Shortest Path Problems

Optimization Methods in Management Science
Master in Management
HEC Lausanne

Dr. Rodrigue Oeuvray

Fall 2019 Semester

Table of Contents

Shortest Path Problems in a Network

- A **network** is a directed graph $G = (V, E)$ where vertices and/or edges have attributes
- We consider here a network $R = (V, E, c)$ where c is a **weighting** of the arcs of the digraph $G = (V, E)$
- In a network, the **length** of a path or of a circuit corresponds to the **sum of the weights** of its edges and not to their number !

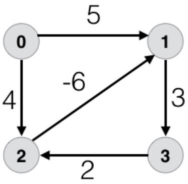
Shortest Path Problems in a Network (Cont'd)

Important Assumption

Shortest path problems are well defined only if there is **no circuit with a negative length**. From now on, we will assume that this is always the case

Shortest Path Problems in a Network (Cont'd)

- The network below contains a negative cycle of length -1 comprised of vertices 1,3,2
- There is an infinite number of paths i between vertices 0 and 1
  - Path 1: (0,(0,1),1)
  - Path 2: (0,(0,1),1,(1,3),3,(3,2) 2,(2,1),1)
  - Path 3: (0,(0,1),1,(1,3),3,(3,2) 2,(2,1),1,(1,3),3,(3,2),2,(2,1),1)
  - ...
- Length of these paths : $5, 4, 3, \ldots$

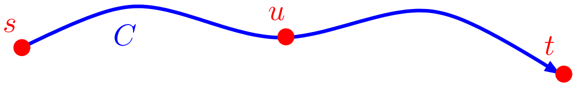


Shortest Versus Longest Path Problems

- The **longest path problem** is the problem of finding a simple path of maximum length in a given graph
- In contrast to the shortest path problem, which can be solved in **polynomial time** in graphs without negative-cost cycles, the longest path problem is **NP-hard**
- However, it has a **linear time** solution for **directed acyclic graphs**, which has important applications in finding **the critical path** in scheduling problems

Bellman's Principle of Optimality

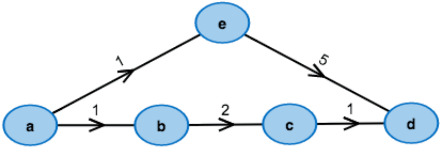
- This principle is the basis of optimization techniques such as **dynamic programming** and **labeling algorithms**
- In the context of shortest paths problems, this principle simply states that a shortest path consists of shortest paths



- Concretely, if C is a shortest path from s to t and if u belongs to this path, then sub-paths from s to u and from u to t are also shortest paths

Bellman's Principle of Optimality (Cont'd)

- **This doesn't mean that the union of shortest paths is a shortest path !**
- (a, e) is a shortest path between a and e, (e, d) is a shortest path between e and d but the shortest path between a and d is not $(a, (a, e), e, (e, d), d)$



- The shortest path between a and d is $(a, (a, b), b, (b, c), c, (c, d), d)$ and has a length of 4

Optimality Conditions for the Shortest Path Problem

Let $R = (V, E, c)$ be a network with $|V| = n$ and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n)$ a scalar vector associated with vertices of R and satisfying $\lambda_i \in \mathbb{R} \cup \{\infty\}$ for all $i \in V$

Theorem

If $\lambda_1, \ldots, \lambda_n$ satisfy

$$\lambda_j \leq \lambda_i + c_{ij} \qquad \forall (i,j) \in E \tag{1}$$

and if C is a path from s to t for which

$$\lambda_j = \lambda_i + c_{ij} \qquad \forall (i,j) \in C \tag{2}$$

then C is the shortest path between s and t

Optimality Conditions: Proof (Cont'd)

Proof. The proof is divided in two parts

1. We first show that $\lambda_t - \lambda_c$ is the length of C
2. We consider a path C' from s to t and we show that its length is longer than $\lambda_t - \lambda_c$

Part 1:

By replacing iteratively $\lambda_j = \lambda_i + c_{ij}$ (Equation 2) for all the vertices of C starting from t, we get

$$\lambda_t = \lambda_s + \sum_{(i,j)\in C} c_{ij} \quad \Longleftrightarrow \quad \lambda_t - \lambda_s = \sum_{(i,j)\in C} c_{ij}$$

and we conclude that $\lambda_t - \lambda_s$ is equal to the length of C

Optimality Conditions: Proof (Cont'd)

Part 2.

Let C' be a path from s to t and C a path between s and t for which $\lambda_j = \lambda_i + c_{ij}$. From Part 1, we know that $\lambda_t - \lambda_s$ is the length of C. By replacing iteratively $\lambda_j \leq \lambda_i + c_{ij}$ (Equation 1) for all the vertices of C' starting from t, we get

$$\lambda_t \leq \lambda_s + \sum_{(i,j)\in C'} c_{ij} \iff \lambda_t - \lambda_s \leq \sum_{(i,j)\in C'} c_{ij}$$

Consequently, the length of C' is larger than the length of C and C is a shortest path between s and t of length $\lambda_t - \lambda_s$

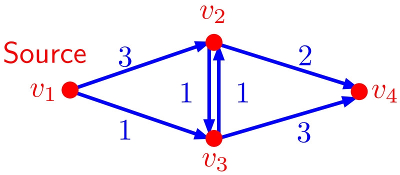
A Generic Algorithm for the Shortest Path Problem

It is possible to develop a **generic** algorithm computing shortest paths between s and all the other vertices of the network from conditions (1) and (2). We define a list L of vertices for which a shortest path has been found.

(1) We start from an initial vector $\boldsymbol{\lambda}$ defined by $\lambda_s = 0$ and $\lambda_i = \infty$ for all $i \neq s$. $L = \{s\}$

(2) While L is non empty, we remove a vertex from L, let's say i, and for each of its successors j, we test if $\lambda_j > \lambda_i + c_{ij}$. If it is the case, we set $\lambda_j = \lambda_i + c_{ij}$ and we put j in L (except if it is already present in L)

Example

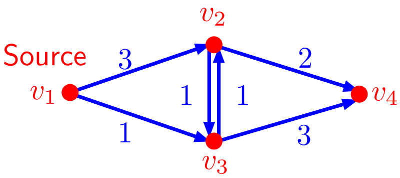


Iteration	v_i removed from L	Labels λ_i	Candidates L
0		$(0, \infty, \infty, \infty)$	$\{v_1\}$
1	v_1	$(0, 3, 1, \infty)$	$\{v_2, v_3\}$
2	v_2	$(0, 3, 1, 5)$	$\{v_3, v_4\}$
3	v_3	$(0, 2, 1, 4)$	$\{v_4, v_2\}$
4	v_4	$(0, 2, 1, 4)$	$\{v_2\}$
5	v_2	$(0, 2, 1, 4)$	$\emptyset$

Note that a vertex can be introduced **several times** in L. At iteration 1, we have introduced v_2 in L as well as in iteration 3

Example (Cont'd)

In order to build the shortest paths, we need to add the predecessor $p(i)$ beside each of the labels:

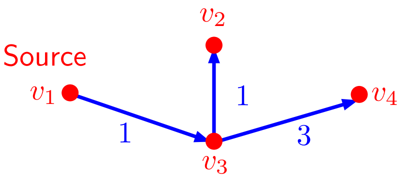


Iter.	v_i	Labels λ_i/p_i	L
0		$(0/\texttt{NULL}, \infty/\texttt{NULL}, \infty/\texttt{NULL}, \infty/\texttt{NULL})$	$\{v_1\}$
1	v_1	$(0/\texttt{NULL}, 3/v_1, 1/v_1, \infty/\texttt{NULL})$	$\{v_2, v_3\}$
2	v_2	$(0/\texttt{NULL}, 3/v_1, 1/v_1, 5/v_2)$	$\{v_3, v_4\}$
3	v_3	$(0/\texttt{NULL}, 2/v_3, 1/v_1, 4/v_3)$	$\{v_4, v_2\}$
4	v_4	$(0/\texttt{NULL}, 2/v_3, 1/v_1, 4/v_3)$	$\{v_2\}$
5	v_2	$(0/\texttt{NULL}, 2/v_3, 1/v_1, 4/v_3)$	$\emptyset$

Example (Cont'd)

Determination of the shortest paths:

- We start from the final destination and we go back to the source by using labels $p(i)$
- The set of arcs that are selected forms a tree, the **shortest path tree**
- Indeed, it is an **arborescence**, i.e. a directed graph in which there is exactly one path from the source u to any other vertex v



Remarks about the Generic Algorithm

- The generic algorithm stops after a finite number of iterations if and only if there is no path starting at s with a negative circuit
- When the algorithm stops, λ_j is the length of shortest path between s and j if $\lambda_j < \infty$. Moreover, $\lambda_j = \infty$ if and only if there is no path between s and j
- The genericity of this algorithm comes from the **absence of rule** specifying the choice of the vertex to remove from L at each iteration

Non-Negative Weightings

Let $R = (V, E, c)$ be a network where $c : E \to \mathbb{R}_+$ is a **non-negative weighting** of the arcs of the digraph $G = (V, E)$. Then we can define an **optimal selection rule** to determine which vertex to remove from L. With this rule, the number of iterations is minimized compared to the generic algorithm

Theorem

*Let $R = (V, E, c)$ be a network where $c : E \to \mathbb{R}_+$ is a non-negative weighting of its arcs. If we remove the vertex with the **smallest** label from L at each iteration, this vertex is never introduced back in L once it has been removed*

Remarks

- When the vertex i is removed from L, λ_i is equal to the length of the shortest path between s and i
- Rather than updating a list of candidates L, we maintain a list T of vertices whose labels are not yet final ($T = L \cup \{j \mid \lambda_j = \infty\}$)
- This algorithm is called **Dijkstra's algorithm**

Dijkstra's Algorithm

Input: a connected network $R = (V, E, c)$, $|V| = n$, $|E| = m$, where $c : E \rightarrow \mathbb{R}_+$ is a **non-negative** weighting of the arcs of the graph $G = (V, E)$. A source $s \in V$

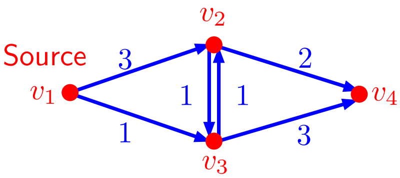
Output: for each vertex $i \in V$, the length λ of the shortest path between s and i ($\lambda_i = \infty$ if no path exists between s and i) and the direct predecessor $p(i)$ of vertex i in such a path

(1) $\lambda_s = 0$, $\lambda_i = \infty \,\forall\, i \neq s$, $p(i) = \texttt{NULL} \,\forall\, i$, $T = V$

(2) While $T \neq \emptyset$ do

- (2.1) Let i be the vertex of T with the smallest label λ_i (choose arbitrarily in case of equality)
- (2.2) If it does not exist ($\lambda_j = \infty \,\forall\, j \in T$) : STOP, vertices in T cannot be reached from s
- (2.3) If not, remove i from T and for each successor $j \in T$ of i, test if $\lambda_j > \lambda_i + c_{ij}$. If it is the case, then set $\lambda_j = \lambda_i + c_{ij}$ and $p(j) = i$

Example



Iter.	i_{min}	$\lambda_i/p(i)$				T
0		$0/\texttt{NULL}$	$\infty/\texttt{NULL}$	$\infty/\texttt{NULL}$	$\infty/\texttt{NULL}$	$\{v_1, v_2, v_3, v_4\}$
1	v_1	$0/\texttt{NULL}$	$3/v_1$	$1/v_1$	$\infty/\texttt{NULL}$	$\{v_2, v_3, v_4\}$
2	v_3		$2/v_3$	$1/v_1$	$4/v_3$	$\{v_2, v_4\}$
3	v_2		$2/v_3$		$4/v_3$	$\{v_4\}$
4	v_4				$4/v_3$	$\emptyset$

Application to a Non-Directed Graph

The Dijkstra's algorithm can also be applied to a **non-directed** graph with a **non-negative** weighting of its edges. Two possibilities:

- replace successors by adjacent vertices in the previous algorithm
- replace each edge by two arcs in the opposite direction with the same weight

Acyclic Graphs

- When a network has **no circuit**, then there exists an algorithm much more performant than the generic algorithm to determine the shortest paths
- Indeed, in a network with no circuit, the **shortest** and the **longest** paths are always well defined and, as soon as there is a path between two vertices, there exists a shortest and a longest path
- Reminder: a graph is acyclic if it has no circuit
- An acyclic graph $G = (V, E)$ has at least one vertex with **no predecessor** and one vertex with **no successor**

Topological Sort

A **topological sort** or **topological ordering** of a directed graph is an **ordering** of its vertices such that for every arc (u, v) from vertex u to vertex v, u comes before v in the ordering

Theorem

A directed graph $G = (V, E)$ has no circuit if and only if it has a topological sort of its vertices

Topological Sort: Algorithm

Input: a directed graph $G = (V, E)$ with no circuit, $|V| = n$

Output: a topological sort $\nu : V \to \{1, \ldots, n\}$ of its vertices

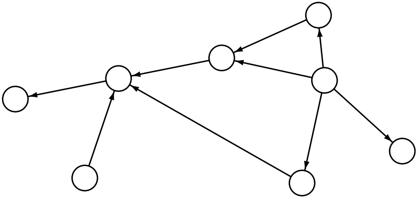
(1) $k = 1$, $W = V$

(2) While $W \neq \emptyset$ do

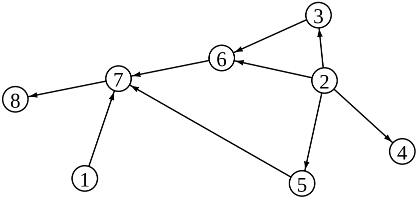
- (2.1) Let i be a vertex without a predecessor in the sub-graph $G_W = (W, E(W))$
- (2.2) Set $\nu(i) = k$, $W = W \setminus \{i\}$ and $k = k + 1$

Topological Sort Algorithm: Example

Let's apply the **topological sort** to the graph below:



Output:



Remark: the ordering based on a topological sort is not unique !

Shortest and Longest Paths in an Acyclic Graph

- We consider a graph that has no circuit and only one vertex with no predecessor, the **root**. We would like to determine the shortest and the longest paths between the root and all the other vertices

- To determine these **shortest** paths, we first apply a topological sort, then we set $\lambda_1 = 0$ for the root, and we finally compute

$$\lambda_k = \min\{\lambda_j + c_{jk} \mid j \in Pred(k)\}$$

  for $k = 2, \ldots, n$, where n is the number of vertices

- To determine the **longest** paths in a graph with no circuit, we first apply a topological sort as before, then we set $\lambda_1 = 0$ for the root, and we compute

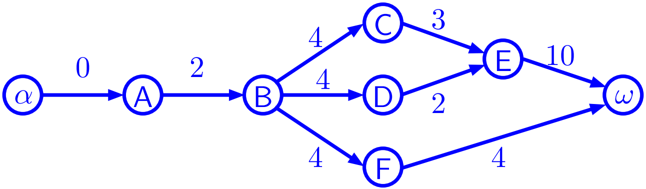
$$\lambda_k = \max\{\lambda_j + c_{jk} \mid j \in Pred(k)\}$$

  for $k = 2, \ldots, n$

Example: Shortest Path

Determine the **shortest** path between α and ω :

Vertex	α	A	B	C	D	E	F	ω
k (top. sort)	1	2	3	4	5	6	7	8
$\lambda_k/p(k)$	$0/\texttt{NULL}$	$0/\alpha$	$2/A$	$6/B$	$6/B$	$8/D$	$6/B$	$10/F$

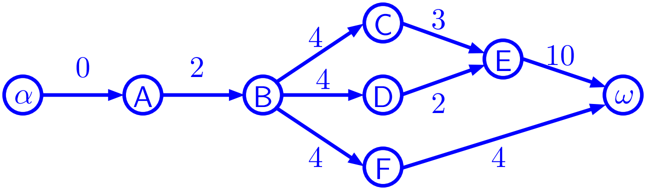


The shortest path is $(\alpha, (\alpha, A), A, (A, B), B, (B, F), F, (F, \omega), \omega)$ and has a length of 10

Example: Longest Path

Determine the **longest** path between α and ω :

Vertex	α	A	B	C	D	E	F	ω
k (top. sort)	1	2	3	4	5	6	7	8
$\lambda_k/p(k)$	$0/\texttt{NULL}$	$0/\alpha$	$2/A$	$6/B$	$6/B$	$9/C$	$6/B$	$19/E$



The longest path is $(\alpha, (\alpha, A), A, (A, B), B, (B, C), C, (C, E), E, (E, \omega), \omega)$ and has a length of 19