Algorithms and Complexity

Network flows

Julián Mestre

School of Information Technologies
The University of Sydney

THE UNIVERSITY OF SYDNEY

Bipartite matching

A matching M is a subset of edges where no two edges are incident to the same vertex.

Matching model the assignment of applicants to jobs, employees to projects, units of study to classroom, etc.

Input:

- Undirected bipartite graph $G = (X, Y, E)$

Task:

- Find a maximum size matching M in G

Bipartite matchings

Matchings have been studied extensively since the early 20th century, which led to the development of a rich Matching Theory

Edmonds defined the notion of efficient computation to motivate his algorithm for finding a maximum matching in general graphs

Hall's marriage theorem gives a characterization of those graphs that admit a perfect matching

Thm. A bipartite graph (X,Y,E) has a perfect matching iff for all $S \subseteq X,Y$ we have $|S| \leq |N(S)|$

Network flow

Let $G=(V, E)$ be a directed graph with capacity function $c : E \rightarrow Z^+$

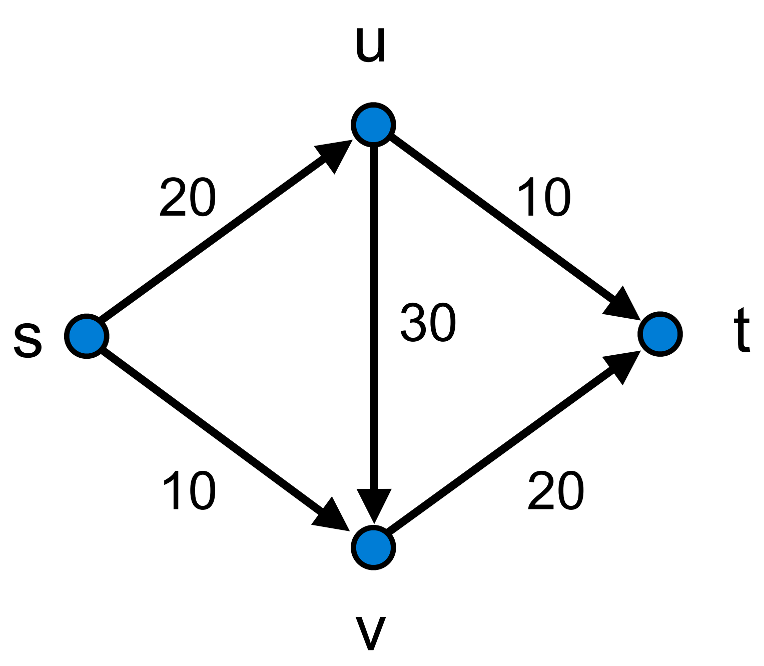
Let s and t be two distinguished vertices such that s has no incoming edges and t has no outgoing edges

A flow is a function $f : E \rightarrow Z^+$ obeying

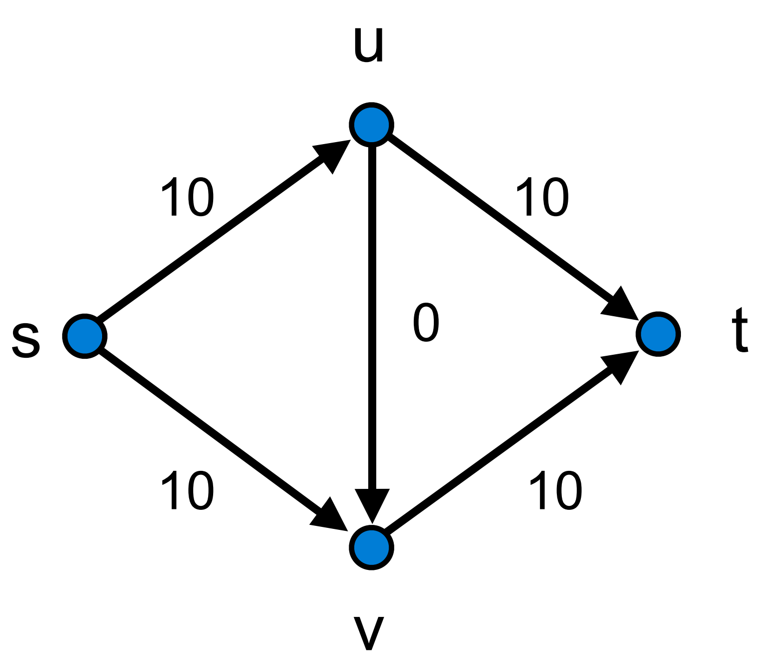
- [Capacity constraint] $f(e) \leq c(e)$ for all edges e in E
- [Flow conservation] $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ for all v in $V \setminus \{s,t\}$

The value of the flow f is defined as $v(f) = \sum_{e \text{ out of } s} f(e)$

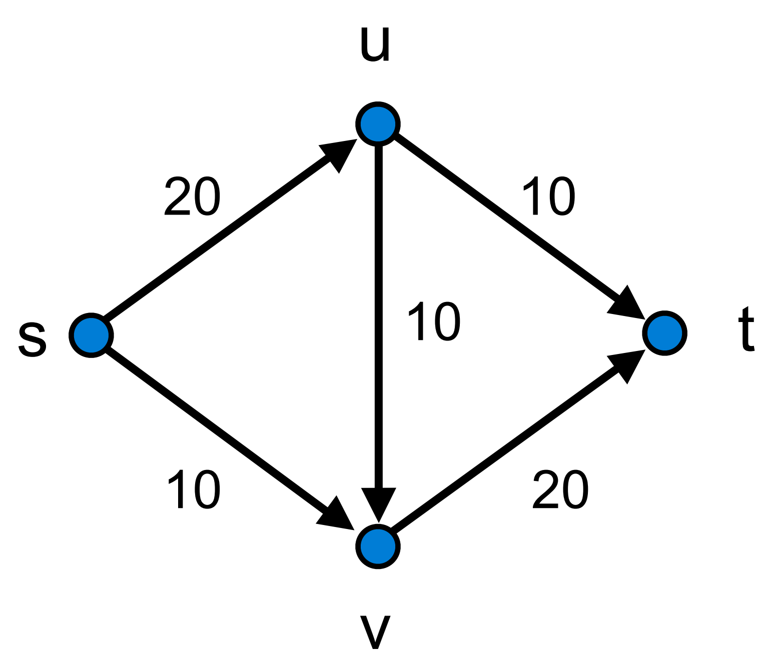
Example



Input graph with
edge capacities



A flow with
value 20



A flow with
value 30

Maximum flow problem

Input:

- Directed graph $G=(V, E)$
- capacity function $c : E \to Z^+$
- source s in V with no incoming edges
- sink t in V with no outgoing edges

Task:

- Find an s-t flow f maximizing $v(f)$

Today we will see the Ford-Fulkerson algorithm for finding a maximum flow and its application to bipartite matching

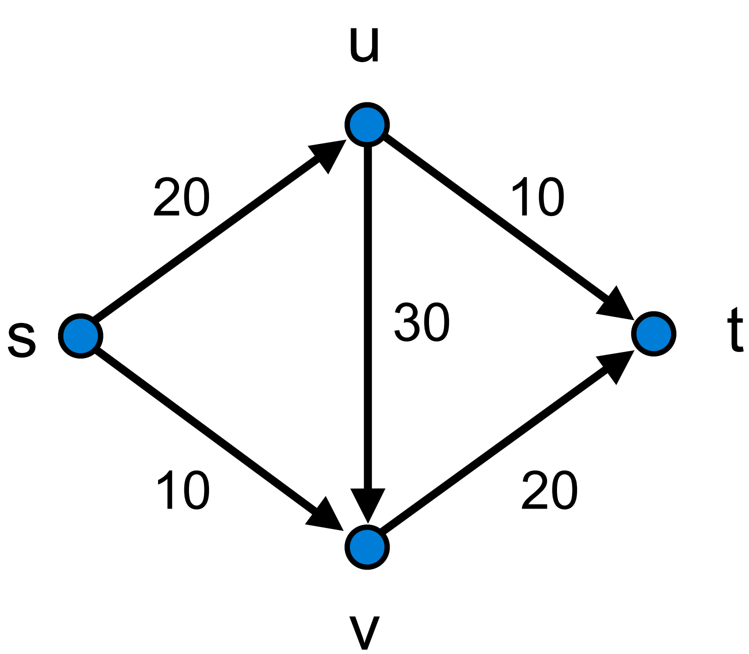
First try

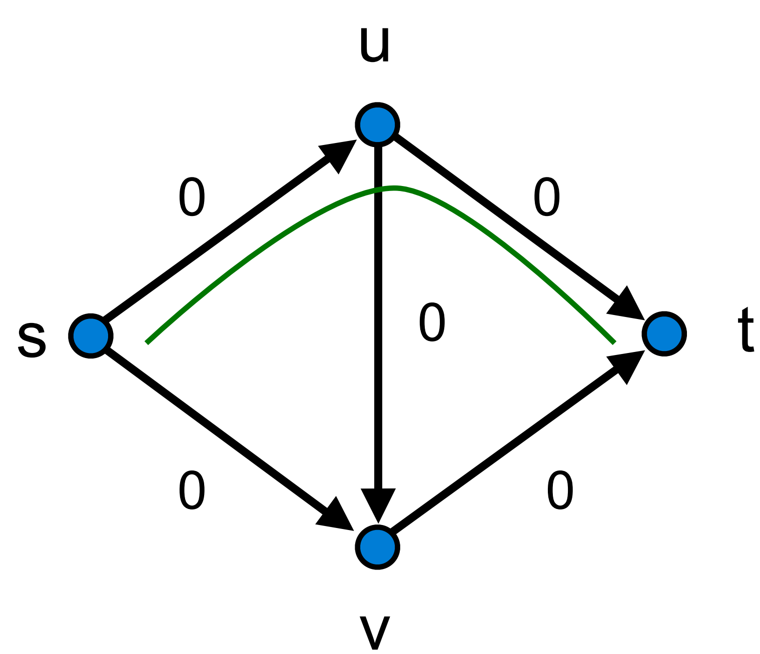
Greedy build a maximum flow by repeatedly choosing an unsaturated s-t path and pushing as much flow as possible along it

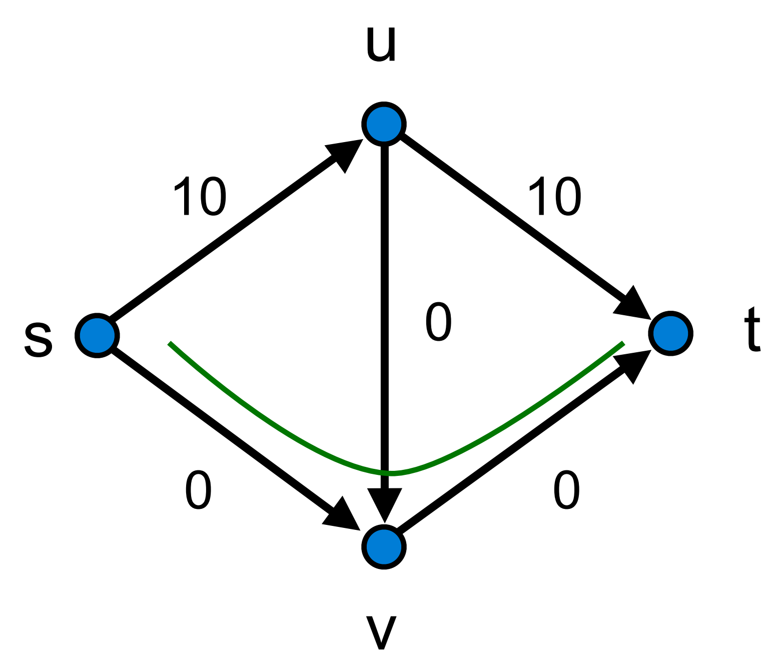
```
def greedy_flow(G=(V,E),s,t,c):

  f[e] = 0 for all e in E
  while ∃ s-t unsaturated path w.r.t. f:
    p = s-t unsaturated path w.r.t. f
    push_flow(p, f)
  return f
```

```
def push_flow(p,f)

  delta = min c(e) - f(e) for e in p
  for e in p:
    f[e] = f[e] + delta
```

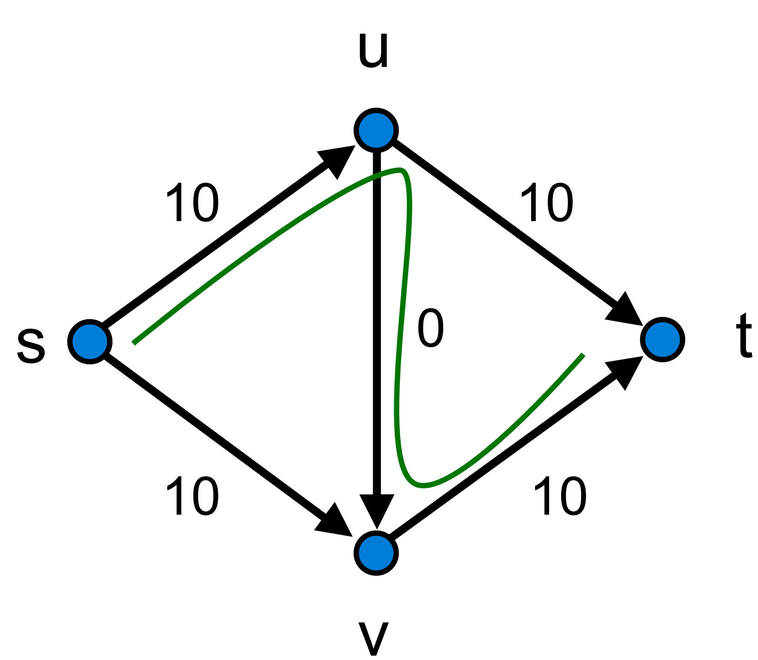
Good example

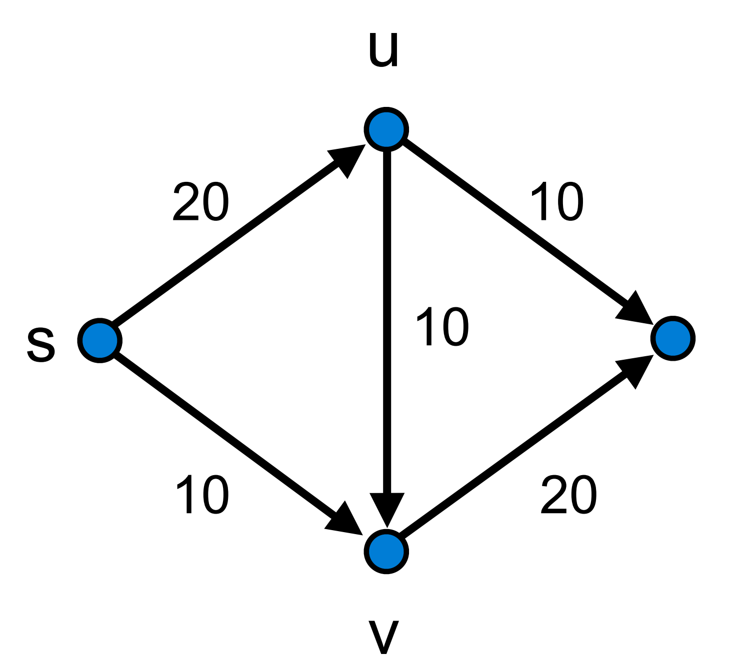




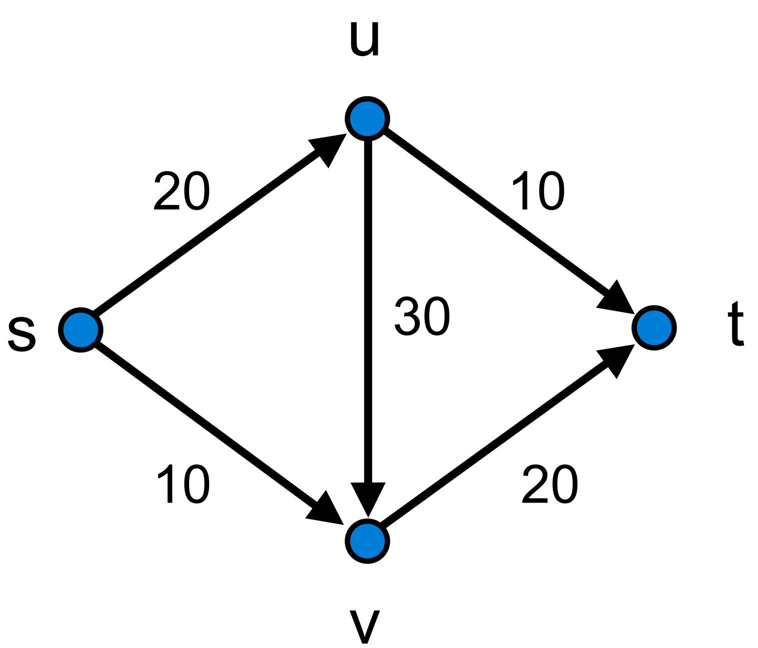


Input graph with edge capacities

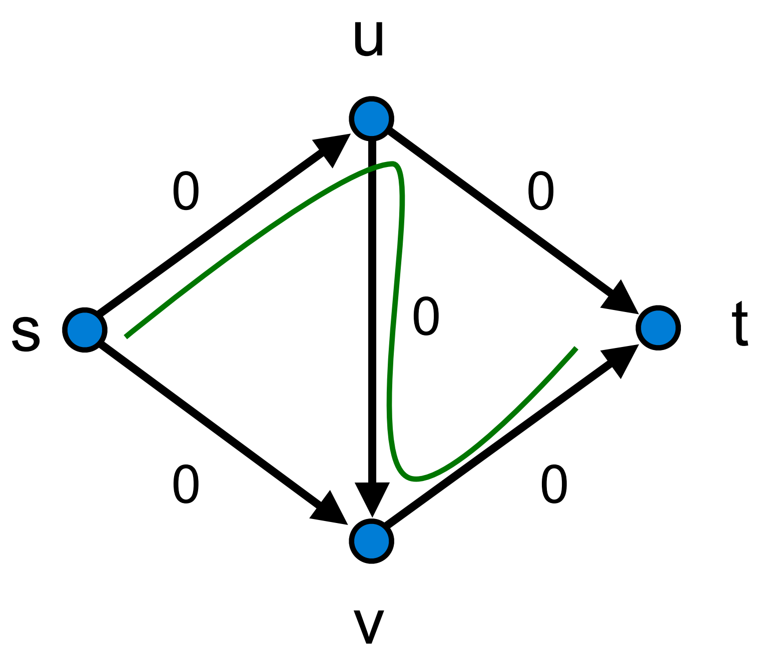


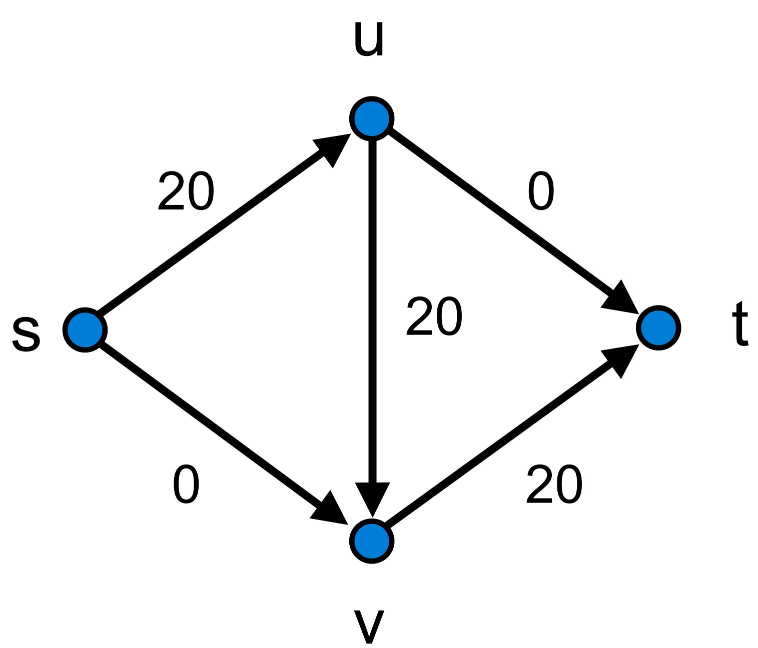


Bad example



Input graph with
edge capacities





We are stuck at
a suboptimal solution

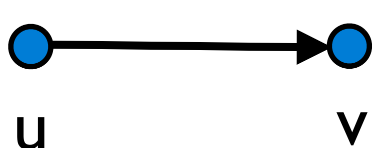
Residual graph

Let $f : E \to Z^+$ be an s-t flow for $G=(V, E)$ with capacities $c : E \to Z^+$

The residual graph $G^f = (V, E')$

- [Forward] If $f(u,v) < c(u,v)$ then $(u,v) \in E'$ with residual capacity $c(u,v) - f(u,v)$
- [Backward] If $f(u,v) > 0$ then $(v,u) \in E'$ with has residual capacity $f(u,v)$

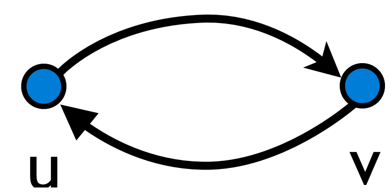
G



$c(u,v) = 15$

$f(u,v) = 5$

G_f



residual capacity of (u,v) = 10

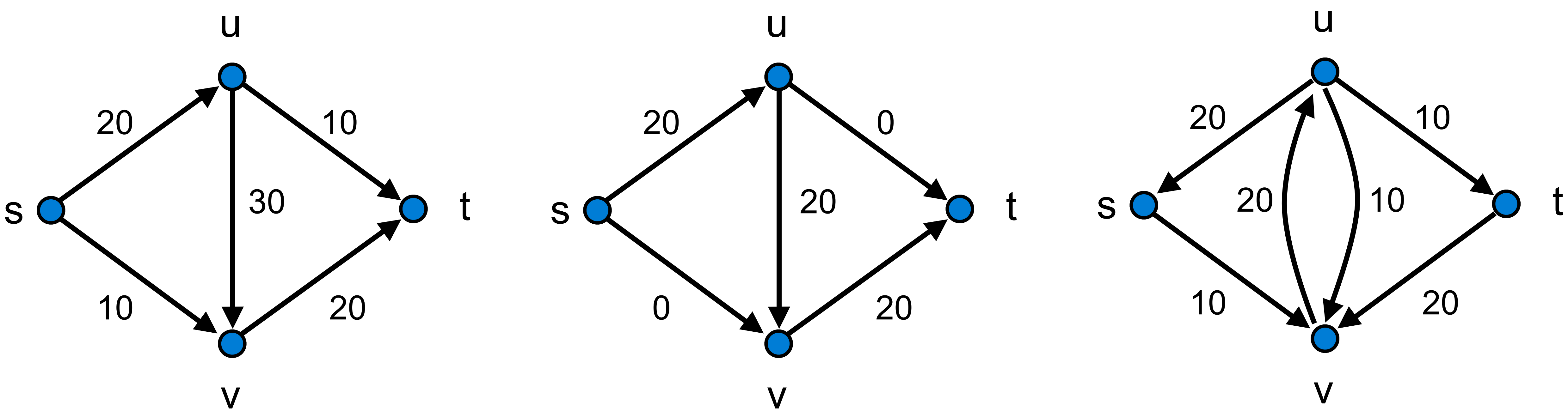
residual capacity of (v,u) = 5

Ford-Fulkerson

Greedy build a maximum flow by repeatedly choosing an s-t path in the residual graph and pushing as much flow as possible along it

```
def FF(G=(V,E),s,t,c):

  f[e] = 0 for all e in E
  while ∃ s-t path in Gf:
    p = some s-t path in Gf
    push_flow(p, f)
  return f


def push_flow(p,f,Gf)

  delta = min res. capacity for e in p
  for e in p:
    if e is forward in Gf:
      f[e] = f[e] + delta
    else:
      f[e] = f[e] - delta
```

Dealing with the bad example

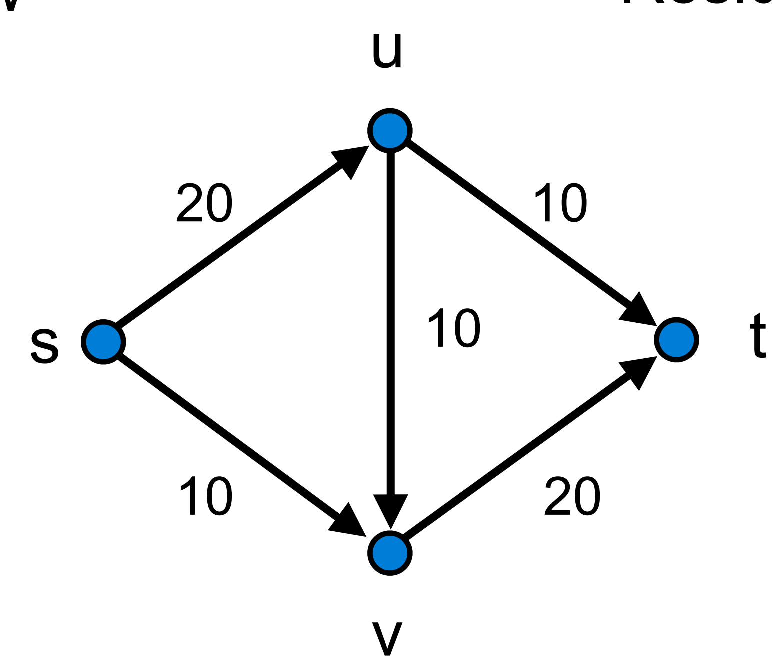


Input graph with edge capacities

Flow

Residual graph

After pushing along (s,v,u,t) we get:



Feasibility

Initially $f(e)=0$ for all edges e, so we start with a feasible flow

Let f' be the flow returned by push_flow(p, f, G_f)

We need to argue that

- f' obeys capacity constraints
- f' obeys flow conservation constraints

Obs. The FF algorithm returns a feasible flow

Time complexity

The graph G^f can be built in $O(m)$ from G and f

Finding the s-t path in G^f takes $O(m)$ time

Pushing flow along p takes $O(n)$ time, which is $O(m)$

Thus, each iteration takes at most $O(m)$ time

There are at most $C = \sum_{e \text{ out of } s} c(e)$ iterations

Obs. The FF algorithm terminates in $O(C\ m)$ time

Optimality

An s-t cut is a partition (A, B) of V such that s in A and t in B. We define its capacity as $c(A,B) = \sum_{u \text{ in } A \text{ and } v \text{ in } B} c(u,v)$

Intuitively, $v(f) \leq c(A,B)$ for any s-t flow f and s-t cut (A, B)

Let f be the flow output by the algorithm. We will construct an s-t cut (A, B) such that $v(f) = c(A,B)$. It will follow that

Obs. The FF algorithm returns a maximum flow

Ford-Fulkerson

Based on our observations, we know that

- FF returns a feasible flow
- FF terminates in O(C m) time
- FF returns an optimal flow

Notice that our proof of optimality implies that FF can be augmented to return a minimum s-t cut as well!

Thm.	The FF algorithm computes a maximum flow problem in O(C m) time

Back to matchings

Let (X, Y, E) be a bipartite graph. We create a network flow H:

- Vertex set of H is set to $X \cup Y \cup \{s,t\}$
- Connect s to every node in X
- Connect every node in Y to t
- If (u,v) in E then create a directed edge (u,v) in H
- $c(e) = 1$ for every edge e in H

It is easy to see that any flow f translates into a matching M such that $|M| = v(f)$, and vice-versa

Suppose the maximum flow does not induce a perfect matching. Then we can use max-flow min-cut duality to get Hall's theorem.

Hall's Theorem

Recall that for the bipartite graph (X, Y, E), we have a network H:

- Vertex set of H is $X \cup Y \cup \{s,t\}$
- Connect s to every node in X
- Connect every node in Y to t
- If (u,v) in E then create a directed edge (u,v) in H
- $c(e) = 1$ for every edge e in H

Thm.

A bipartite graph (X, Y, E) has a perfect matching iff for all $S \subseteq X, Y$ we have $|S| \leq |N(S)|$