Chapter 3

Shortest path problems

1. Introduction

Optimal path problems are very common in everyday life. They arise when we are looking for a path between two points in a network so that a cost function is minimized. They also arise as subproblems in a wide range of combinatorial problems, especially problems with scheduling.

Several algorithms can efficiently tackle this problem. However, there is a more complex version known as the *traveling salesman problem*. The latter aims to build a Hamiltonian path with the lowest possible cost. Despite the apparent similarity, the traveling salesman problem is more difficult to solve and will not be discussed in this chapter.

It is essential to differentiate the problem of finding the shortest path from the construction of the minimum spanning tree weight. In most cases, building this tree does not mean that the paths are minimal.

2. Definitions

2.1. Networks

Definition 3.1

A *network* is a directed graph $G = (X, E)$ fo which a function $d : E \to \mathbb{R}$ defines the length (or cost) of each arc. Such network is denoted as $R = (X, E, d)$ (a real-value valued graph).

In practice, $d(e)$ may represent a transportation cost, distance, duration, and so on. It can also represent a reward in the case of a negative value. An example of such a graph is given in figure 3.1 .

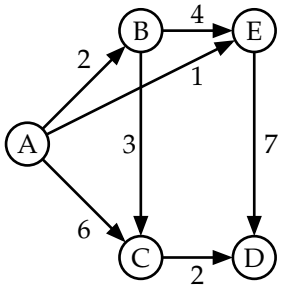

Figure 3.1 - an example of a network

2.2. Cost of a path

The cost of a path in a network R is the sum of all costs of the arcs that make it up, weighted by their multiplicity (how many times an arc is repeated).

The shortest path problem consists of computing paths with the lowest cost. In the network shown in figure 3.1 , the shortest path from vertex A to vertex D is (A, B, C, D), with a cost of $L = d(A, B) + d(B, C) + d(C, D) = 2 + 3 + 2 = 7$. In this example, the multiplicity of each arc is 1.

2.3. Absorbing circuit

A circuit is *absorbing* if the sum of the cost of its arcs is negative. Shortest path problems make no sense in this case (since the minimal cost is $-\infty$). As a result, the absence of absorbing circuits in the graph is an important requirement for shortest path discovery algorithms. The cost of the circuit (A, B, C, A) in figure 3.2 is -1. Therefore, it is an absorbing circuit.

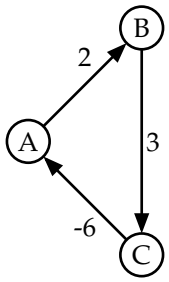

Figure 3.2 - an absorbing circuit

Similarly, the longest path problem requires that the graph not contain any circuits with a positive sum of costs.

2.4. Problem specification

If the network does not have an absorbing circuit, we can limit the search to trails (paths with no repeating edges). Actually, there are three kinds of shortest path problems (**A**, **B** and **C**):
- **A**: looking for the shortest path between two vertices x and y in a network $R(X, E, d)$.
- **B**: looking for the shortest path between any two vertices x and y in the network.
- **C**: looking for the shortest paths between a vertex x (the graph's root) and any other vertex in the network.

These problems are actually related. For example, an algorithm for **A** can be applied multiple times to solve **B** or **C**. In the sequel, we will only focus on the problem **C**. The latter admits a solution if:
- x is the root of the network.
- he network does not have an absorbing circuit.

An algorithm that solves the **C** version of this problem yields an arborescence of the shortest paths starting from the root x.

3. Dijkstra's algorithm

Consider a network $R = (X, E, d)$ that has no negative weights. Let x be a root of R. Dijskstra's algorithm calculates the cost λy, which represents the shortest path between x and y.

Algorithm of Dijkstra to calculate λs in a network $R = (X, E, d)$

$S = \{x_0\}$
$\lambda x_0 = 0$ ▷ the cost the shortest path between x_0 and x_0 is null
for each successor x_i of x_0
    $\lambda x_i = d(x_0, x_i)$
for each non-successor x_j of x_0
    $\lambda x_j = \infty$
while $S \neq X$
    Choose $x_k \notin S$ such that $\lambda x_k = \min_{x_l \notin S} \lambda x_l$
    $S = S \cup \{x_k\}$
    for each $x_m \in \Gamma^+(x_k) - S$
        $\lambda x_m = \min(\lambda x_k + d(x_k, x_m), \lambda x_m)$ ▷ the successors of x_k which are not in S

We will calculate the minimum costs of the paths in the network shown in figure 3.3 that begin at vertex X. Later on, we will build the arborescence of the shortest paths.

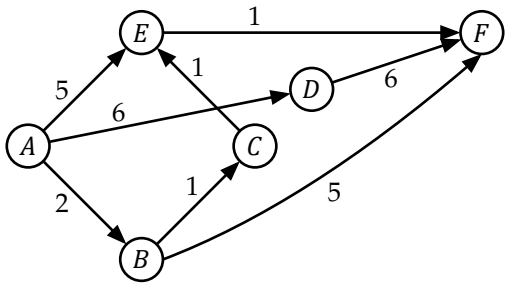

Figure 3.3 - building the shortest path that begins at A

S	λA	λB	λC	λD	λE	λF
A	0	**2**	∞	6	5	∞
A, B	0	2	**3**	6	5	9
A, B, C	0	2	3	6	**4**	9
A, B, C, E	0	2	3	6	4	**5**
A, B, C, E, F	0	2	3	**6**	4	5
A, B, C, E, F, D	0	2	3	6	4	5

Algorithm for building the arborescence of the shortest paths in a network $R = (X, E, d)$

for each edge $(x_i, x_j) \in E$
    if $\lambda x_i - \lambda x_j = d(x_i, x_j)$
        The arc (x_i, x_j) belongs **to** the arborescence

We obtain the graph in figure 3.4 , where the red arcs indicate the arborescence of the shortest paths. Each path in this arborescence represents the shortest path between vertex A and a given vertex in the network. As this is an arborescence, each path is unique.

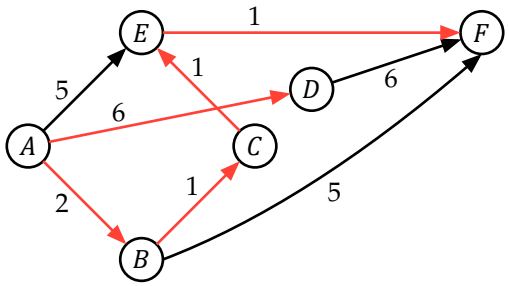

Figure 3.4 - arborescence of the optimal paths from A

4. Bellman-Ford's algorithm

The Bellman-Ford algorithm can be employed even if the network has negative weights. Actually, there are many versions of this algorithm. We will only present the version that can be applied to a network without circuits. Obviously, such a network cannot have an absorbing circuit.

The Bellman-Ford algorithm, presented here, is based on topological sorting (since the network has no circuits). It then computes the shortest path based on the rank of each vertex.

Algorithm of Bellman-Ford to compute λs in a circuit-less network $R = (X, E, d)$

```
1  Sort the vertices topologically
2  for each vertex x in rank 1
3  |  λx = 0
4  for j = 2..l                          ▷ let l be the number of ranks
5  |  for each vertex x_k in rank j
6  |  |  λx_k = min(λx_i + d(x_i, x_k)) where x_i ∈ Γ⁻(x_k)
```

Let us apply this algorithm to the graph in figure 3.5 . The ranking of vertices is given in figure ref{fig:class-bellman}.

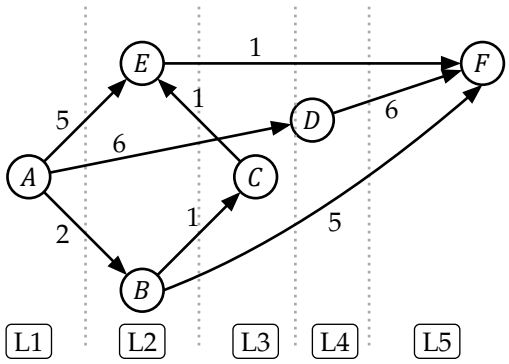

Figure 3.5 - building the shortest path that begins at A

Therefore, the Bellman-Ford's algorithm proceeds as follows:

Minimal cost	Computation
λA	0 (this is the root)
λB	$\lambda A + d(A,B) = 2$
λD	$\lambda A + d(A,D) = 6$
λC	$\lambda B + d(B,C) = 3$
λE	$\min(\lambda A + d(A,E), \lambda C + d(C,E)) = \min(5,4) = 4$
λF	$\min(\lambda E + d(E,F), \lambda D + d(D,F), \lambda B + d(B,F)) = \min(5,12,9) = 5$

After computing the λs, we use the same algorithm as in Dijkstra's method to build the arborescence of the shortest paths.