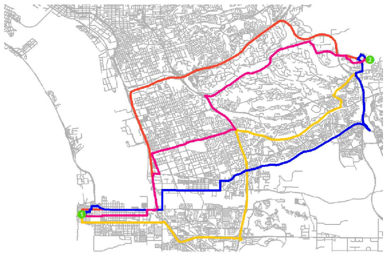
Shortest Path 1



Mauro Dell'Amico
DISMI, Universitá di Modena e Reggio Emilia
mauro.dellamico{at}unimore.it, www.or.unimore.it

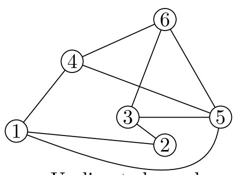
Definitions

A **graph** $G = (V, E)$ is given by a pair of sets:

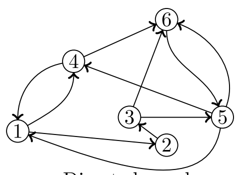
$V = \{v_1, v_2, \ldots, v_n\}$: set of vertices
$E = \{e_1, e_2, \ldots, e_m\}$: set of edges
(pair of vertices in V: $e_i = (v_i^{from}, v_i^{to})$)

- If pair of vertices defining an edge are *undirected* the graph is *not oriented* (or simply "graph")
- If the pair of vertices is *ordered* it is called **arc** and the graph is *directed*. In this case we use A for **arc set**

A **cost** may be associated to each edge/arc ($c_e, e \in E$, or $c_{ij}, (i, j) \in A$)



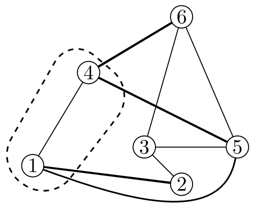
Undirected graph



Directed graph

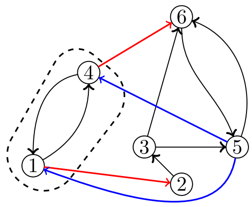
Definitions: cut

- Given $S \subset V$ a **cut** $\delta(S)$ is the subset of edges connecting S with $V \setminus S$
- For directed graphs we have to distinguish between arcs exiting from S and entering in S:
  $\delta^+(S) = \{(i,j), i \in S, j \in V \setminus S\}$ and
  $\delta^-(S) = \{(i,j), i \in V \setminus S, jS\}$



Undirected graph

$\delta(S) = \{(1,2),(1,5),(4,5),(4,6)\}$



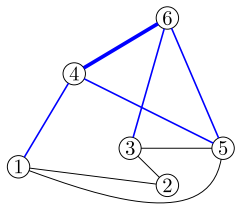
Directed graph

$\delta^+(S) = \{(1,2),(4,6)\}$

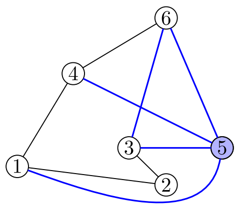
$\delta^-(S) = \{(5,1),(5,4)\}$

Definitions: paths

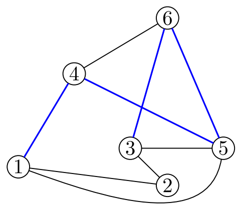
- A **path** is a sequences of vertices, pairwise connected by and edge/arc
- A path is **simple** if doesn't use two times the same edge/arc
- A path is **elementary** if it doesn't use two times the same vertex



path=(1,4,6,5,4,6,3)



simple path=(4,5,6,3,5,1)



elementary path=(1,4,5,6,3)

- A **cycle** is a path that starts and ends in the same vertex

Shortest path

Given a directed graph $G = (V, A)$ with cost c_{ij} associated to each arc $(i, j) \in A$, the **Shortest Path Problem (SPP)** asks for the minimum cost path from a given vertex $s \in V$ to a given vertex $t \in V$ (or to all vertices)

- If $c_{ij} \geq 0$, $\forall (i, j) \in A$, the path is elementary, and SPP can be solved in polynomial time
- If some c_{ij} is negative, then G may contain negative cost circuits
  - in this case the circuit is used infinite times to minimize the cost
  - it is necessary to **add a constraint** forbidding to visit two times a vertex
  - SPP becomes NP-complete

Mathematical model

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ is in the path;} \\ 0 & \text{otherwise.} \end{cases}$$

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\underbrace{\sum_{(h,j)\in\delta^+(h)} x_{hj}}_{\text{arcs from } h} - \underbrace{\sum_{(i,h)\in\delta^-(h)} x_{ih}}_{\text{arcs to } h} = \begin{cases} 1 & \text{se } h = s \\ -1 & \text{se } h = t \\ 0 & \forall h \in V \setminus \{s,t\} \end{cases}$$

$$\underbrace{\sum_{(i,j)\in A: i,j\in S} x_{ij}}_{\text{arcs in } S} \leq |S| - 1, \quad \forall S \subseteq V, S \neq \emptyset \qquad (*)$$

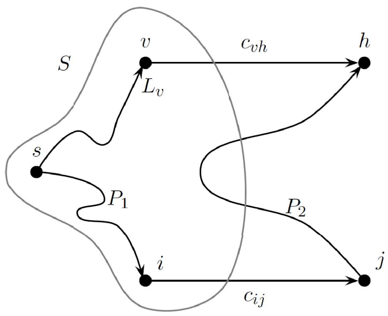
$$x_{ij} \in \{0,1\}, \forall (i,j) \in A$$

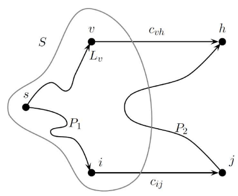
The $2^n - 1$ constraints $(*)$ are the *subtour elimination*.

Dijkstra algorithm

- This algorithm finds the shortest path on graphs with **no negative costs**
- It iteratively extends a set $S \subseteq V$, starting with $S = \{s\}$

Theorem. Let L_i be the cost of the shortest path from s to i, for all $i \in S \subset V$, $S \ni s$. Let $(v, h) = argmin\{L_i + c_{ij} : (i, j) \in \delta^+(S)\}$. Then, $L_v + c_{vh}$ is the cost of the shortest path from s to h.





Proof. The theorem states that the shortest path from s to h reaches $v \in S$ using only vertices in S, and than goes directly to h. Suppose, by contradiction, that the shortest path to h is a different path P. Let $(i, j) \in P \cap \delta^+(S)$ be the first (possibly unique) arc of P exiting form S. Let $P = P_1 \cup \{(i, j)\} \cup P_2$. we have

$$C(P) = \underbrace{c(P_1)}_{\geq L_i} + c_{ij} + \underbrace{C(P_2)}_{\geq 0} \geq L_i + c_{ij} \geq L_v + c_{vh}.$$

Hence $L_v + c_{vh}$ is the minimu cost of any path from s to h.

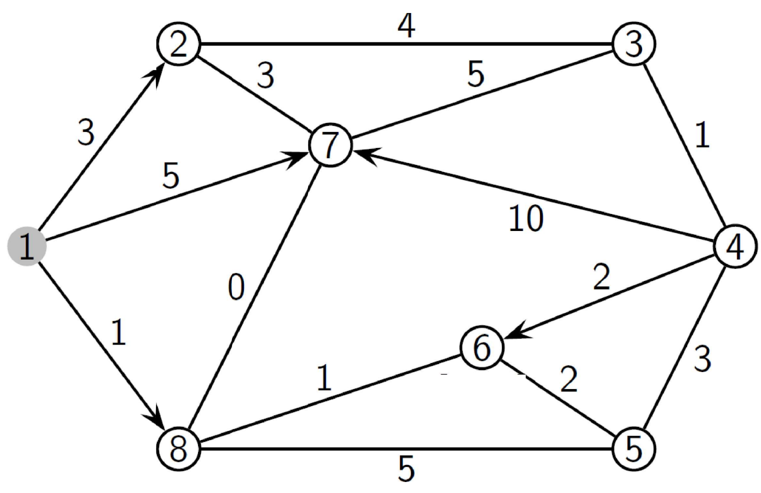
The algorithm

The theorem suggests an iterative algorithm:

Dijkstra (1 **version)**
$S = \{s\}$; $L[s] = 0$; $pred[s] = s$;
while $(|S| \neq n)$ **do**
$(v, h) = argmin\{L[i] + c_{ij} : (i, j) \in \delta^+(S)\}$;
$L[h] = L[v] + c_{vh}$;
$pred[h] = v$;
$S = S \cup \{h\}$;
endwhile

The **while** loop is execute $O(|V|)$ times. The search for the min $L[i] + c_{ij}$ requires $O(|A|)$. The time complexity is $O(|V||A|)$.

Example



Example

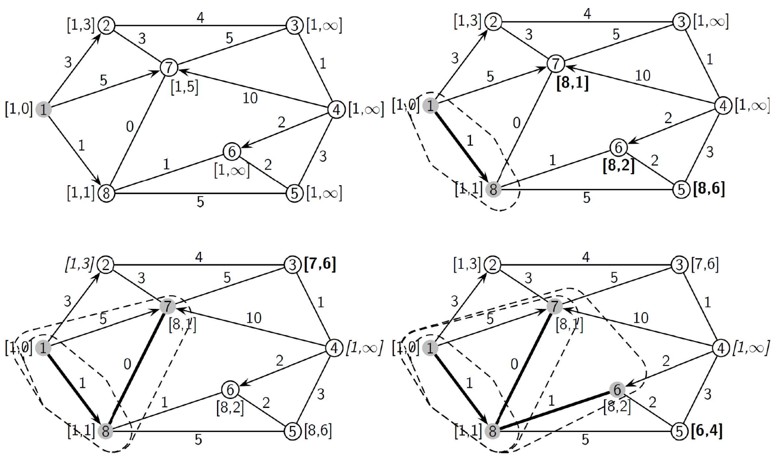
We can reduce the time complexity to $O(|V|^2)$ if we store the information computed at each iteration

$$L[j] = \begin{cases} \text{cost of the shortest path from } s \text{ to } j, & \text{if } j \in S; \\ \min\{L[i] + c_{ij} : i \in S\}, & \text{if } j \notin S; \end{cases}$$

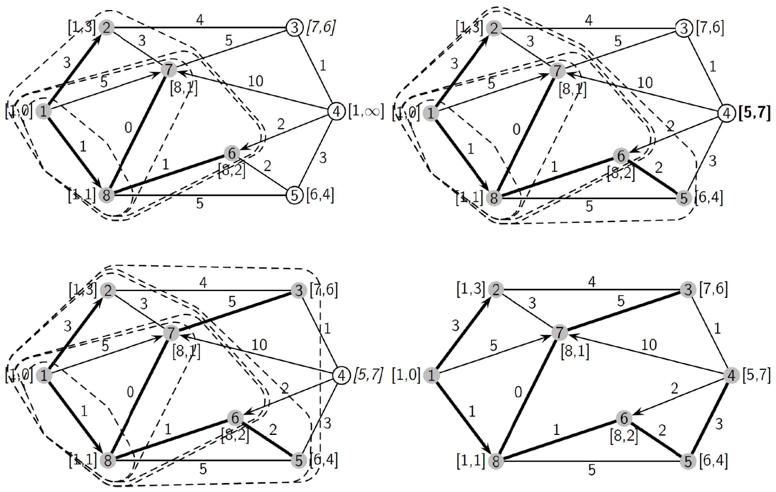
Dijkstra (II **version**)

- **for** $j \in V$ **do** $L[j] = c_{sj};\ pred[j] = s;$
- $S = \{s\};\ L[s] = 0;$
- **while** $(|S| \neq n)$ **do**
  - $h = argmin\{L[h] : h \in V \setminus S\};$
  - $S = S \cup \{h\};$
  - *//update labels*
  - **for** $j \in V \setminus S$ **do**
    - **if** $L[h] + c_{hj} < L[j]$ **then**
      - $L[j] = L[h] + c_{hj} < L[j];\ pred[j] = h;$
    - **endif**
  - **endfor**
- **endwhile**

Example



Example



Tabular version

$$
[c_{ij}] = \left[\begin{array}{c|c|c|c|c|c|c|c}
 & 3 & & & & & 5 & 1 \\ \hline
 & & 4 & & & & 3 & \\ \hline
 & 4 & & 1 & & & 5 & \\ \hline
 & & 1 & & 3 & 2 & 10 & \\ \hline
 & & & 3 & & 2 & & 5 \\ \hline
 & & & & 2 & & & 1 \\ \hline
 & 3 & 5 & & & & & 0 \\ \hline
 & & & & 5 & 1 & 0 &
\end{array}\right]
$$

S	$L[j]$							$pred[j]$
	2	3	4	5	6	7	8	2 3 4 5 6 7 8
$\{1\}$	3	∞	∞	∞	∞	5	$\boxed{1}$	1 1 1 1 1 1 1
$\{1, 8\}$	3	∞	∞	6	2	$\boxed{1}$	1	1 1 1 8 8 8 1
$\{1, 8, 7\}$	3	6	∞	6	$\boxed{2}$	1	1	1 7 1 8 8 8 1
$\{1, 8, 7, 6\}$	$\boxed{3}$	6	∞	4	2	1	1	1 7 1 6 8 8 1
$\{1, 8, 7, 6, 2\}$	3	6	∞	$\boxed{4}$	2	1	1	1 7 1 6 8 8 1
$\{1, 8, 7, 6, 2, 5\}$	3	$\boxed{6}$	7	4	2	1	1	1 7 5 6 8 8 1
$\{1, 8, 7, 6, 2, 5, 3\}$	3	6	$\boxed{7}$	4	2	1	1	1 7 5 6 8 8 1
$\{1, 8, 7, 6, 2, 5, 3, 4\}$	3	6	7	4	2	1	1	1 7 5 6 8 8 1