Operation Research and Transport
Shortest Path Algorithm

V. Leclère (ENPC)

February 7th, 2022

In the previous episode

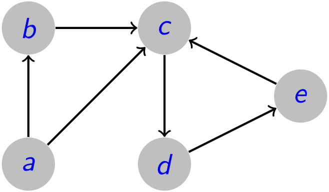
We have seen:

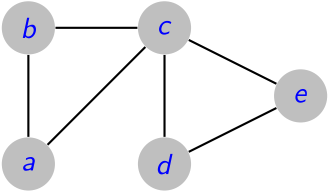
- A few definitions of game theory (Nash equilibrium, Pareto efficient points, Social optimum)
- Examples of Braess paradox
- Applications of the course in the industry

Contents

What is a Graph?

- A graph is one of the elementary modelization tools of Operation Research.
- A directed graph (V, E) is defined by
  - A finite set of n vertices V
  - A finite set of m edges each linked to an origin and a destination.
- A graph is said to be undirected if we do not distinguish between the origin and the destination.





A few definitions

Consider a directed graph (V, E).

- If $(u, v) \in E$, u is a predecessor of v, and v is a successor of u.
- A path is a sequence of edges $\{e_k\}_{k\in[\![1,n]\!]}$, such that the destination of one edge is the origin of the next. The origin of the first edge is the origin of the path, and the destination of the last edge is the destination of the path.
- A (directed) graph is connected if for all $u, v \in V$, there is a u-v-path.
- A cycle is a path where the destination vertex is the origin.

A weighted graph

- A weighted (directed) graph is a (directed) graph (V, E) with a weight function $\ell : E \to \mathbb{R}$.
- The weight of a $s - t-$path p is sum of the weights of the edges contained in the path :

$$\ell(p) := \sum_{e \in p} \ell(e).$$

- The shortest path from o to d is the path of minimal weight with origin o and destination d.
- An absorbing cycle is a cycle of strictly negative weight.

Contents

An optimality condition

The methods we are going to present are based on a label function over the vertices. This function should be understood as an estimated cost of the shortest path cost between the origin and the current vertex.

Theorem

Suppose that there exists a function $\lambda : V \mapsto \mathbb{R} \cup \{+\infty\}$, *such that*

$$\forall (i,j) \in E, \qquad \lambda_j \leq \lambda_i + \ell(i,j).$$

Then, if p *is an* s-t*-path, we have* $\ell(p) \leq \lambda(t) - \lambda(s)$[a]
In particular, if p *is such that*

$$\forall (i,j) \in p, \qquad \lambda_j = \lambda_i + \ell(i,j),$$

then p *is a shortest path.*

[a] with the convention $\infty - \infty = \infty$.

A generic algorithm

We keep a list of candidates vertices $U \subset V$, and a label function $\lambda : V \mapsto \mathbb{R} \cup \{+\infty\}$.

$U := \{o\}$;
$\lambda(o) := 0$;
$\forall v \neq o, \quad \lambda(v) = +\infty$;

while $U \neq \emptyset$ **do**
  choose $u \in U$;
  for v *successor of* u **do**
    if $\lambda(v) > \lambda(u) + \ell(u, v)$ **then**
      $\lambda(v) := \lambda(u) + \ell(u, v)$;
      $U := U \cup \{v\}$;
  $U := U \setminus \{u\}$;

Algorithm properties

- If $\lambda(u) < \infty$ then $\lambda(u)$ is the cost of a o-u-path.
- If $u \notin U$ then
  - either $\lambda(i) = \infty$ (never visited)
  - or

$$\text{for all successor } v \text{ of } u, \qquad \lambda(v) \leq \lambda(u) + \ell(u, v).$$

- If the algorithm end $\lambda(u)$ is the smallest cost to go from o to u.
- Algorithm end iff there is no path starting at o and containing an absorbing circuit.

Dijkstra's algorithm

Assume that all costs are non-negative.

$U := \{o\}$;
$\lambda(o) := 0$;
$\forall v \neq o, \quad \lambda(v) = +\infty$;

while $U \neq \emptyset$ **do**
  choose $u \in \arg\min_{u' \in U} \lambda(u')$;
  for v *successor of* u **do**
    if $\lambda(v) > \lambda(u) + \ell(u, v)$ **then**
      $\lambda(v) := \lambda(u) + \ell(u, v)$;
      $U := U \cup \{v\}$;
  $U := U \setminus \{u\}$;

Algorithm 1: Dijkstra algorithm

A video explanation

https://www.youtube.com/watch?v=zXfDYaahsNA

Application example

s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞

Application example

s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞
0	(3)	∞	∞	(3)	∞	(5)	∞

Application example

s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞
0	(3)	∞	∞	(3)	∞	(5)	∞
0	3	(5)	∞	(3)	∞	(5)	∞

Application example

s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞
0	(3)	∞	∞	(3)	∞	(5)	∞
0	3	(5)	∞	(3)	∞	(5)	∞
0	3	(4)	∞	3	∞	(5)	∞

Application example

s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞
0	(3)	∞	∞	(3)	∞	(5)	∞
0	3	(5)	∞	(3)	∞	(5)	∞
0	3	(4)	∞	3	∞	(5)	∞
0	3	4	(5)	3	∞	(5)	∞

Application example

s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞
0	(3)	∞	∞	(3)	∞	(5)	∞
0	3	(5)	∞	(3)	∞	(5)	∞
0	3	(4)	∞	3	∞	(5)	∞
0	3	4	(5)	3	∞	(5)	∞
0	3	4	5	3	(8)	(5)	∞

Application example

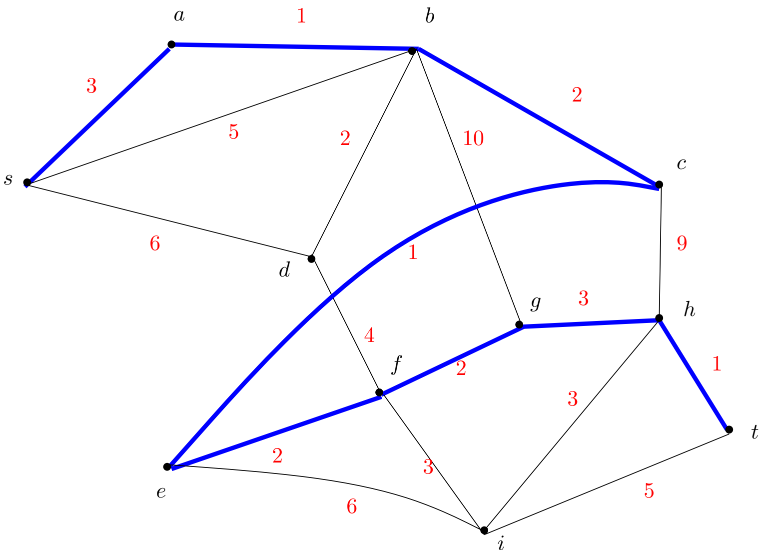
s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞
0	(3)	∞	∞	(3)	∞	(5)	∞
0	3	(5)	∞	(3)	∞	(5)	∞
0	3	(4)	∞	3	∞	(5)	∞
0	3	4	(5)	3	∞	(5)	∞
0	3	4	5	3	(8)	(5)	∞
0	3	4	5	3	(7)	5	(12)

Application example

s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞
0	(3)	∞	∞	(3)	∞	(5)	∞
0	3	(5)	∞	(3)	∞	(5)	∞
0	3	(4)	∞	3	∞	(5)	∞
0	3	4	(5)	3	∞	(5)	∞
0	3	4	5	3	(8)	(5)	∞
0	3	4	5	3	(7)	5	(12)
0	3	4	5	3	7	5	(9)

Application example

s	a	b	c	d	e	f	t
(0)	∞	∞	∞	∞	∞	∞	∞
0	(3)	∞	∞	(3)	∞	(5)	∞
0	3	(5)	∞	(3)	∞	(5)	∞
0	3	(4)	∞	3	∞	(5)	∞
0	3	4	(5)	3	∞	(5)	∞
0	3	4	5	3	(8)	(5)	∞
0	3	4	5	3	(7)	5	(12)
0	3	4	5	3	7	5	(9)
0	3	4	5	3	7	5	9


a
1
b
3
5
2
10
2
c
s
6
1
9
d
g
3
h
4
f
2
1
3
t
2
3
e
6
5
i

Shortest path complexity with positive cost

Theorem

Let $G = (V, E)$ be a directed graph, $o \in V$ and a cost function $\ell : E \to \mathbb{R}_+$.
When applying Dijkstra's algorithm, each node is visited at most once. Once a node v has been visited it's label is constant across iterations and equal to the cost of shortest o-v-path.
In particular, a shortest path from o to any vertex v can be found in $O(n^2)$, where $n = |V|$.

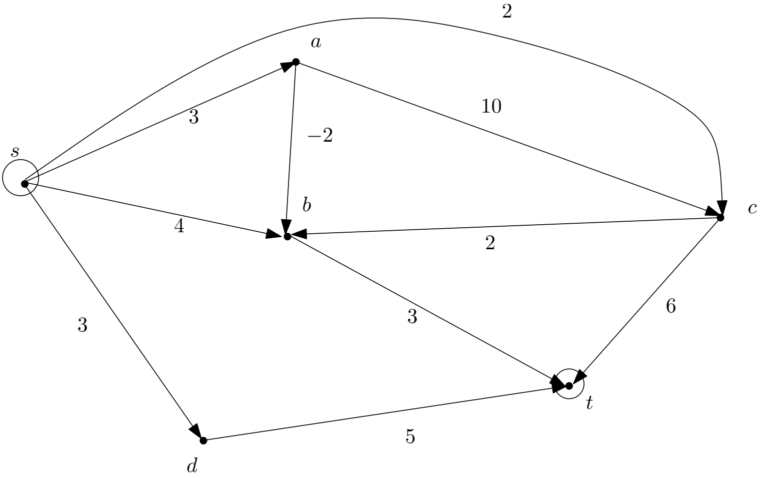
Note that with specific implementation (e.g. in binary trees of nodes) we can obtain a complexity in $O(n + m \log(\log(m)))$.

Contents

Recall on DFS

Deep First Search is an algorithm to visit every node on a graph. It consists in going as deep as possible (taking any children of a given node) and backtracking when you reach a leaf.
https://www.youtube.com/watch?v=fI6X6IBkzcw

Acircuitic graph



Topological Ordering

Definition

A topological ordering of a graph is an ordering (injective function from V to $\mathbb{N}$) of the vertices such that the starting endpoint of every edge occurs earlier in the ordering than the ending endpoint of the edge.

Applications :

- courses prerequisite
- compilation order
- manufacturing
- ...

Topological order is equivalent to acircuitic.

Theorem

A directed graph is acyclic if and only if there exist a topological ordering. A topological ordering can be found in $O(|V| + |E|)$.

Proof :

- If G has a topological ordering then it is acyclic. (by contradiction).
- If G is a DAG, then it has a root node (with no incoming edges). (by contradiction).
- If G is a DAG then G has a topological ordering (by induction).
- Done in $O(|V| + |E|)$ (maintain $count(v)$: number of incoming edges, S: set of remaining nodes with no incoming edges).

video explanation

https://www.youtube.com/watch?v=gyddxytyAiE (They use DFS to count the in-degree, it is simply a fancy way of looping on arcs)

Contents

Bellman's idea

A part of an optimal path is still optimal.

$\lambda(v) :=$ minimum cost of o-v-path, with $\lambda(v) := \infty$ if such a path doesn't exist.

Bellman's equation

$$\lambda(v) = \min_{(u,v)\in E} \big(\lambda(u) + \ell(u,v)\big)$$

There exist a predecessor u of v such that the shortest path between o and v is given by the shortest path between o and u adding the edge (u,v).

Bellman's idea

A part of an optimal path is still optimal.

$\lambda(v) :=$ minimum cost of o-v-path, with $\lambda(v) := \infty$ if such a path doesn't exist.

Bellman's equation

$$\lambda(v) = \min_{(u,v)\in E} \big(\lambda(u) + \ell(u,v)\big)$$

There exist a predecessor u of v such that the shortest path between o and v is given by the shortest path between o and u adding the edge (u,v).

Bellman's idea

A part of an optimal path is still optimal.

$\lambda(v) :=$ minimum cost of o-v-path, with $\lambda(v) := \infty$ if such a path doesn't exist.

Bellman's equation

$$\lambda(v) = \min_{(u,v)\in E} \big(\lambda(u) + \ell(u,v)\big)$$

There exist a predecessor u of v such that the shortest path between o and v is given by the shortest path between o and u adding the edge (u,v).

Dynamic Programming algorithm

Assume that the graph is connected and without cycle.

Data: Graph, cost function
$\lambda(s) := 0$;
$\forall v \neq s, \quad \lambda(v) = +\infty$;

while $\exists v \in V, \quad \lambda(v) = \infty$ **do**
  choose a vertex v such that all predecessors u have a finite label ;
  $\lambda(v) := \min\{\lambda(u) + \ell(u,v) \quad | \quad (u,v) \in E\}$;

Algorithm 2: Bellman Forward algorithm

The while loop can be replaced by a for loop over the nodes in a topological order.

Algorithm

Theorem

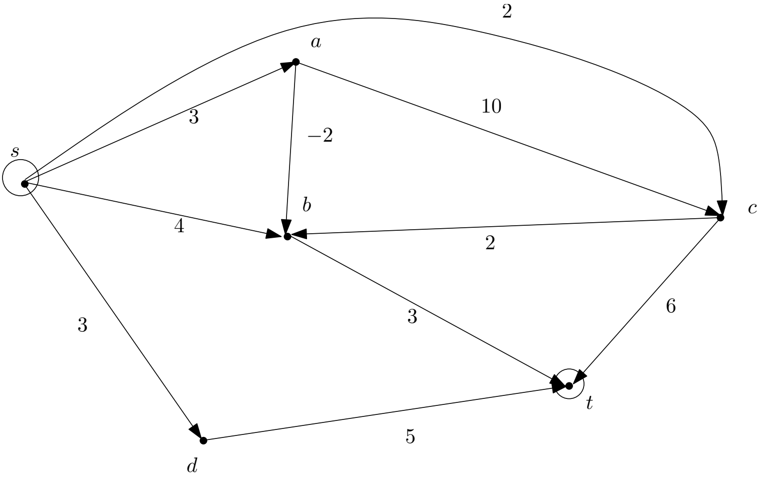
Let $D = (V, E)$ be a directed graph without cycle, and $w : E \to \mathbb{R}$ a cost function. The shortest path from o to any vertex $v \in V$ can be computed in $O(n + m)$.

Note that we do not require the costs to be positive for the Bellman algorithm. In particular, we can also compute the longest path.

Video explanation

https://www.youtube.com/watch?v=TXkDpqjDMHA (up to 6:30)

Acircuitic graph



Application example

s	a	c	b	d	t
0	∞	∞	∞	∞	∞

Application example

s	a	c	b	d	t
0	∞	∞	∞	∞	∞
0	$0+3$	∞	∞	∞	∞

Application example

s	a	c	b	d	t
0	∞	∞	∞	∞	∞
0	$0+3$	∞	∞	∞	∞
0	3	$\min\{0+2, 10+3\}$	∞	∞	∞

Application example

s	a	c	b	d	t
0	∞	∞	∞	∞	∞
0	$0+3$	∞	∞	∞	∞
0	3	$\min\{0+2, 10+3\}$	∞	∞	∞
0	3	2	$\min\{0+4, 3-2, 2+2\}$	∞	∞

Application example

s	a	c	b	d	t
0	∞	∞	∞	∞	∞
0	$0+3$	∞	∞	∞	∞
0	3	$\min\{0+2, 10+3\}$	∞	∞	∞
0	3	2	$\min\{0+4, 3-2, 2+2\}$	∞	∞
0	3	2	1	$0+3$	∞

Application example

s	a	c	b	d	t
0	∞	∞	∞	∞	∞
0	$0+3$	∞	∞	∞	∞
0	3	$\min\{0+2, 10+3\}$	∞	∞	∞
0	3	2	$\min\{0+4, 3-2, 2+2\}$	∞	∞
0	3	2	1	$0+3$	∞
0	3	2	1	3	4

Contents

Algorithm Principle

- To reach destination d from origin o in a weighted directed graph we keep a label function $\lambda(n)$.
- The label function is defined as a sum $\lambda = g + h$, where
  - $g(n)$ is the best cost of a o-n-path
  - $h(n)$ is a (user-given) heuristic of the cost of a n-d-path

$U := \{s\}$; $\lambda(s) := h(s)$; $\forall v \neq s, \quad g(v) = +\infty$;
while $U \neq \emptyset$ **do**
  choose $u \in \arg\min_{u' \in U} g(u') + h(u')$;
  for v *successor of* u **do**
    if $g(v) > g(u) + \ell(u, v)$ **then**
      $g(v) := g(u) + \ell(u, v)$;
      $U := U \cup \{v\}$;
  $U := U \setminus \{u\}$;

Algorithm 3: $A^\star$ algorithm

Heuristic definitions

Definition (admissible heuristic)

A heuristic is admissible if it underestimates the actual cost to get to the destination, i.e. if for all vertex $v \in V$, $h(v)$ is lower or equal to the cost of a shortest path from v to d.

Example: in the case of a graph in $\mathbb{R}^2$ with a cost proportional to the euclidean distance, an admissible heuristic is the euclidean distance between v and t (the "direct flight" distance).

Definition (consistent heuristic)

The heuristic h is consistent if it is admissible and for every $(u, v) \in E$, $h(u) \leq \ell(u, v) + h(v)$.

A consistent heuristic satisfies a "triangle inequality".

Consistent heuristic

- $h \equiv 0$ is consistent. In this case, $A^\star$ reduced to Dijkstra.
- If h is consistent, $A^\star$ can be implemented more efficiently.
- Roughly speaking, no node needs to be processed more than once, and $A^\star$ is equivalent to running Dijkstra's algorithm with the reduced cost $\tilde{\ell}(u, v) = \ell(u, v) + h(v) - h(u)$.

Choice of heuristic

- If $h \equiv 0$, we have Dijkstra algorithm.
- If h is admissible, $A^\star$ yields the shortest path.
- If h is consistent we have Dijkstra's algorithm with the reduced cost $\tilde{\ell}(u, v) = \ell(u, v) + h(v) - h(u)$.
- If h is exact we explore only the best path.
- If h is not admissible the algorithm might not yield the shortest path, but can be fast to find a good path.

Video explanation

Detailed explanation of A* :
https://www.youtube.com/watch?v=eSOJ3ARN5FM
Some comparison of the algorithm :
https://www.youtube.com/watch?v=GC-nBgi9r0U
A quick run of A* :
https://www.youtube.com/watch?v=19h1g22hby8