Algorithm design and analysis

— Network Flow —

Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

Algorithm design and analysis

— Maximum Flow and Minimum Cut —

Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

Maximum Flow and Minimum Cut

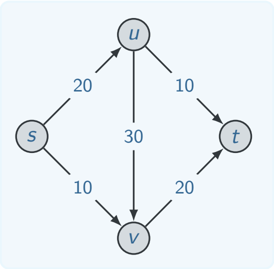
- Two rich algorithmic problems.
- Fundamental problems in combinatorial optimization.
- Beautiful mathematical duality between flows and cuts.
- Numerous non-trivial applications:
  - Bipartite matching.
  - Data mining.
  - Project selection.
  - Airline scheduling.
  - Baseball elimination.
  - Image segmentation.
  - Network connectivity.
  - Open-pit mining.
  - Network reliability.
  - Distributed computing.
  - Egalitarian stable matching.
  - Security of statistical data.
  - Network intrusion detection.
  - Multi-camera scene reconstruction.
  - Gene function prediction.

Flow Networks

- Use directed graphs to model transportation networks:
  - edges carry traffic and have capacities.
  - nodes act as switches.
  - *source* nodes generate traffic, *sink* nodes absorb traffic.

Flow Networks

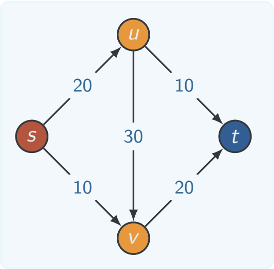
- Use directed graphs to model transportation networks:
  - edges carry traffic and have capacities.
  - nodes act as switches.
  - *source* nodes generate traffic, *sink* nodes absorb traffic.



- A flow network is a directed graph $G = (V, E)$
  - Each edge $e \in E$ has a capacity $c(e) > 0$.

Flow Networks

- Use directed graphs to model transportation networks:
  - edges carry traffic and have capacities.
  - nodes act as switches.
  - *source* nodes generate traffic, *sink* nodes absorb traffic.



- A flow network is a directed graph $G = (V, E)$
  - Each edge $e \in E$ has a capacity $c(e) > 0$.
  - There is a single source node $s \in V$.
  - There is a single sink node $t \in V$.
  - Nodes other than s and t are internal.

Defining Flow

- In a flow network $G = (V, E)$, an s-t flow is a function $f : E \to \mathbb{R}^+$ such that
  - (i) Capacity conditions For each $e \in E$, $0 \leq f(e) \leq c(e)$.
  - (ii) Conservation conditions For each internal node v,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

- The value of a flow is $\nu(f) = \sum_{e \text{ out of } s} f(e)$.

Defining Flow

- In a flow network $G = (V, E)$, an s-t flow is a function $f : E \to \mathbb{R}^+$ such that
  (i) Capacity conditions For each $e \in E$, $0 \leq f(e) \leq c(e)$.
  (ii) Conservation conditions For each internal node v,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

- The value of a flow is $\nu(f) = \sum_{e \text{ out of } s} f(e)$.
- Useful notation:

$f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$ $\quad$ $f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$

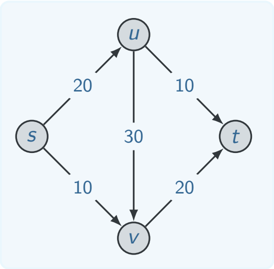
For $S \subseteq V$,

$f^{\text{out}}(S) = \sum_{e \text{ out of } S} f(e)$ $\quad$ $f^{\text{in}}(S) = \sum_{e \text{ into } S} f(e)$

Maximum Flow

INSTANCE A flow network G

SOLUTION The flow with largest value in G

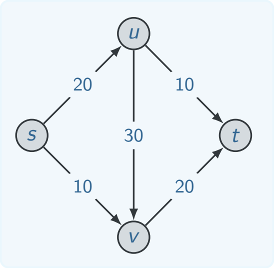


- Assumptions:
  1. No edges enter s, no edges leave t.

Maximum Flow

INSTANCE A flow network G

SOLUTION The flow with largest value in G

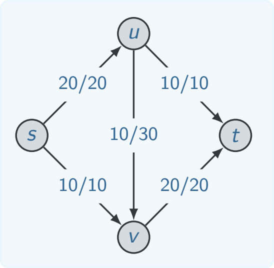


- Assumptions:
  1. No edges enter s, no edges leave t.
  2. There is at least one edge incident on each node.

Maximum Flow

INSTANCE A flow network G

SOLUTION The flow with largest value in G



- Assumptions:
  1. No edges enter s, no edges leave t.
  2. There is at least one edge incident on each node.
  3. All edge capacities are integers.

Algorithm design and analysis

— Ford-Fulkerson Algorithm —

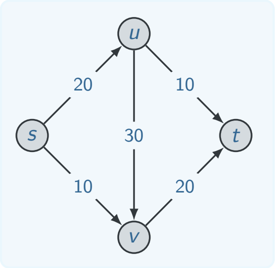
Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

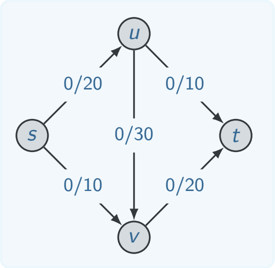
Developing the Algorithm

- A flow network is a directed graph $G = (V, E)$



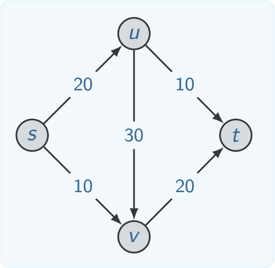
Developing the Algorithm

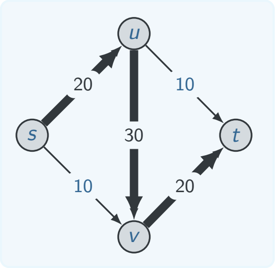
- A flow network is a directed graph $G = (V, E)$
- Let us take a greedy approach.
  1. Start with zero flow along all edges.



Developing the Algorithm

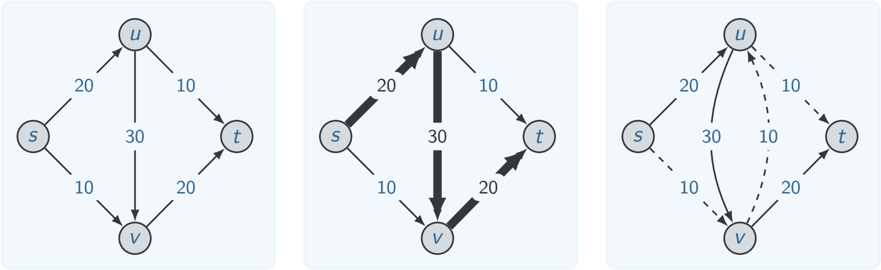
- A flow network is a directed graph $G = (V, E)$
- Let us take a greedy approach.
  1. Start with zero flow along all edges.
  2. Find an s-t path and push as much flow along it as possible.





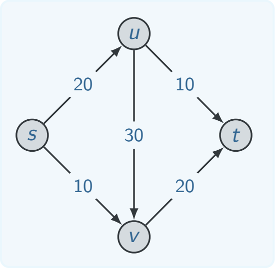
Developing the Algorithm

- A flow network is a directed graph $G = (V, E)$
- Let us take a greedy approach.
  1. Start with zero flow along all edges.
  2. Find an s-t path and push as much flow along it as possible.
  3. Key idea: Push flow along edges with leftover capacity and undo flow on edges already carrying flow.



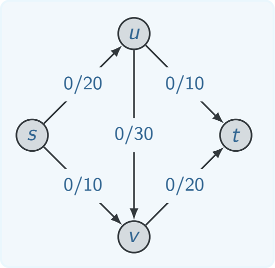
Residual Graph

- Given a flow network $G = (V, E)$ and a flow f on G, the residual graph G_f of G with respect to f is a directed graph such that
  (i) *Nodes* – G_f has the same nodes as G.



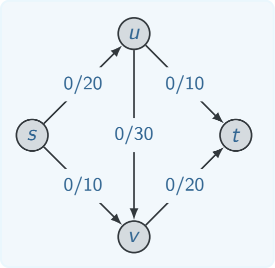
Residual Graph

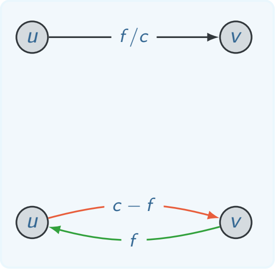
- Given a flow network $G = (V, E)$ and a flow f on G, the residual graph G_f of G with respect to f is a directed graph such that
  (i) *Nodes* – G_f has the same nodes as G.



Residual Graph

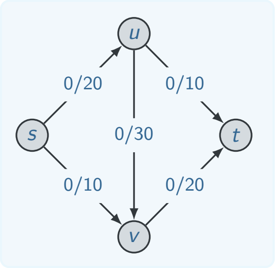
- Given a flow network $G = (V, E)$ and a flow f on G, the **residual graph** G_f of G with respect to f is a directed graph such that
  - (i) *Nodes* – G_f has the same nodes as G.
  - (ii) **Forward** edges – For each edge $e = (u, v) \in E$ such that $f(e) < c(e)$, G_f contains the edge (u, v) with a **residual capacity** $c(e) - f(e)$.
  - (iii) **Backward** edges – For each edge $e \in E$ such that $f(e) > 0$, G_f contains the edge $e' = (v, u)$ with a residual capacity $f(e)$.

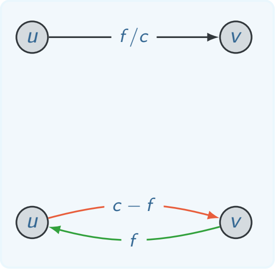


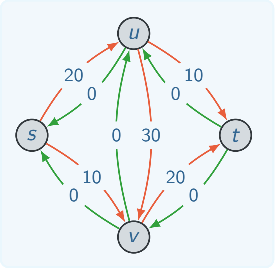


Residual Graph

- Given a flow network $G = (V, E)$ and a flow f on G, the **residual graph** G_f of G with respect to f is a directed graph such that
  (i) *Nodes* – G_f has the same nodes as G.
  (ii) **Forward** edges – For each edge $e = (u, v) \in E$ such that $f(e) < c(e)$, G_f contains the edge (u, v) with a **residual capacity** $c(e) - f(e)$.
  (iii) **Backward** edges – For each edge $e \in E$ such that $f(e) > 0$, G_f contains the edge $e' = (v, u)$ with a residual capacity $f(e)$.

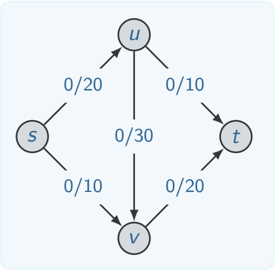






Augmenting Paths in a Residual Graph

- Let P be a simple s-t path in G_f.
- bottleneck(P, f) is the minimum residual capacity of any edge in P.
- The following operation `augment`(f, P) yields a new flow f' in G:



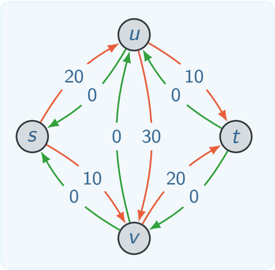
Algorithm: Augmented path

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.
output: The distances of the vertices from s

```
Let b = bottleneck(P, f) ;
foreach edge e = (u, v) ∈ P do
    if e is a forward edge then
        increase f(e) in G by b
    else if e is a backward edge then
        decrease f(e) in G by b;
end
```

Augmenting Paths in a Residual Graph

- Let P be a simple s-t path in G_f.
- bottleneck(P, f) is the minimum residual capacity of any edge in P.
- The following operation `augment`(f, P) yields a new flow f' in G:



Algorithm: Augmented path

---

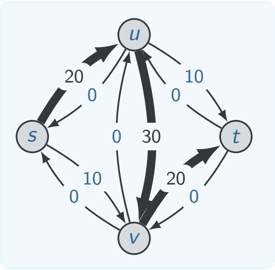
input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.
output: The distances of the vertices from s

```
Let b = bottleneck(P, f) ;
foreach edge e = (u, v) ∈ P do
    if e is a forward edge then
        increase f(e) in G by b
    else if e is a backward edge then
        decrease f(e) in G by b;
end

```

Augmenting Paths in a Residual Graph

- Let P be a simple s-t path in G_f.
- bottleneck(P, f) is the minimum residual capacity of any edge in P.
- The following operation `augment`(f, P) yields a new flow f' in G:



Algorithm: Augmented path

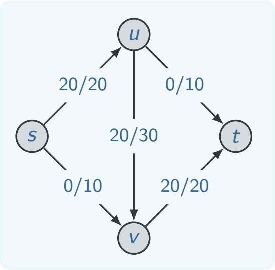
---

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.
output: The distances of the vertices from s

```
Let b = bottleneck(P, f) ;
foreach edge e = (u, v) ∈ P do
    if e is a forward edge then
        increase f(e) in G by b
    else if e is a backward edge then
        decrease f(e) in G by b;
end
```

Augmenting Paths in a Residual Graph

- Let P be a simple s-t path in G_f.
- bottleneck(P, f) is the minimum residual capacity of any edge in P.
- The following operation `augment`(f, P) yields a new flow f' in G:



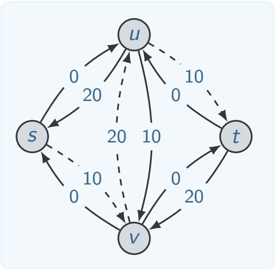
Algorithm: Augmented path

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.
output: The distances of the vertices from s

```
Let b = bottleneck(P, f) ;
foreach edge e = (u, v) ∈ P do
    if e is a forward edge then
        increase f(e) in G by b
    else if e is a backward edge then
        decrease f(e) in G by b;
end
```

Augmenting Paths in a Residual Graph

- Let P be a simple s-t path in G_f.
- bottleneck(P, f) is the minimum residual capacity of any edge in P.
- The following operation `augment`(f, P) yields a new flow f' in G:



Algorithm: Augmented path

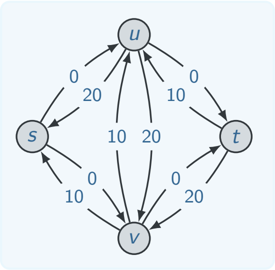
input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.

output: The distances of the vertices from s

Let b = bottleneck(P, f) ;
foreach *edge* $e = (u, v) \in P$ **do**
  if *e is a forward edge* **then**
    increase $f(e)$ in G by b
  else if *e is a backward edge* **then**
    decrease $f(e)$ in G by b;
end

Augmenting Paths in a Residual Graph

- Let P be a simple s-t path in G_f.
- bottleneck(P, f) is the minimum residual capacity of any edge in P.
- The following operation `augment`(f, P) yields a new flow f' in G:



Algorithm: Augmented path

---

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.
output: The distances of the vertices from s

```
Let b = bottleneck(P, f) ;
foreach edge e = (u, v) ∈ P do
    if e is a forward edge then
        increase f(e) in G by b
    else if e is a backward edge then
        decrease f(e) in G by b;
end
```

Correctness of `augment`(f, P)

- A simple s-t path in the residual graph is an augmenting path.
- Let f' be the flow returned by `augment`(f, P).
- Claim: f' is a flow. Verify capacity and conservation conditions.

Correctness of $\texttt{augment}(f, P)$

- A simple s-t path in the residual graph is an augmenting path.
- Let f' be the flow returned by $\texttt{augment}(f, P)$.
- Claim: f' is a flow. Verify capacity and conservation conditions.
  - Only need to check edges and internal nodes in P.

Correctness of augment(f, P)

- A simple s-t path in the residual graph is an augmenting path.
- Let f' be the flow returned by augment(f, P).
- Claim: f' is a flow. Verify capacity and conservation conditions.
  - Only need to check edges and internal nodes in P.
  - Capacity condition on $e = (u, v) \in G_f$: Note that bottleneck(P, f) $\leq$ residual capacity of (u, v).

Correctness of $\texttt{augment}(f, P)$

- A simple s-t path in the residual graph is an augmenting path.
- Let f' be the flow returned by $\texttt{augment}(f, P)$.
- Claim: f' is a flow. Verify capacity and conservation conditions.
  - Only need to check edges and internal nodes in P.
  - Capacity condition on $e = (u, v) \in G_f$: Note that bottleneck(P, f) $\leq$ residual capacity of (u, v).
    - e is a forward edge: $0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$.

Correctness of $\texttt{augment}(f, P)$

- A simple s-t path in the residual graph is an augmenting path.
- Let f' be the flow returned by $\texttt{augment}(f, P)$.
- Claim: f' is a flow. Verify capacity and conservation conditions.
  - Only need to check edges and internal nodes in P.
  - Capacity condition on $e = (u, v) \in G_f$: Note that bottleneck(P, f) $\leq$ residual capacity of (u, v).
    - e is a forward edge: $0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$.
    - e is a backward edge: $c(e) \geq f(e) \geq f'(e) = f(e) - \text{bottleneck}(P, f) \geq f(e) - f(e) = 0$.

Correctness of $\texttt{augment}(f, P)$

- A simple s-t path in the residual graph is an augmenting path.
- Let f' be the flow returned by $\texttt{augment}(f, P)$.
- Claim: f' is a flow. Verify capacity and conservation conditions.
  - Only need to check edges and internal nodes in P.
  - Capacity condition on $e = (u, v) \in G_f$: Note that bottleneck$(P, f) \leq$ residual capacity of (u, v).
    - e is a forward edge: $0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$.
    - e is a backward edge: $c(e) \geq f(e) \geq f'(e) = f(e) - \text{bottleneck}(P, f) \geq f(e) - f(e) = 0$.
  - Conservation condition on internal node $v \in P$.

Correctness of $\texttt{augment}(f, P)$

- A simple s-t path in the residual graph is an augmenting path.
- Let f' be the flow returned by $\texttt{augment}(f, P)$.
- Claim: f' is a flow. Verify capacity and conservation conditions.
  - Only need to check edges and internal nodes in P.
  - Capacity condition on $e = (u, v) \in G_f$: Note that bottleneck(P, f) $\leq$ residual capacity of (u, v).
    - e is a forward edge: $0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$.
    - e is a backward edge: $c(e) \geq f(e) \geq f'(e) = f(e) - \text{bottleneck}(P, f) \geq f(e) - f(e) = 0$.
  - Conservation condition on internal node $v \in P$. Four cases to work out.

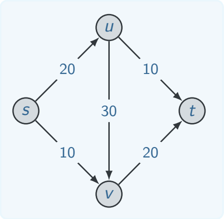
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

$f(e) = 0, \forall e \in E$;
while *there is a path s-t in the residual graph* G_f **do**
  Let P be a simple s-t path in G_f;
  $f' = augment(f, P)$;
  Update f to be f';
  Update the residual graph G_f to be $G_{f'}$;
end
return f;



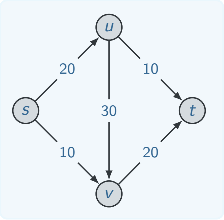
Ford-Fulkerson Algorithm

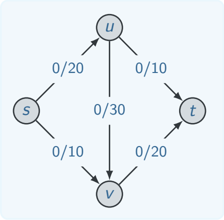
Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

$f(e) = 0, \forall e \in E$;
while *there is a path s-t in the residual graph G_f* **do**
    Let P be a simple s-t path in G_f;
    $f' = augment(f, P)$;
    Update f to be f';
    Update the residual graph G_f to be $G_{f'}$;
end
return f;



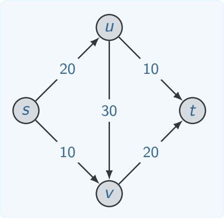


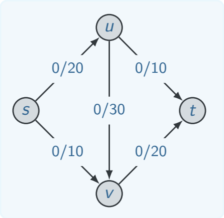
Algorithm: Ford-Fulkerson Algorithm

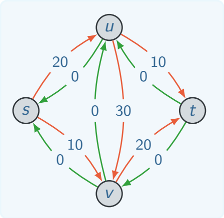
input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
f(e) = 0, ∀e ∈ E;
while there is a path s-t in the residual graph G_f do
    Let P be a simple s-t path in G_f;
    f' = augment(f, P);
    Update f to be f';
    Update the residual graph G_f to be G_f';
end
return f;
```





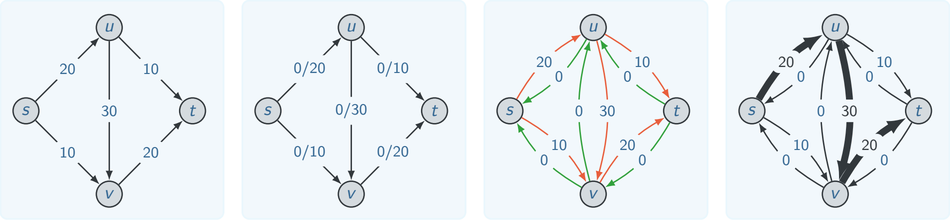


Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
f(e) = 0, ∀e ∈ E;
while there is a path s-t in the residual graph G_f do
    Let P be a simple s-t path in G_f;
    f' = augment(f, P);
    Update f to be f';
    Update the residual graph G_f to be G_f';
end
return f;
```



Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

$f(e) = 0, \forall e \in E$;

while *there is a path s-t in the residual graph G_f* **do**

  Let P be a simple s-t path in G_f;

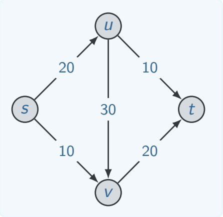
  $f' = augment(f, P)$;

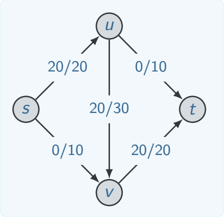
  Update f to be f';

  Update the residual graph G_f to be $G_{f'}$;

end

return f;



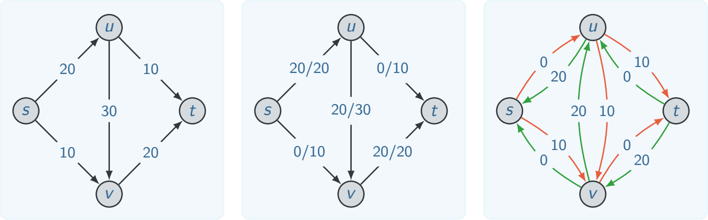


Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
f(e) = 0, ∀e ∈ E;
while there is a path s-t in the residual graph G_f do
    Let P be a simple s-t path in G_f;
    f' = augment(f, P);
    Update f to be f';
    Update the residual graph G_f to be G_f';
end
return f;
```



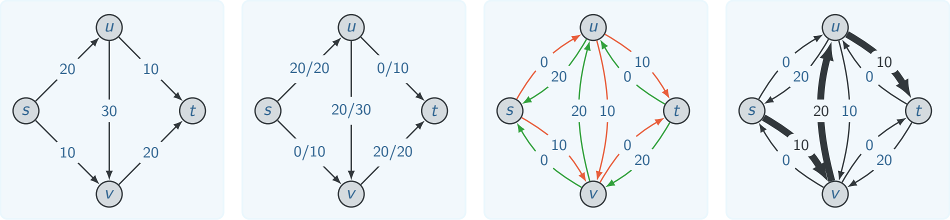
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

$f(e) = 0, \forall e \in E$;
while *there is a path s-t in the residual graph* G_f **do**
    Let P be a simple s-t path in G_f;
    $f' = augment(f, P)$;
    Update f to be f';
    Update the residual graph G_f to be $G_{f'}$;
end
return f;



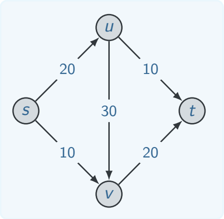
Ford-Fulkerson Algorithm

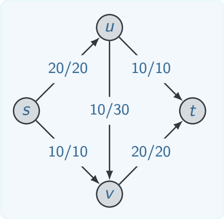
Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

$f(e) = 0, \forall e \in E$;
while *there is a path s-t in the residual graph* G_f **do**
  Let P be a simple s-t path in G_f;
  $f' = augment(f, P)$;
  Update f to be f';
  Update the residual graph G_f to be $G_{f'}$;
end
return f;





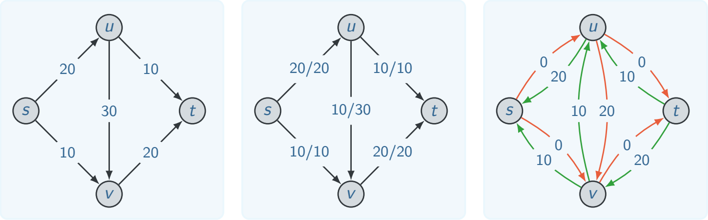
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

$f(e) = 0, \forall e \in E$;
while *there is a path s-t in the residual graph G_f* **do**
  Let P be a simple s-t path in G_f;
  $f' = augment(f, P)$;
  Update f to be f';
  Update the residual graph G_f to be $G_{f'}$;
end
return f;



Termination of the Ford-Fulkerson Algorithm

- Claim: at each stage, flow values and residual capacities are integers.

Termination of the Ford-Fulkerson Algorithm

- Claim: at each stage, flow values and residual capacities are integers. Prove by induction.

Termination of the Ford-Fulkerson Algorithm

- Claim: at each stage, flow values and residual capacities are integers. Prove by induction.
- Claim: Flow value strictly increases when we apply `augment`(f, P).

Termination of the Ford-Fulkerson Algorithm

- Claim: at each stage, flow values and residual capacities are integers. Prove by induction.
- Claim: Flow value strictly increases when we apply $\texttt{augment}(f, P)$. $v(f') = v(f) + \text{bottleneck}(P, f) > v(f)$.

Termination of the Ford-Fulkerson Algorithm

- Claim: at each stage, flow values and residual capacities are integers. Prove by induction.
- Claim: Flow value strictly increases when we apply $\texttt{augment}(f, P)$. $v(f') = v(f) + \text{bottleneck}(P, f) > v(f)$.
- Claim: Maximum value of any flow is $C = \sum_{e \text{ out of } s} c(e)$.

Termination of the Ford-Fulkerson Algorithm

- Claim: at each stage, flow values and residual capacities are integers. Prove by induction.
- Claim: Flow value strictly increases when we apply $\texttt{augment}(f, P)$. $v(f') = v(f) + \text{bottleneck}(P, f) > v(f)$.
- Claim: Maximum value of any flow is $C = \sum_{e \text{ out of } s} c(e)$.
- Claim: Algorithm terminates in at most C iterations.

Termination of the Ford-Fulkerson Algorithm

- Claim: at each stage, flow values and residual capacities are integers. Prove by induction.
- Claim: Flow value strictly increases when we apply `augment`(f, P). $v(f') = v(f) + \text{bottleneck}(P, f) > v(f)$.
- Claim: Maximum value of any flow is $C = \sum_{e \text{ out of } s} c(e)$.
- Claim: Algorithm terminates in at most C iterations.
- Claim: Algorithm runs in $O(mC)$ time.

Correctness of the Ford-Fulkerson Algorithm

- How large can the flow be?

- How large can the flow be?
- Can we characterise the magnitude of the flow in terms of the structure of the graph? For example, for every flow f, $\nu(f) \leq C = \sum_{e \text{out of } s} c(e)$.
- Is there a better bound?

- How large can the flow be?
- Can we characterise the magnitude of the flow in terms of the structure of the graph? For example, for every flow f, $\nu(f) \leq C = \sum_{e \text{out of } s} c(e)$.
- Is there a better bound?
- Idea: An **s-t cut** is a partition of V into sets A and B such that $s \in A$ and $t \in B$.
  - **Capacity** of the cut (A, B) is $c(A, B) = \sum_{e \text{ out of } A} c(e)$.
  - Intuition: For every flow f, $\nu(f) \leq c(A, B)$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
  - $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
  - $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
  - For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
  - $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
  - For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
  - $\nu(f) = \sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right)$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
  - $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
  - For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
  - $\nu(f) = \sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right)$.
    - An edge e that has both ends in A or both ends out of A does not contribute.
    - An edge e that has its tail in A contributes $f(e)$.
    - An edge e that has its head in A contributes $-f(e)$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
  - $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
  - For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
  - $\nu(f) = \sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right)$.
    - An edge e that has both ends in A or both ends out of A does not contribute.
    - An edge e that has its tail in A contributes $f(e)$.
    - An edge e that has its head in A contributes $-f(e)$.
  - $\sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A)$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
  - $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
  - For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
  - $\nu(f) = \sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right)$.
    - An edge e that has both ends in A or both ends out of A does not contribute.
    - An edge e that has its tail in A contributes $f(e)$.
    - An edge e that has its head in A contributes $-f(e)$.
  - $\sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
- Corollary: $\nu(f) = f^{\text{in}}(B) - f^{\text{out}}(B)$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
  - $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
  - For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
  - $\nu(f) = \sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right)$.
    - An edge e that has both ends in A or both ends out of A does not contribute.
    - An edge e that has its tail in A contributes $f(e)$.
    - An edge e that has its head in A contributes $-f(e)$.
  - $\sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
- Corollary: $\nu(f) = f^{\text{in}}(B) - f^{\text{out}}(B)$.
- $\boxed{\nu(f) \leq c(A, B)}$.

Fun Facts about Cuts

- Let f be any s-t flow and (A, B) any s-t cut.
- Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
  - $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
  - For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
  - $\nu(f) = \sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right)$.
    - An edge e that has both ends in A or both ends out of A does not contribute.
    - An edge e that has its tail in A contributes $f(e)$.
    - An edge e that has its head in A contributes $-f(e)$.
  - $\sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
- Corollary: $\nu(f) = f^{\text{in}}(B) - f^{\text{out}}(B)$.
- $\boxed{\nu(f) \leq c(A, B)}$.

$$\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A) \leq f^{\text{out}}(A) = \sum_{e \text{ out of } A} f(e)$$

$$\leq \sum_{e \text{ out of } A} c(e) = c(A, B).$$

Max-Flows and Min-Cuts

- Let f be any s-t flow and (A, B) any s-t cut. We proved $\nu(f) \leq c(A, B)$.

Max-Flows and Min-Cuts

- Let f be any s-t flow and (A, B) any s-t cut. We proved $\nu(f) \leq c(A, B)$.
- Very strong statement: The value of every flow is $\leq$ capacity of any cut.

Max-Flows and Min-Cuts

- Let f be any s-t flow and (A, B) any s-t cut. We proved $\nu(f) \leq c(A, B)$.
- Very strong statement: The value of every flow is $\leq$ capacity of any cut.
- Corollary: The maximum flow is, at most, the smallest capacity of a cut.

Max-Flows and Min-Cuts

- Let f be any s-t flow and (A, B) any s-t cut. We proved $\nu(f) \leq c(A, B)$.
- Very strong statement: The value of every flow is $\leq$ capacity of any cut.
- Corollary: The maximum flow is, at most, the smallest capacity of a cut.
- Question: Is the reverse true? Is the smallest capacity of a cut at most the maximum flow?

Max-Flows and Min-Cuts

- Let f be any s-t flow and (A, B) any s-t cut. We proved $\nu(f) \leq c(A, B)$.
- Very strong statement: The value of every flow is $\leq$ capacity of any cut.
- Corollary: The maximum flow is, at most, the smallest capacity of a cut.
- Question: Is the reverse true? Is the smallest capacity of a cut at most the maximum flow?
- Answer: Yes, and the Ford-Fulkerson algorithm computes this cut!

Flows and Cuts

- Let $\bar{f}$ denote the flow computed by the Ford-Fulkerson algorithm.
- Enough to show $\exists$ s-t cut (A^*, B^*) such that $\nu(\bar{f}) = c(A^*, B^*)$.
- When the algorithm terminates, the residual graph has no s-t path.

Flows and Cuts

- Let $\bar{f}$ denote the flow computed by the Ford-Fulkerson algorithm.
- Enough to show $\exists$ s-t cut (A^*, B^*) such that $\nu(\bar{f}) = c(A^*, B^*)$.
- When the algorithm terminates, the residual graph has no s-t path.
- Claim: If f is an s-t flow such that G_f has no s-t path, then there is an s-t cut (A^*, B^*) such that $\nu(f) = c(A^*, B^*)$.
  - Claim applies to *any* flow f such that G_f has no s-t path, and not just to the flow $\bar{f}$ computed by the Ford-Fulkerson algorithm.

Proof of Claim Relating Flows to Cuts

- Claim: f is an s-t flow and G_f has no s-t path $\Rightarrow \exists$ s-t cut (A^*, B^*), $\nu(f) = c(A^*, B^*)$.
- A^* = set of nodes reachable from s in G_f, $B^* = V - A^*$.

Proof of Claim Relating Flows to Cuts

- Claim: f is an s-t flow and G_f has no s-t path $\Rightarrow \exists$ s-t cut (A^*, B^*), $\nu(f) = c(A^*, B^*)$.
- $A^* =$ set of nodes reachable from s in G_f, $B^* = V - A^*$.
- Claim: (A^*, B^*) is an s-t cut.

Proof of Claim Relating Flows to Cuts

- Claim: f is an s-t flow and G_f has no s-t path $\Rightarrow \exists$ s-t cut (A^*, B^*), $\nu(f) = c(A^*, B^*)$.
- A^* = set of nodes reachable from s in G_f, $B^* = V - A^*$.
- Claim: (A^*, B^*) is an s-t cut.
- Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then .

Proof of Claim Relating Flows to Cuts

- Claim: f is an s-t flow and G_f has no s-t path $\Rightarrow \exists$ s-t cut (A^*, B^*), $\nu(f) = c(A^*, B^*)$.
- A^* = set of nodes reachable from s in G_f, $B^* = V - A^*$.
- Claim: (A^*, B^*) is an s-t cut.
- Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then $f(e) = c(e)$.

Proof of Claim Relating Flows to Cuts

- Claim: f is an s-t flow and G_f has no s-t path $\Rightarrow \exists$ s-t cut (A^*, B^*), $\nu(f) = c(A^*, B^*)$.
- $A^* =$ set of nodes reachable from s in G_f, $B^* = V - A^*$.
- Claim: (A^*, B^*) is an s-t cut.
- Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then $f(e) = c(e)$.
- Claim: If $e' = (u', v')$ such that $u' \in B^*$, $v' \in A^*$, then .

Proof of Claim Relating Flows to Cuts

- Claim: f is an s-t flow and G_f has no s-t path $\Rightarrow \exists$ s-t cut (A^*, B^*), $\nu(f) = c(A^*, B^*)$.
- A^* = set of nodes reachable from s in G_f, $B^* = V - A^*$.
- Claim: (A^*, B^*) is an s-t cut.
- Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then $f(e) = c(e)$.
- Claim: If $e' = (u', v')$ such that $u' \in B^*$, $v' \in A^*$, then $f(e') = 0$.

Proof of Claim Relating Flows to Cuts

- Claim: f is an s-t flow and G_f has no s-t path $\Rightarrow \exists$ s-t cut (A^*, B^*), $\nu(f) = c(A^*, B^*)$.
- A^* = set of nodes reachable from s in G_f, $B^* = V - A^*$.
- Claim: (A^*, B^*) is an s-t cut.
- Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then $f(e) = c(e)$.
- Claim: If $e' = (u', v')$ such that $u' \in B^*$, $v' \in A^*$, then $f(e') = 0$.
- Claim: $\nu(f) = c(A^*, B^*)$.

Max-Flow Min-Cut Theorem

- The flow $\bar{f}$ computed by the Ford-Fulkerson algorithm is a maximum flow.
- Given a flow of maximum value, we can compute a minimum s-t cut in $O(m)$ time.

Max-Flow Min-Cut Theorem

- The flow $\bar{f}$ computed by the Ford-Fulkerson algorithm is a maximum flow.
- Given a flow of maximum value, we can compute a minimum s-t cut in $O(m)$ time.
- In every flow network, there is a flow f and a cut (A, B) such that $\nu(f) = c(A, B)$.
- Max-Flow Min-Cut Theorem: in every flow network, the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut.

Max-Flow Min-Cut Theorem

- The flow $\bar{f}$ computed by the Ford-Fulkerson algorithm is a maximum flow.
- Given a flow of maximum value, we can compute a minimum s-t cut in $O(m)$ time.
- In every flow network, there is a flow f and a cut (A, B) such that $\nu(f) = c(A, B)$.
- Max-Flow Min-Cut Theorem: in every flow network, the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut.
- Corollary: If all capacities in a flow network are integers, then there is a maximum flow f where every flow value $f(e)$ is an integer.

Algorithm design and analysis

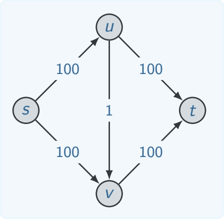
— Scaling Max-Flow Algorithm —

Silvio Guimarães

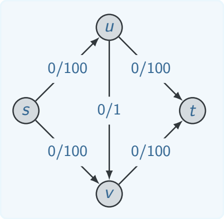
Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

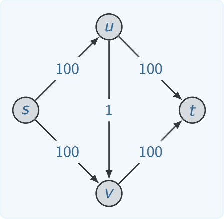
Feb 2023

Bad Augmenting Paths

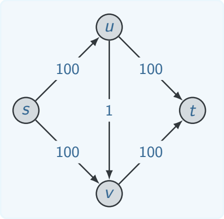


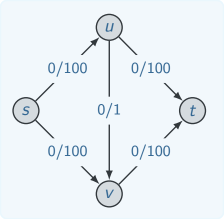
Bad Augmenting Paths

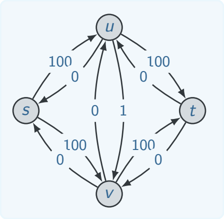




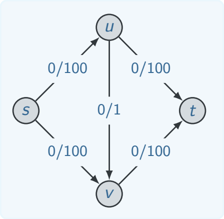
Bad Augmenting Paths

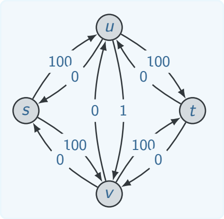


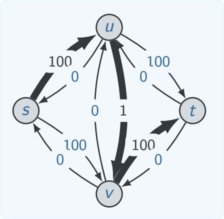


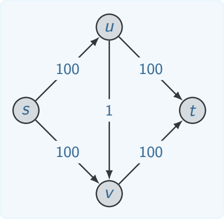


Bad Augmenting Paths

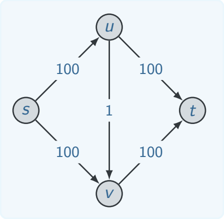


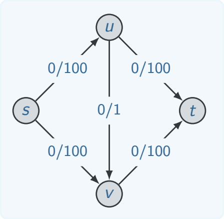


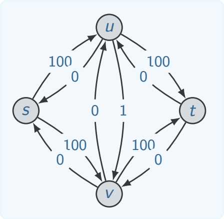


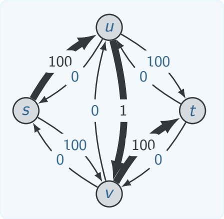


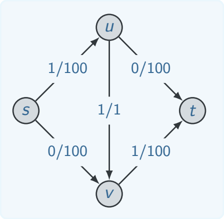
Bad Augmenting Paths



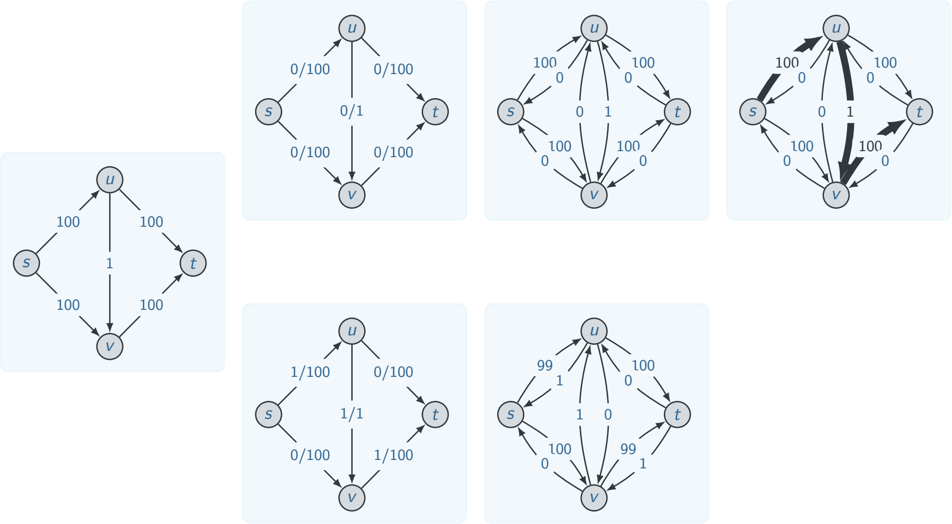




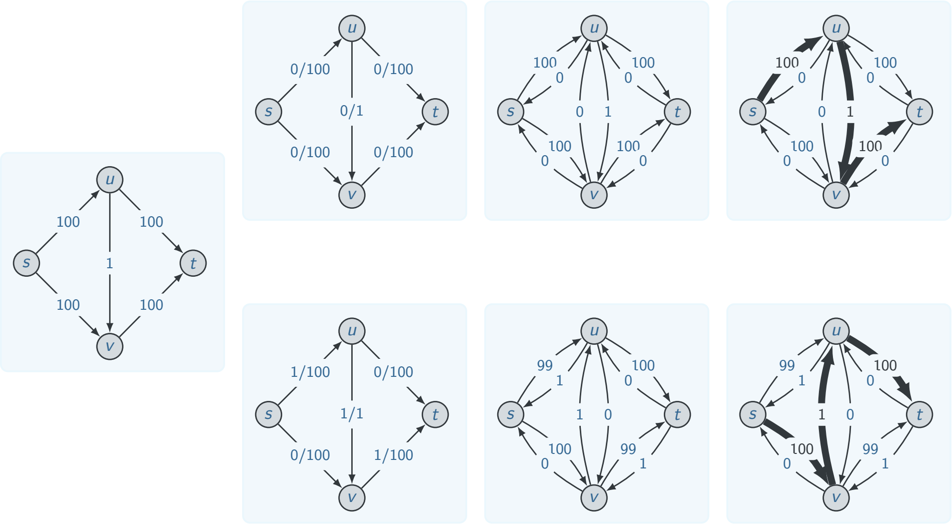




Bad Augmenting Paths



Bad Augmenting Paths



Improving Ford-Fulkerson Algorithm

- Bad case for Ford-Fulkerson algorithm is when the bottleneck edge is the augmenting path has a low capacity.
- Idea: decrease number of iterations by picking s-t path with bottleneck edge of largest capacity.

Improving Ford-Fulkerson Algorithm

- Bad case for Ford-Fulkerson algorithm is when the bottleneck edge is the augmenting path has a low capacity.
- Idea: decrease number of iterations by picking s-t path with bottleneck edge of largest capacity. Computing this path can slow down each iteration considerably.

Other Maximum Flow Algorithms

- Running time of the Ford-Fulkerson algorithm is $O(mC)$, which is pseudo-polynomial: polynomial in the magnitudes of the numbers in the input.
- Desire a strongly polynomial algorithm: running time is depends only on the *size* of the graph and is *independent* of the numerical values of the capacities (as long as numerical operations take $O(1)$ time).

Other Maximum Flow Algorithms

- Running time of the Ford-Fulkerson algorithm is $O(mC)$, which is pseudo-polynomial: polynomial in the magnitudes of the numbers in the input.
- Desire a strongly polynomial algorithm: running time is depends only on the *size* of the graph and is *independent* of the numerical values of the capacities (as long as numerical operations take $O(1)$ time).
- Edmonds-Karp, Dinitz: choose augmenting path to be the shortest path in G_f (use breadth-first search). Algorithm runs in $O(mn)$ iterations.
- Improved algorithms take time $O(mn \log n)$, $O(n^3)$, etc. on augmenting paths. Runs in $O(n^2 m)$ or $O(n^3)$ time.

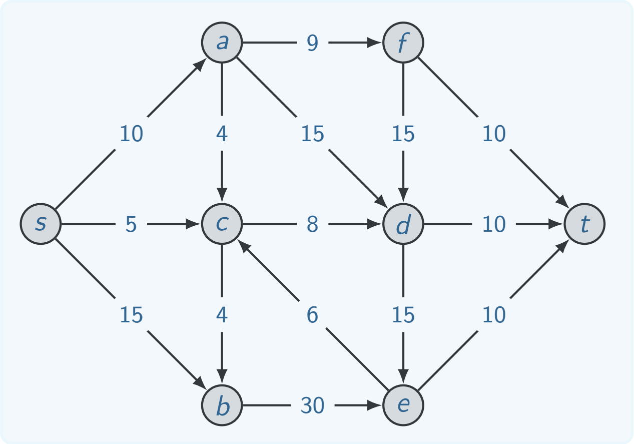
Algorithm design and analysis

— Exercises —

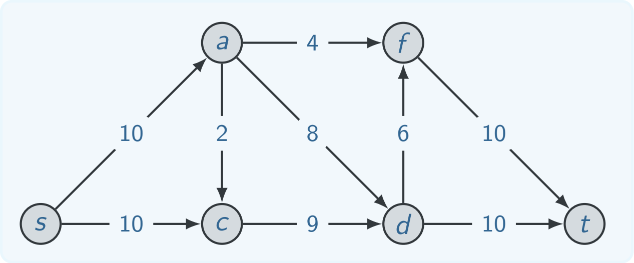
Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Compute the maximum flow



Compute the maximum flow

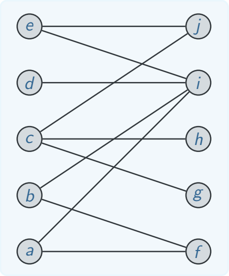


Bipartite graph matching

BIPARTITE GRAPH MATCHING

INSTANCE Let $G = (L \cup R, E)$ be an undirected graph. $M \subseteq E$ is a matching if each node appear in, at most, one edge in M.

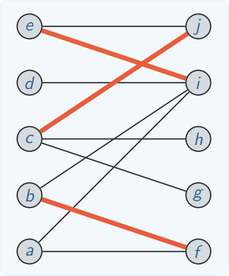
SOLUTION Find a max cardinality matching.



Bipartite graph matching

INSTANCE Let $G = (L \cup R, E)$ be an undirected graph. $M \subseteq E$ is a matching if each node appear in, at most, one edge in M.

SOLUTION Find a max cardinality matching.

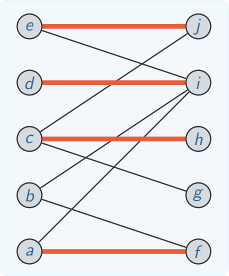


Bipartite graph matching

BIPARTITE GRAPH MATCHING

INSTANCE Let $G = (L \cup R, E)$ be an undirected graph. $M \subseteq E$ is a matching if each node appear in, at most, one edge in M.

SOLUTION Find a max cardinality matching.



Edge Disjoint Paths

DISJOINT PATH PROBLEM

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

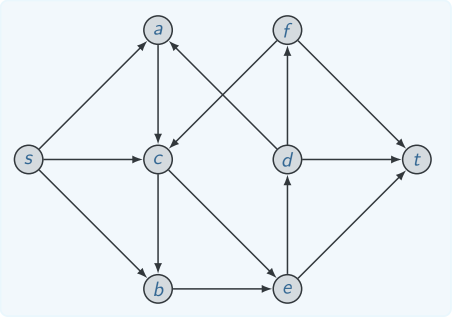
SOLUTION Find a max number of edge-disjoint s-t paths.

Edge Disjoint Paths

Disjoint path problem

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

SOLUTION Find a max number of edge-disjoint s-t paths.

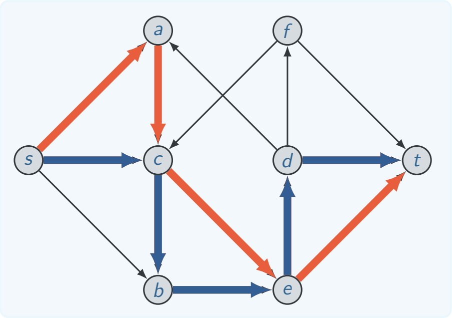


Edge Disjoint Paths

DISJOINT PATH PROBLEM

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

SOLUTION Find a max number of edge-disjoint s-t paths.



Network Connectivity

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

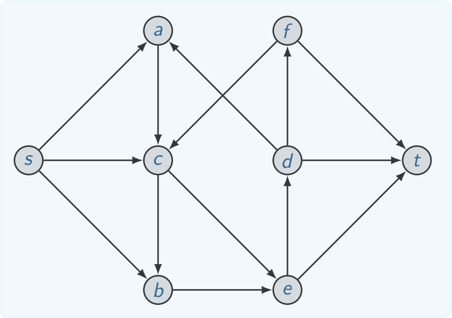
SOLUTION Find a min number of edges whose removal disconnects t from s

Network Connectivity

Network Connectivity

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

SOLUTION Find a min number of edges whose removal disconnects t from s



Network Connectivity

NETWORK CONNECTIVITY

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

SOLUTION Find a min number of edges whose removal disconnects t from s