost, the Bellman subnetwork is a tree.

Bellman's subnetwork

Proof

To show that it is a tree, we show that

- it contains $n = m - 1$ arcs,
- it has no simple cycle.

Proof: number of arcs

As there are m nodes and one arc is selected for each node $j \neq o$, the subnetwork contains $m - 1$ arcs.

Bellman's Subnetwork

Proof: no simple cycle

- Suppose, by contradiction, that the Bellman subnetwork contains a simple cycle:
$$(i_1 \to i_2 \to \cdots \to i_\ell \to i_1).$$
- Then the total cost of the cycle is:
$$c_{i_1 i_2} + c_{i_2 i_3} + \cdots + c_{i_{\ell-1} i_\ell} + c_{i_\ell i_1},$$
- Using the Bellman equalities, we get:
$$\lambda_{i_2} - \lambda_{i_1} + \lambda_{i_3} - \lambda_{i_2} + \cdots + \lambda_{i_1} - \lambda_{i_\ell} = 0.$$
- Hence, the cycle has zero cost, contradicting the assumption.

Bellman's subnetwork

Shortest path tree

- The Bellman subnetwork is a tree.
- This tree contains exactly one path from o to any node j.
- As each arc in the path satisfies the optimality condition, the path is the shortest.
- This subnetwork is called the **shortest path tree**.

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

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

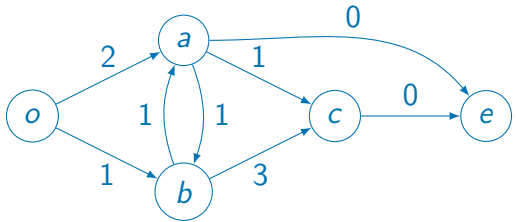
Choose $i \in \mathcal{S}$ such that $\lambda_i \leq \lambda_j$, for all $j \in \mathcal{S}$

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

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

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

Example



$\mathcal{S}$	i	λ_o	λ_a	λ_b	λ_c	λ_e
$\{ o \}$	o	0	∞	∞	∞	∞
$\{ a,b \}$	b	0	2 (o)	1 (o)	∞	∞
$\{ a,c \}$	a	0	2 (o)	1 (o)	4 (b)	∞
$\{ c,e \}$	e	0	2 (o)	1 (o)	3 (a)	2 (a)
$\{ c \}$	c	0	2 (o)	1 (o)	3 (a)	2 (a)
$\{ \}$		0	2 (o)	1 (o)	3 (a)	2 (a)

Permanent labels

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

Property

At the end of each iteration, for all $i \in \mathcal{T}$ and $j \notin \mathcal{T}$, we have $\lambda_i \leq \lambda_j$.

Intuition

Nodes in $\mathcal{T}$ are closer to the origin than nodes not in $\mathcal{T}$.

Property

At the end of the first iteration

- The origin o has been treated.
- $\mathcal{T} = \{o\}$.
- For each j such that $(o, j) \in \mathcal{A}$, the update rule sets $\lambda_j = c_{oj} \geq 0 = \lambda_o$, so the property holds.
- For all other j, $\lambda_j = +\infty$ and the property holds as well.

Property

Induction: Suppose the property holds before an iteration

- Let ℓ be the node selected such that $\lambda_\ell = \min_{j \in \mathcal{S}} \lambda_j$. Then ℓ is added to $\mathcal{T}$.
- Note that $\lambda_\ell \geq \lambda_i$, $\forall i \in \mathcal{T}$, by the property before the iteration.
- For each arc (ℓ, m):
  - If $m \in \mathcal{T}$, then
    - $\lambda_m \leq \lambda_\ell$, as the property holds before the iteration.
    - $\lambda_m \leq \lambda_\ell + c_{\ell m}$, as $c_{\ell m} \geq 0$.
    - Therefore, λ_m is not updated.
    - Important: once in $\mathcal{T}$, labels don't change any more.
  - If $m \notin \mathcal{T}$ and λ_m not updated. Then, $\lambda_m \geq \lambda_\ell$, by the node selection rule.
  - If $m \notin \mathcal{T}$ and λ_m updated. Then, $\lambda_m = \lambda_\ell + c_{\ell m}$, and $\lambda_m \geq \lambda_\ell$.
  - Therefore, if $m \notin \mathcal{T}$,

$$\lambda_m \geq \lambda_\ell \geq \lambda_i, \forall i \in \mathcal{T}.$$

Property

Summary

When a node ℓ is treated, the following holds for each arc (ℓ, m):

- If m is already permanent, then λ_m is not updated.
- If m is not yet permanent, then $\lambda_m \geq \lambda_\ell$ after the update.

The node ℓ becomes permanent after treatment, and its label λ_ℓ is less than or equal to the label of any non-permanent node.

Properties

We have shown two properties

- At the end of each iteration, for all $i \in \mathcal{T}$ and $j \notin \mathcal{T}$, we have $\lambda_i \leq \lambda_j$.
- If $i \in \mathcal{T}$ before the iteration, λ_i is not modified during the iteration.

Permanent labels

- As a consequence, if $i \in \mathcal{T}$ before the iteration, i is not included in $\mathcal{S}$ and stays in $\mathcal{T}$ until the end of the algorithm.
- Therefore, if $i \in \mathcal{T}$, λ_i is the length of the shortest path from o to i.
- $\mathcal{T}$ is called the set of permanent labels.
- Consequently, each node is treated at most once by the algorithm.

Longest path

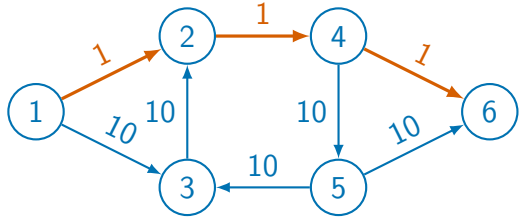
Motivation

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

Definition

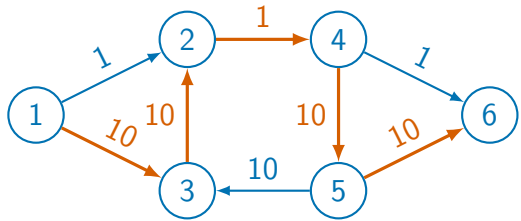
Shortest path problem

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



Longest path problem

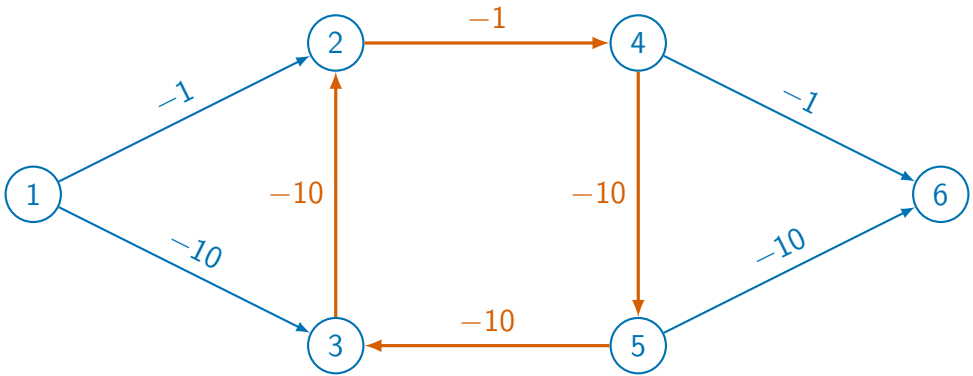
Find a simple forward path between o and d with the maximal cost.



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

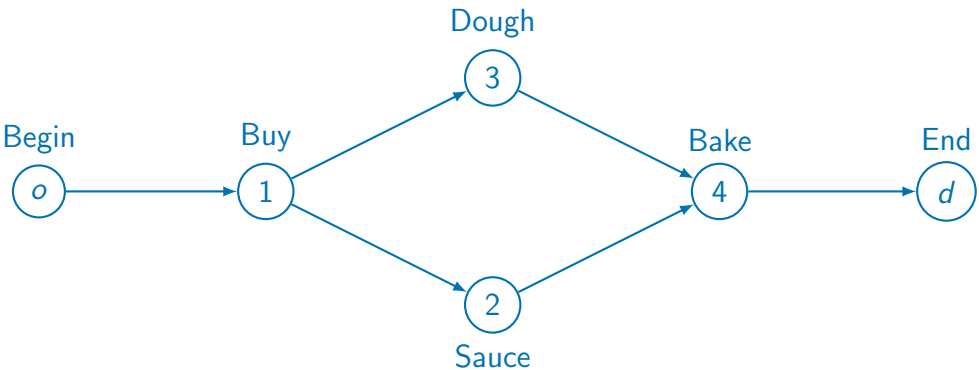
One node per task

Network

Begin and end

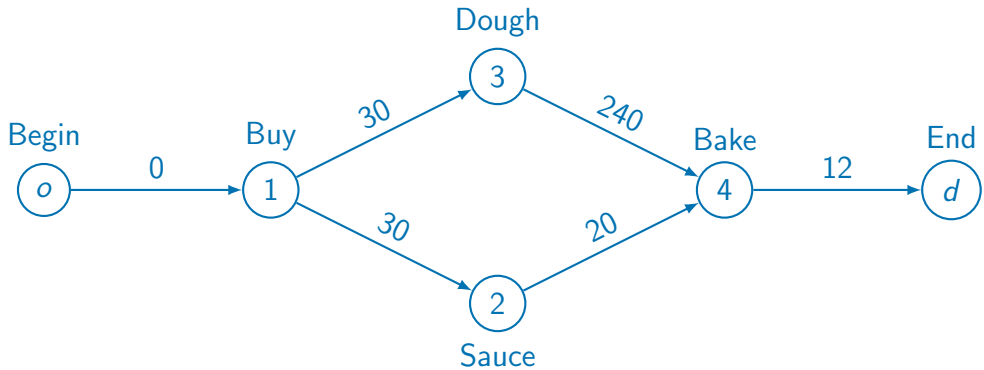
Network

One arc per precedence



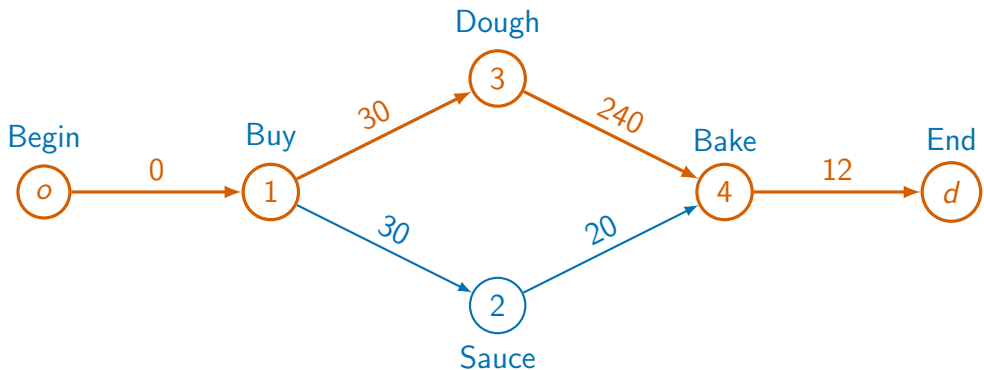
Network

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.