CX1107
Data Structures and Algorithms

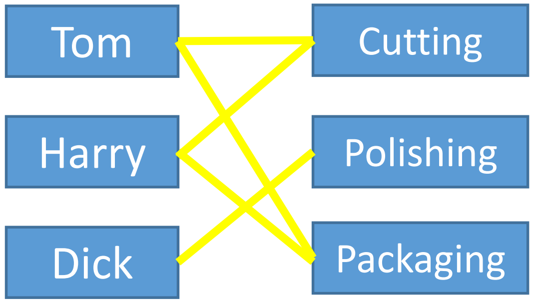
Bipartite Graph – Matching Problem

Maximum Flow Algorithm

Dr. Loke Yuan Ren
Lecturer
yrloke@ntu.edu.sg

College of Engineering
School of Computer Science and Engineering

Matching Problem

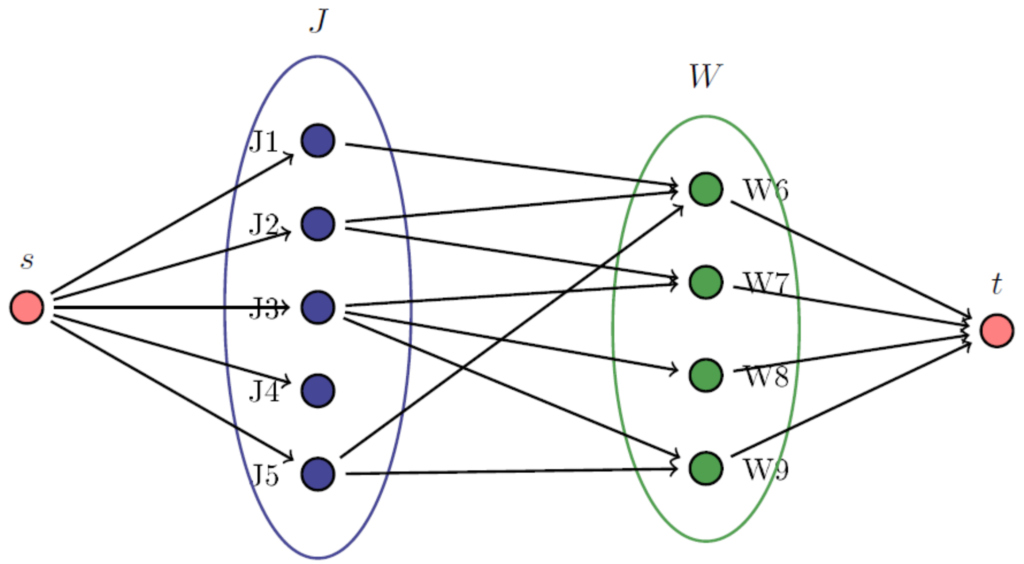


- A graph that vertex V can be partitioned into two subsets, e.g. J and W.
- J and W are two disjoint sets.
- Every edge connects a vertex in J to one in W.
- This graph is known as Bipartite Graph.

- Matching:
  - A subset of edges that are mutually non-adjacent
  - No two edges have an endpoint in common
- Problem: Matching with the maximum number of edges

Maximum Matching == Maximum Flow

- Introduce two vertices: a source *s* and a sink *t*
- A flow network G(V,E) is a directed graph in which each edge (j, w) has a nonnegative capacity.
- The capacity is {0, 1} for matching problem



The Ford-Fulkerson Method

- An iterative improvement strategy
- Proposed by L. R. Ford Jr. and D. R. Fulkerson in 1956
- Iteratively find an additional flow (match) in the network
  - A residual network is used to find the available flow
  - $c_f(j,w) = c(j,w) - f(j,w)$
- Initially, there is no flow
- Residual network == original network

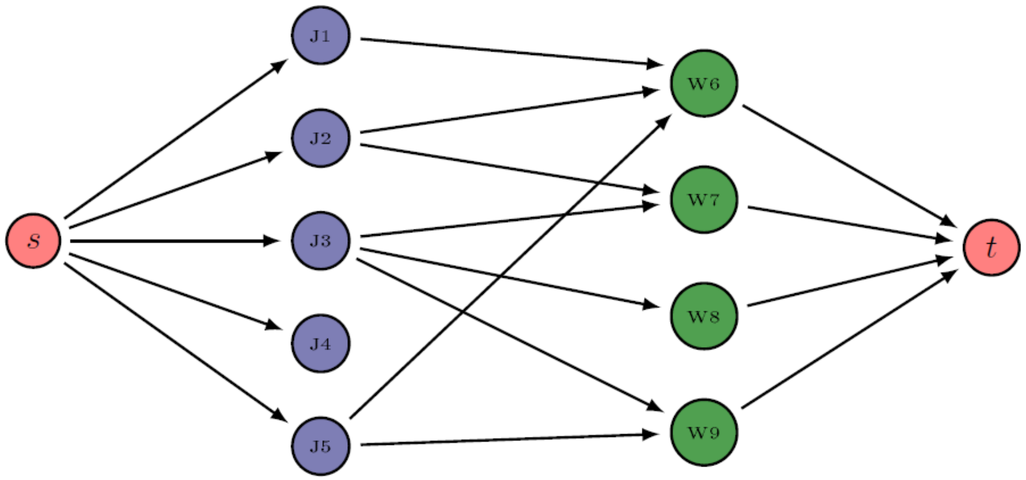
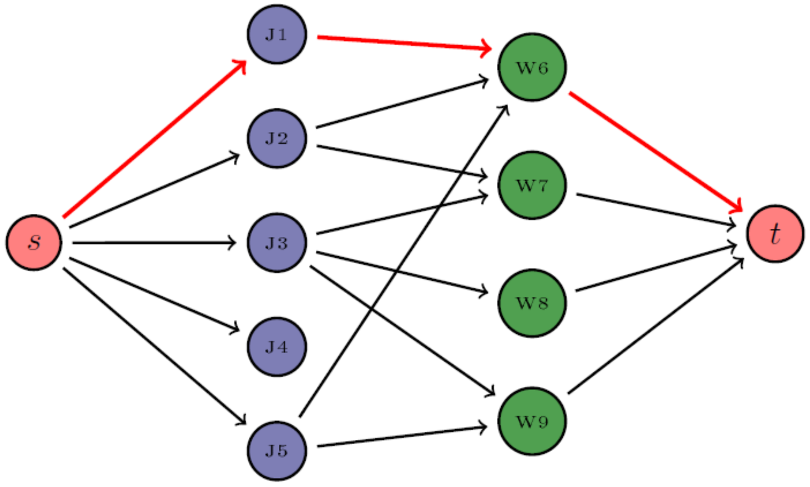


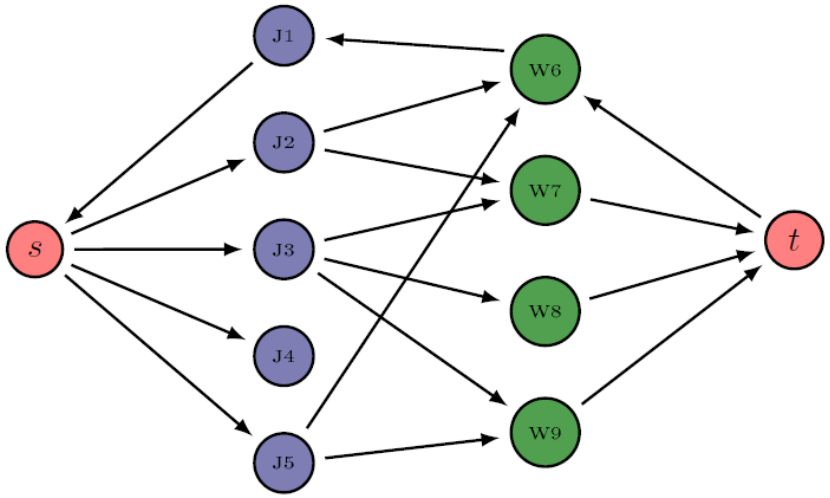
Figure 12.6: Residual Network G_f where $c_f(j,w)$ is 1 $\forall e_f$

The Ford-Fulkerson Method

- In each iteration,
  - Find an augmenting path p from s to t in the residual network G_f
  - Update the original matching network G by adding p
  - Update the residual network G_f

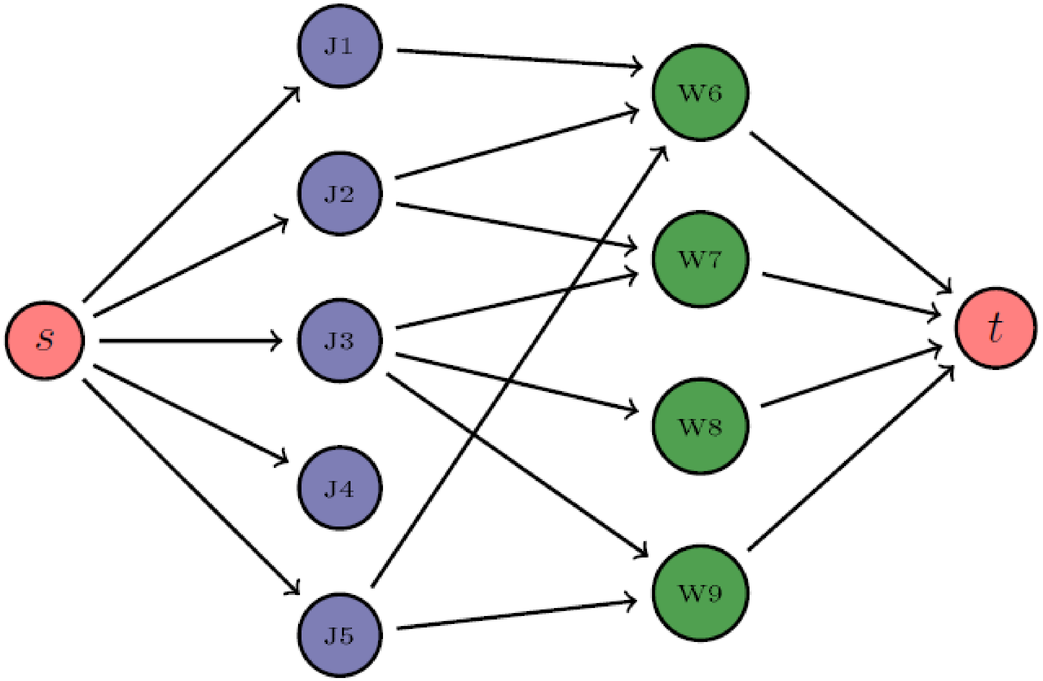


(a) Matching Network G

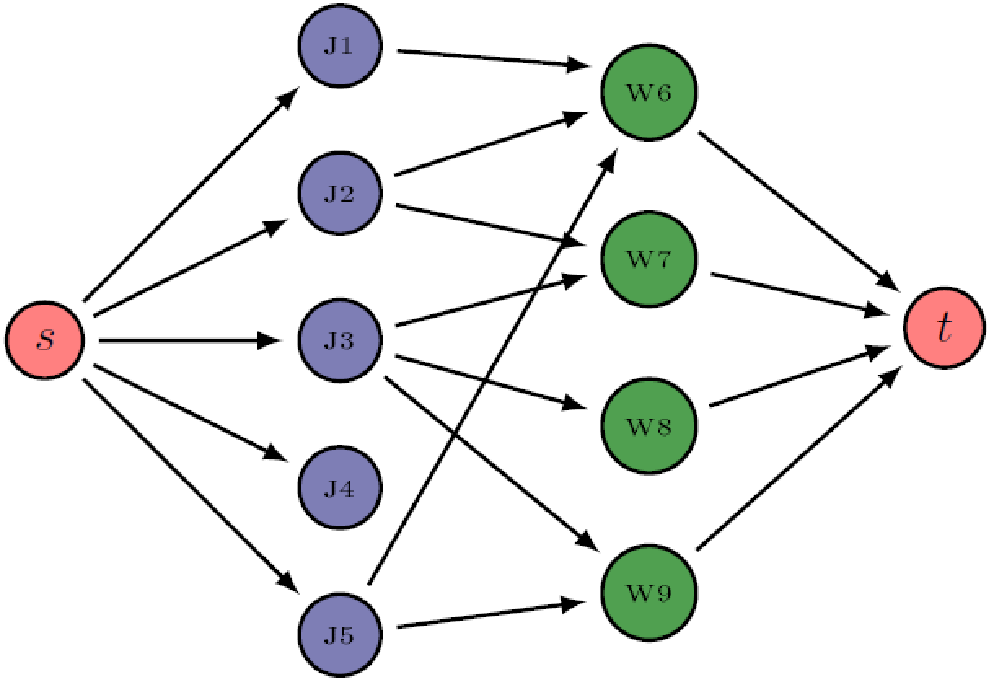


(b) Residual Network G_f

First …



Network Graph G



Residual Graph G_f

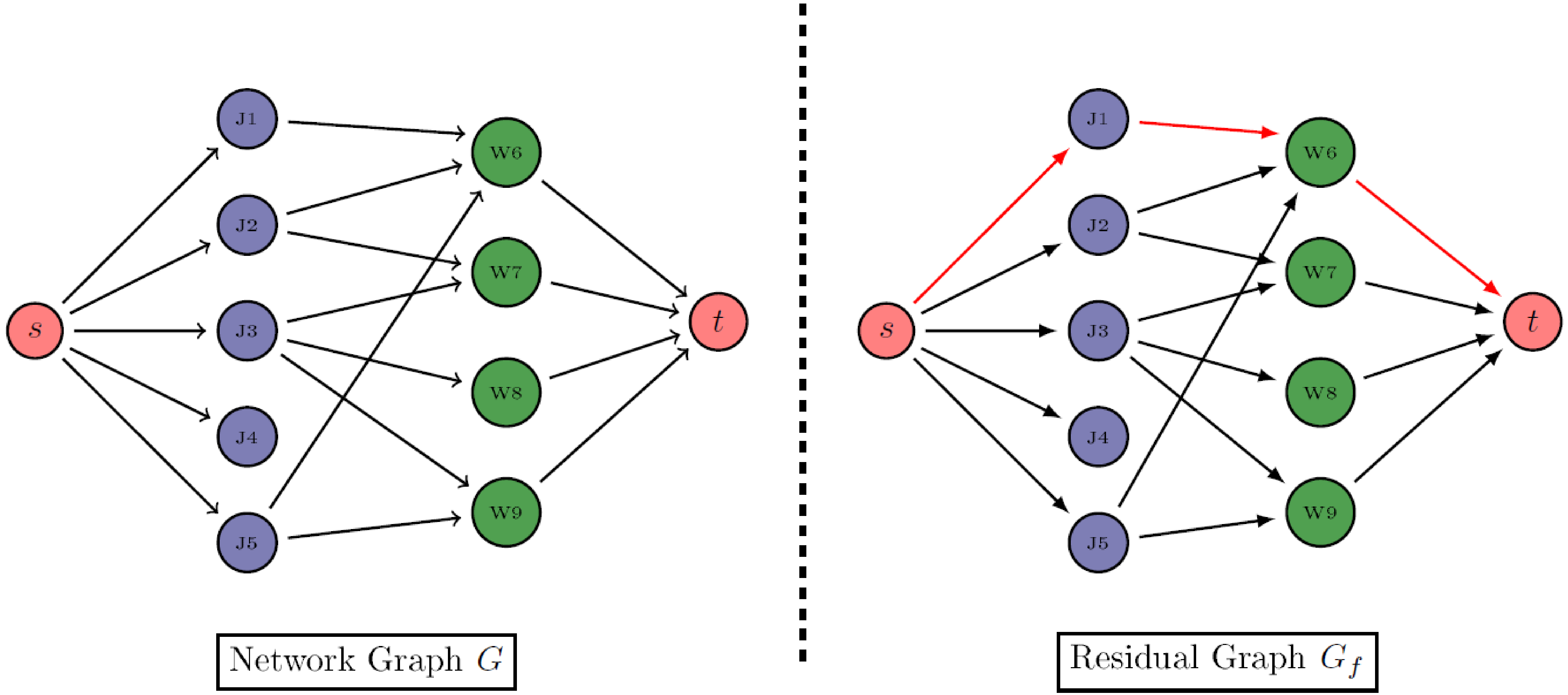
Algorithm 2 Ford-Fulkerson

$c_f(j,w) = c(j,w) - f(j,w)$

function FORD-FULKERSON(Graph G, Vertex s, Vertex t)
  for each edge $(u,v) \in E[G]$ **do** ▷ Initialization of Flows
    $f[u,v] \leftarrow 0$
    $f[v,u] \leftarrow 0$
  end for
  while Finding a path from s to t in G_f **do**
    $c_{min}(p) \leftarrow \min\{c_f(u,v) : (u,v) \in p\}$
    for each edge $(u,v) \in p$ **do**
      if $(u,v) \in E$ **then**
        $f[u,v] \leftarrow f[u,v] + c_{min}(p)$ ▷ adding the new flow
      else
        $f[v,u] \leftarrow f[v,u] - c_{min}(p)$
      end if
      $c_f(u,v) \leftarrow c_f(u,v) - c_{min}(p)$ ▷ Update Residual Graph, G_f
      $c_f(v,u) \leftarrow c_f(v,u) + c_{min}(p)$
    end for
  end while
end function

- $c_f(j,w) = c(j,w)$: same as the original network G
- $c_f(j,w)$ is non-zero

Next …



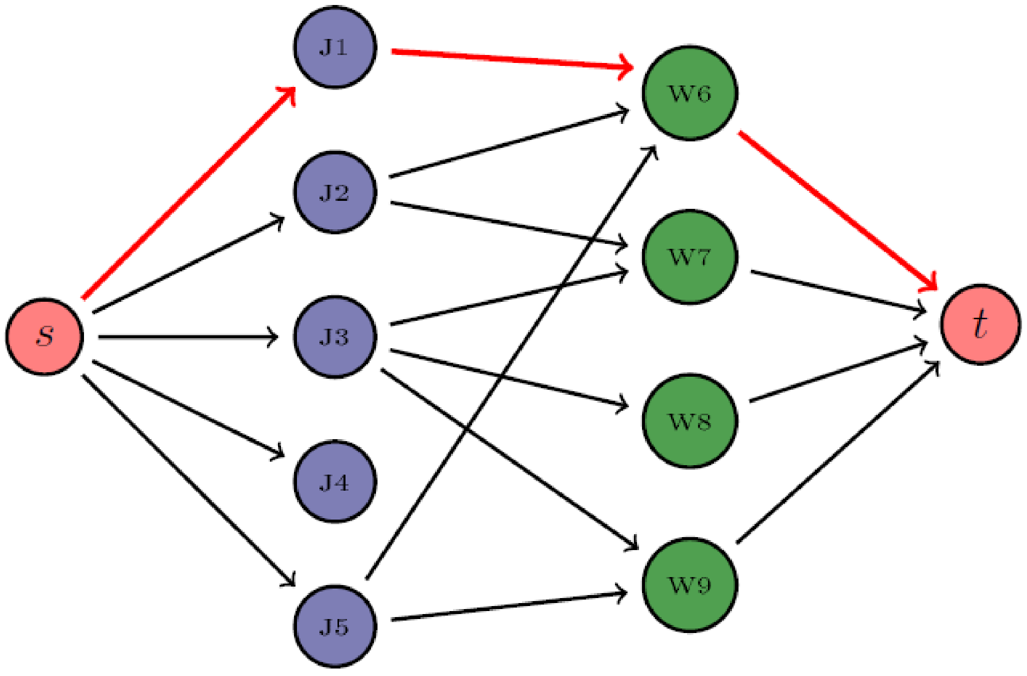
Algorithm 2 Ford-Fulkerson

$c_f(j,w) = c(j,w) - f(j,w)$

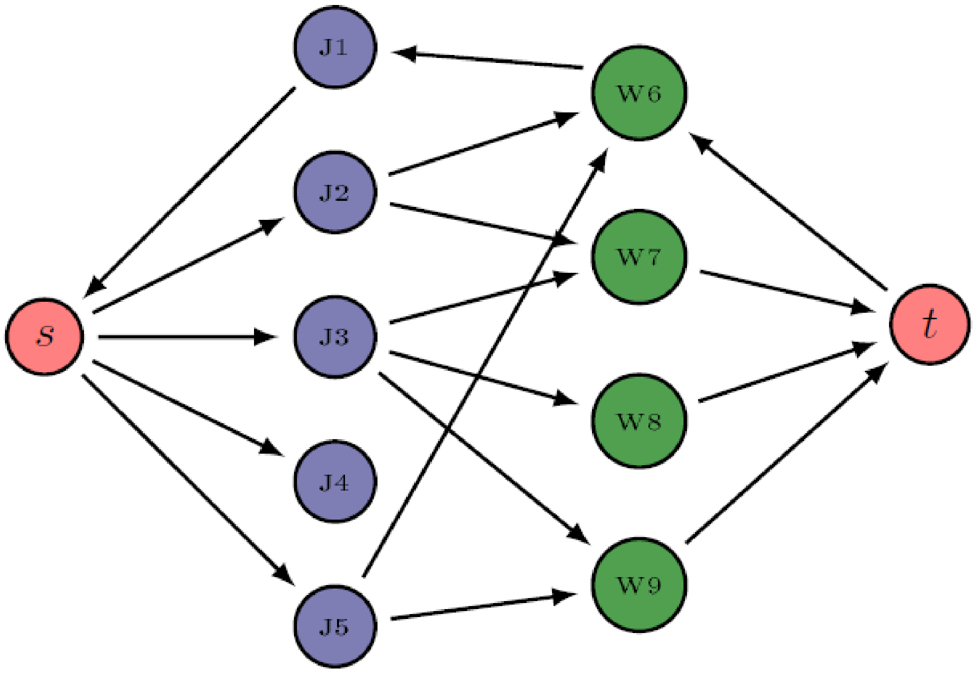
function FORD-FULKERSON(Graph G, Vertex s, Vertex t)
  for each edge $(u,v) \in E[G]$ **do** ▷ Initialization of Flows
    $f[u,v] \leftarrow 0$
    $f[v,u] \leftarrow 0$
  end for
  while Finding a path from s to t in G_f **do**
    $c_{min}(p) \leftarrow \min\{c_f(u,v) : (u,v) \in p\}$
    for each edge $(u,v) \in p$ **do**
      if $(u,v) \in E$ **then**
        $f[u,v] \leftarrow f[u,v] + c_{min}(p)$ ▷ adding the new flow
      else
        $f[v,u] \leftarrow f[v,u] - c_{min}(p)$
      end if
      $c_f(u,v) \leftarrow c_f(u,v) - c_{min}(p)$ ▷ Update Residual Graph, G_f
      $c_f(v,u) \leftarrow c_f(v,u) + c_{min}(p)$
    end for
  end while
end function

- $c_f(j,w) = c(j,w)$: same as the original network G
- $c_f(j,w)$ is non-zero
- $c_{min}(p)$ is always 1 for matching problem here

Next...

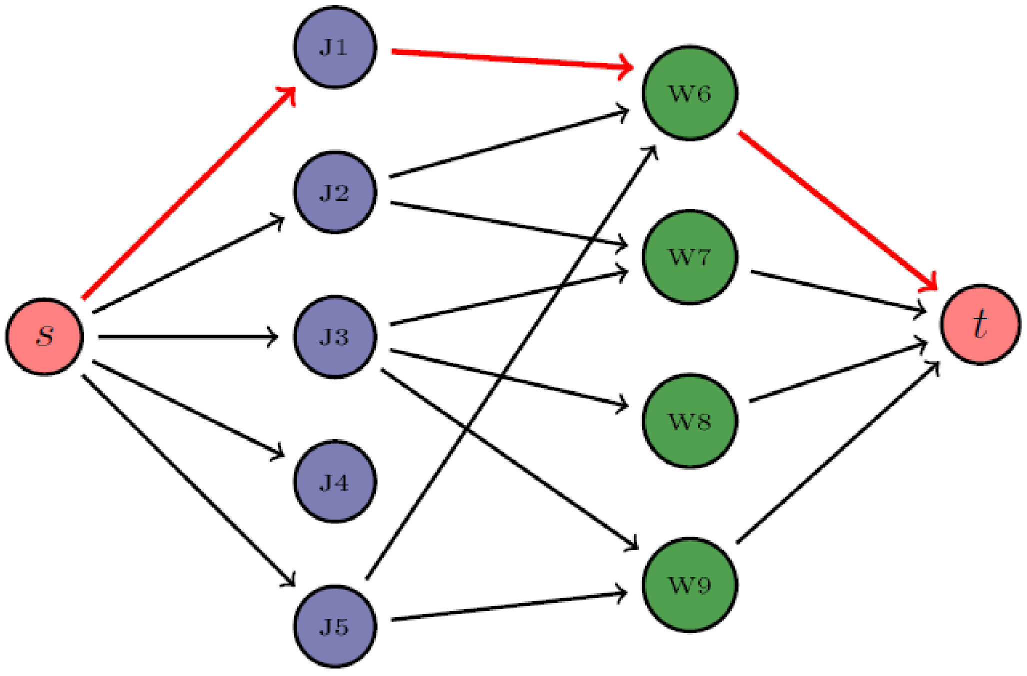


Network Graph G

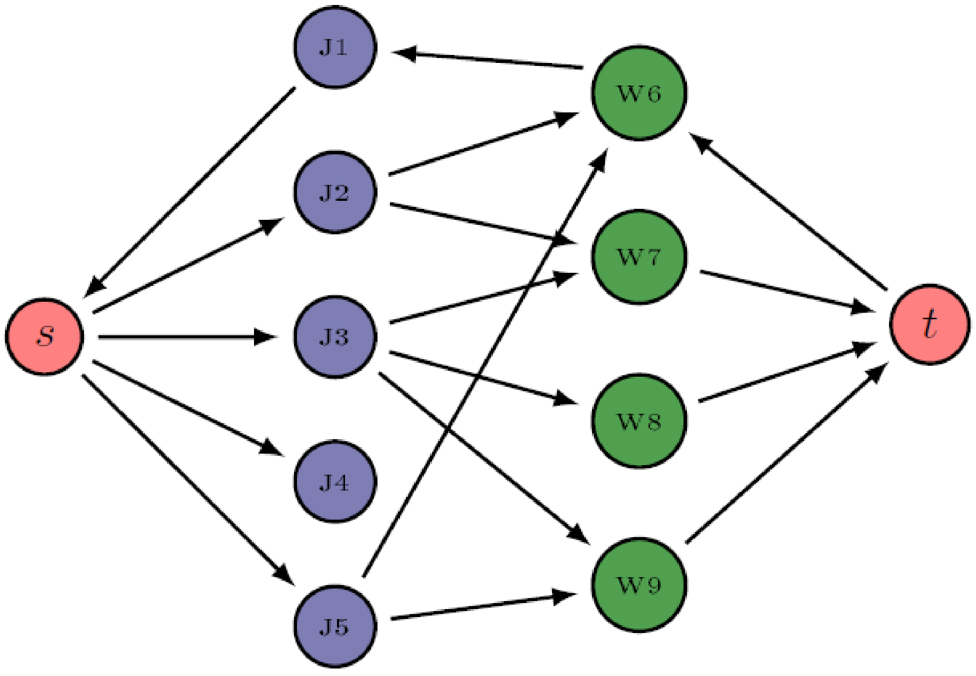


Residual Graph G_f

Next…

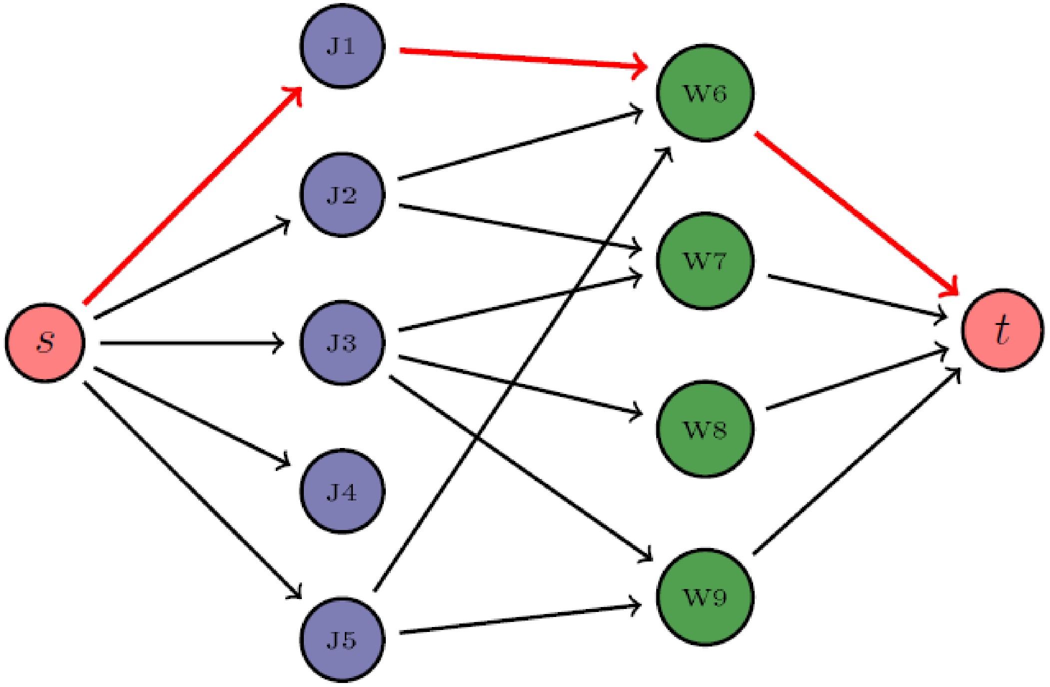


Network Graph G

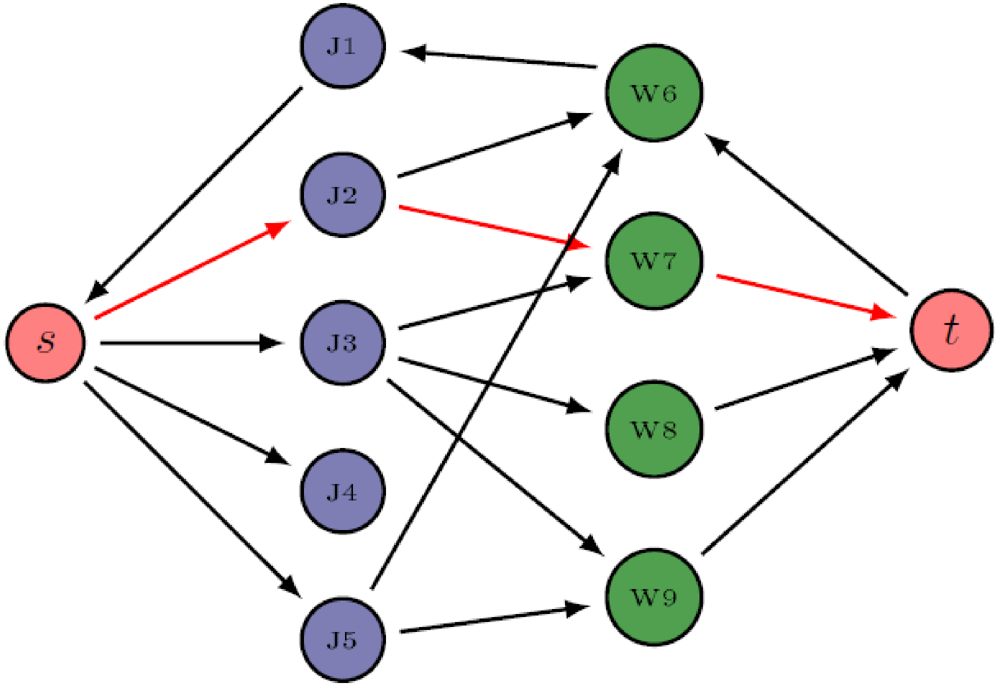


Residual Graph G_f

Next …

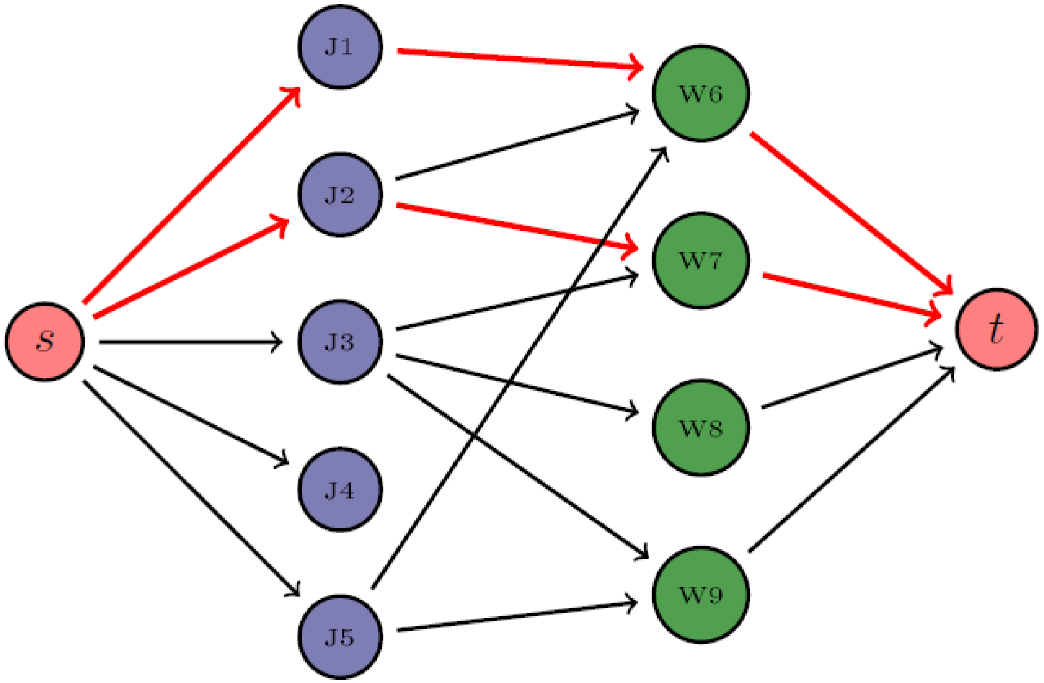


Network Graph G

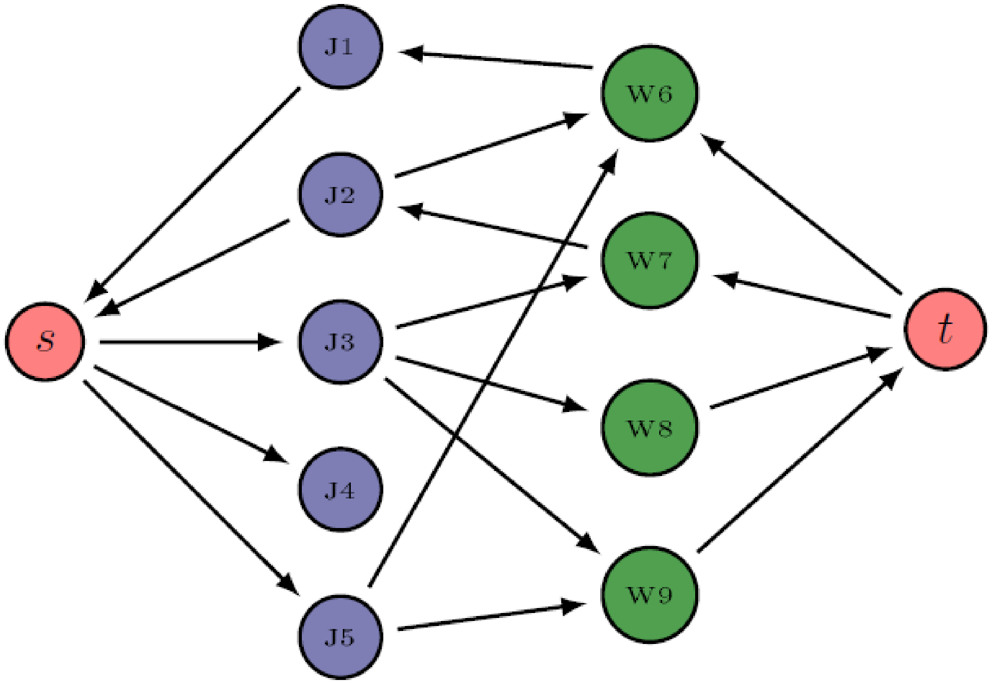


Residual Graph G_f

Next …

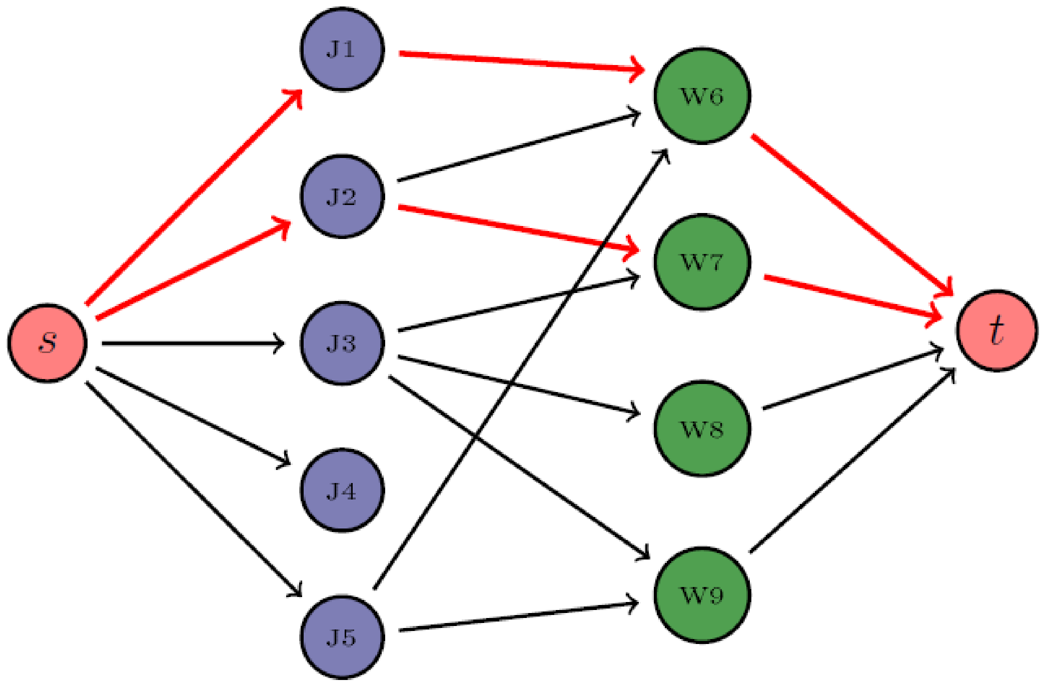


Network Graph G

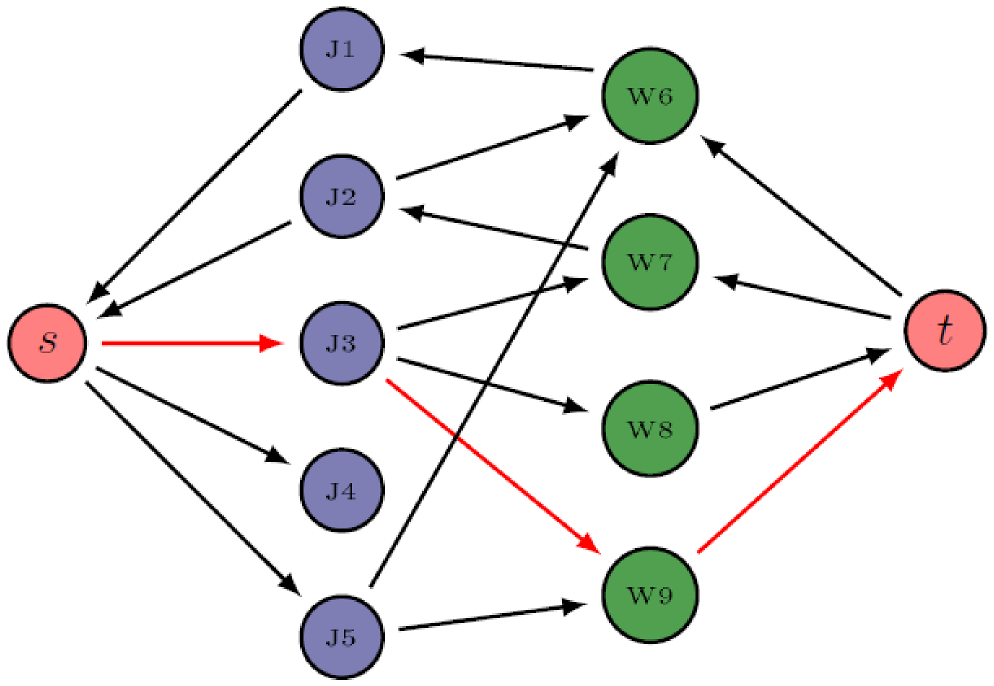


Residual Graph G_f

Next …

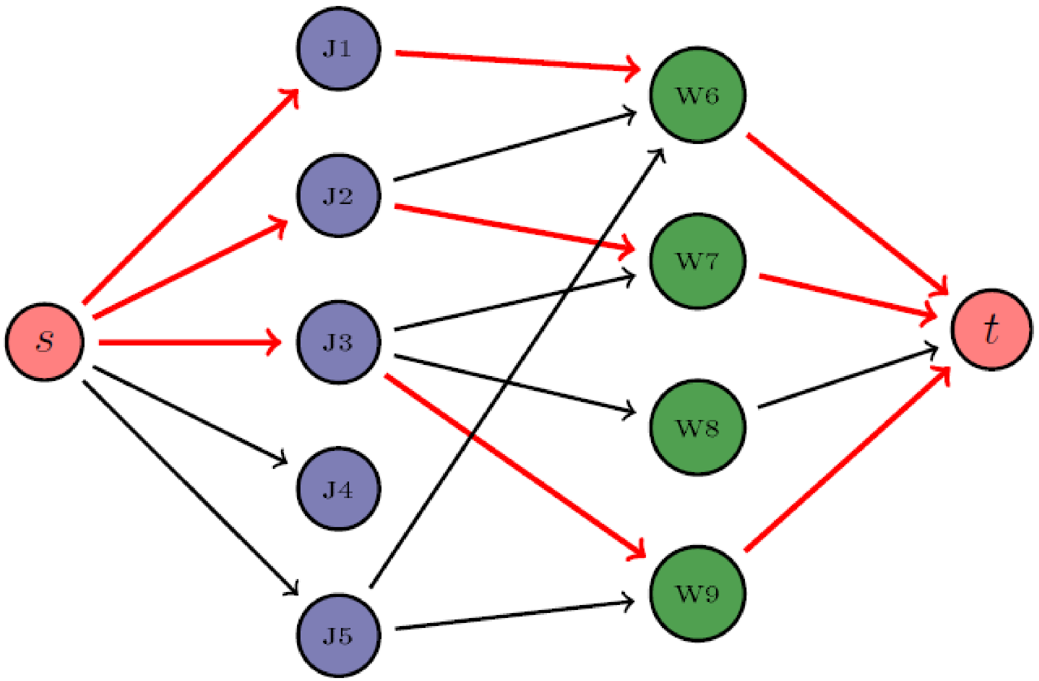


Network Graph G

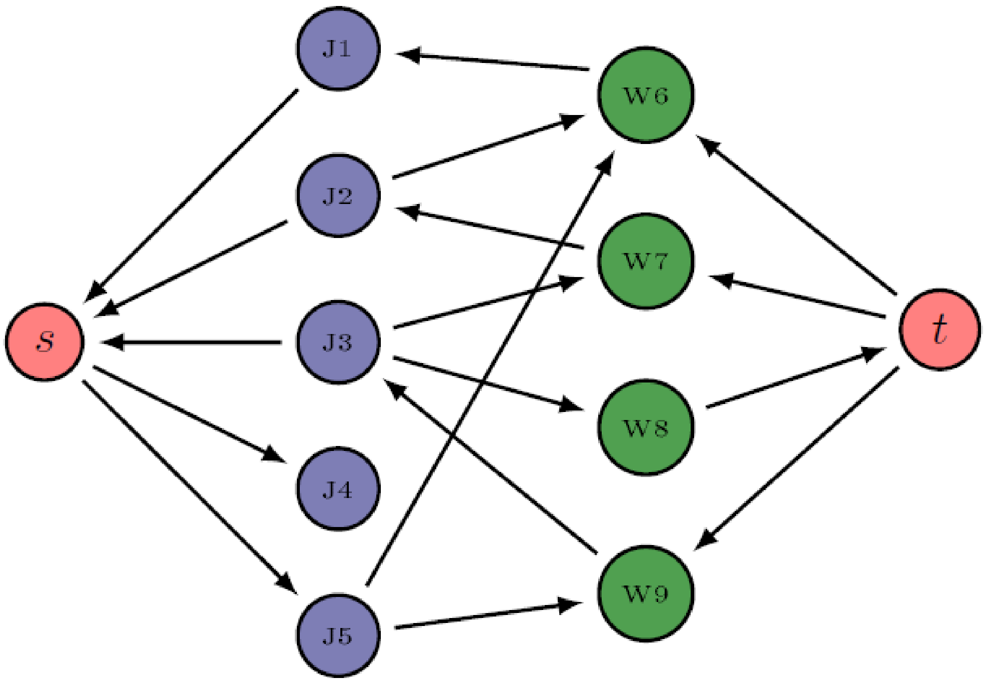


Residual Graph G_f

Next …

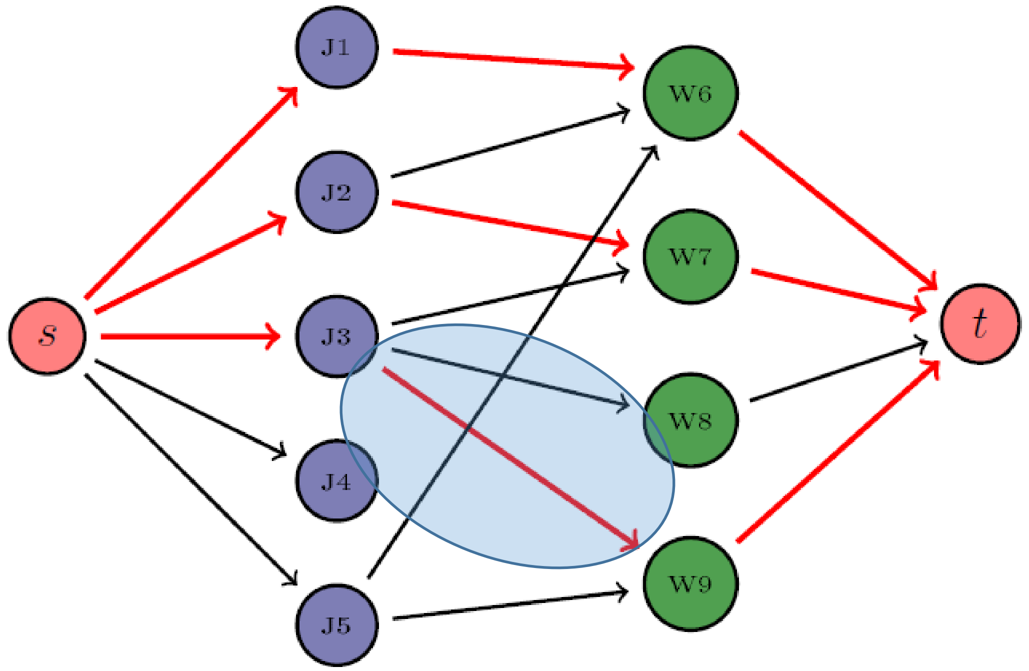


Network Graph G

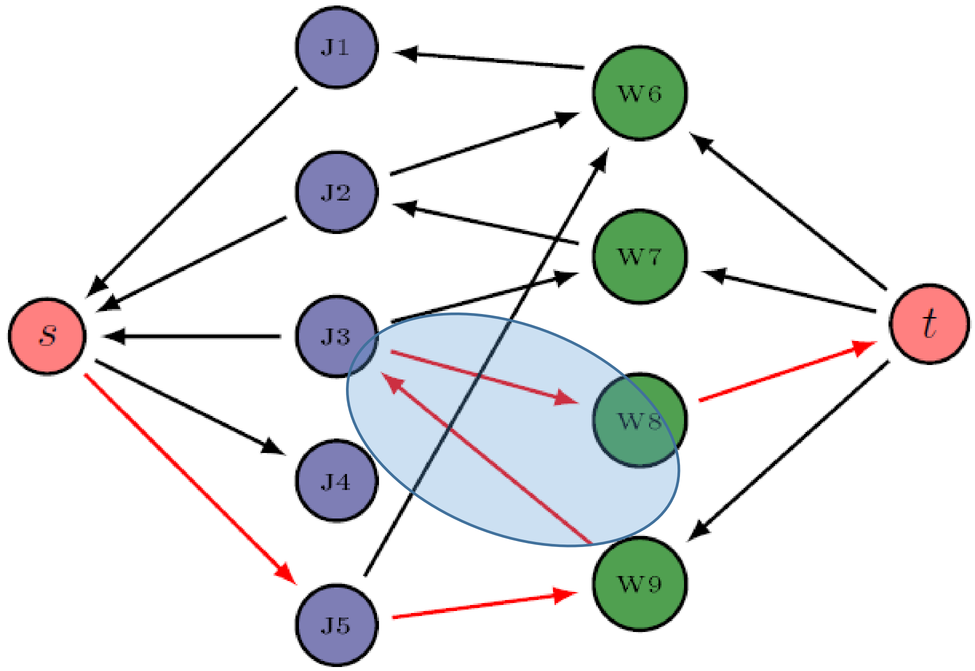


Residual Graph G_f

Next… Why does it work?



Network Graph G



Residual Graph G_f

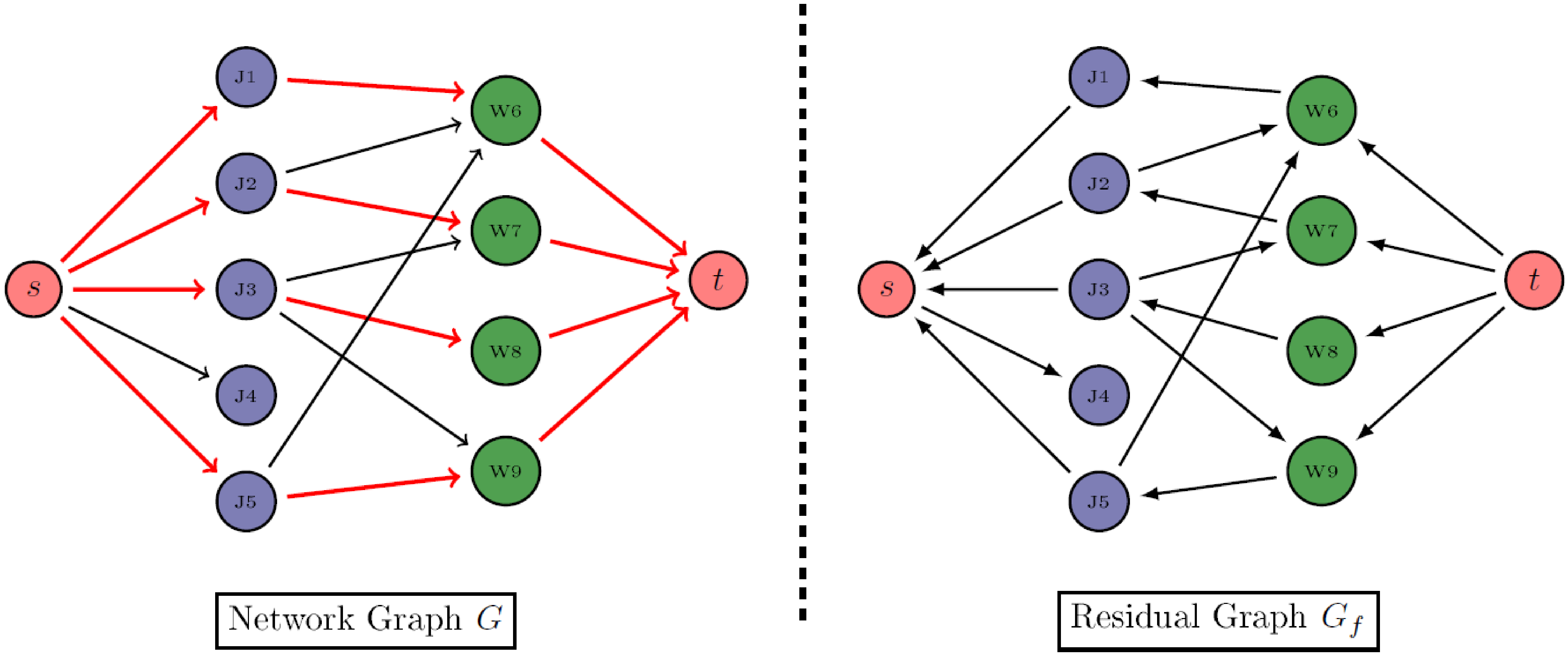
Algorithm 2 Ford-Fulkerson

$$c_f(j,w) = c(j,w) - f(j,w)$$

```
function FORD-FULKERSON(Graph G, Vertex s, Vertex t)
    for each edge (u, v) ∈ E[G] do                         ▷ Initialization of Flows
        f[u, v] ← 0
        f[v, u] ← 0
    end for
    while Finding a path from s to t in G_f do
        c_min(p) ← min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p do
            if (u, v) ∈ E then
                f[u, v] ← f[u, v] + c_min(p)               ▷ adding the new flow
            else
                f[v, u] ← f[v, u] − c_min(p)
            end if
            c_f(u, v) ← c_f(u, v) − c_min(p)               ▷ Update Residual Graph, G_f
            c_f(v, u) ← c_f(v, u) + c_min(p)
        end for
    end while
end function
```

- $c_f(j,w) = c(j,w)$: same as the original network G
- $c_f(j,w)$ is non-zero
- $c_{min}(p)$ is always 1 for matching problem here
- Remove a flow in opposite direction

Next…No new flow. Done!



Algorithm 2 Ford-Fulkerson

$c_f(j, w) = c(j, w) - f(j, w)$

```
function FORD-FULKERSON(Graph G, Vertex s, Vertex t)
    for each edge (u, v) ∈ E[G] do                      ▷ Initialization of Flows
        f[u, v] ← 0
        f[v, u] ← 0
    end for
    while Finding a path from s to t in G_f do
        c_min(p) ← min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p do
            if (u, v) ∈ E then
                f[u, v] ← f[u, v] + c_min(p)            ▷ adding the new flow
            else
                f[v, u] ← f[v, u] − c_min(p)
            end if
            c_f(u, v) ← c_f(u, v) − c_min(p)            ▷ Update Residual Graph, G_f
            c_f(v, u) ← c_f(v, u) + c_min(p)
        end for
    end while
end function
```

- $c_f(j, w) = c(j, w)$: same as the original network G
- $c_f(j, w)$ is non-zero
- Remove a flow in opposite direction

Summary

- Ford Fulkerson Method
  - Maximum Flow Problem
  - Maximum Matching Problem

- Finding the augmenting paths
  - BFS or DFS or any graph traversal

- Matching problem is a {0,1} weighted graph or an unweighted graph
  - Implementation is easier