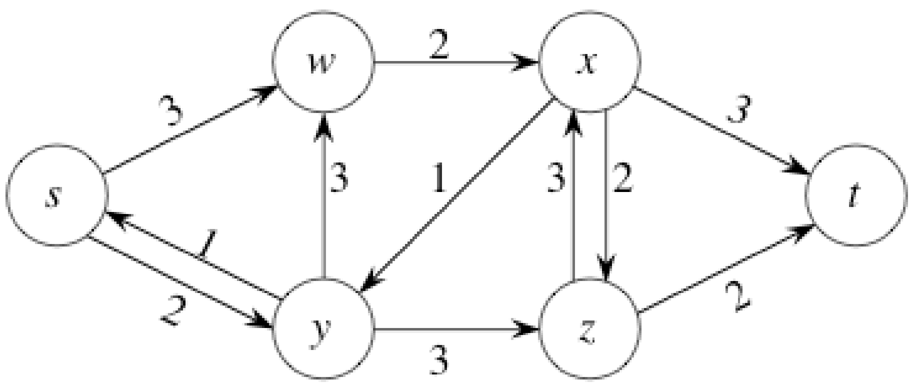
Graph Algorithms:
Maximum Flow

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

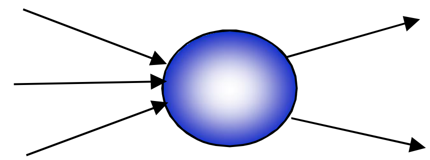
Flow Network

- Directed graph $G = (V, E)$ with non-negative edge weights $c : E \rightarrow R$
  - $c(u, v)$: nonnegative *capacity* of an edge $(u, v) \in E$
    - $c(u, v) = 0$ if $(u, v) \notin E$
  - s: source of the network
  - t: sink of the network



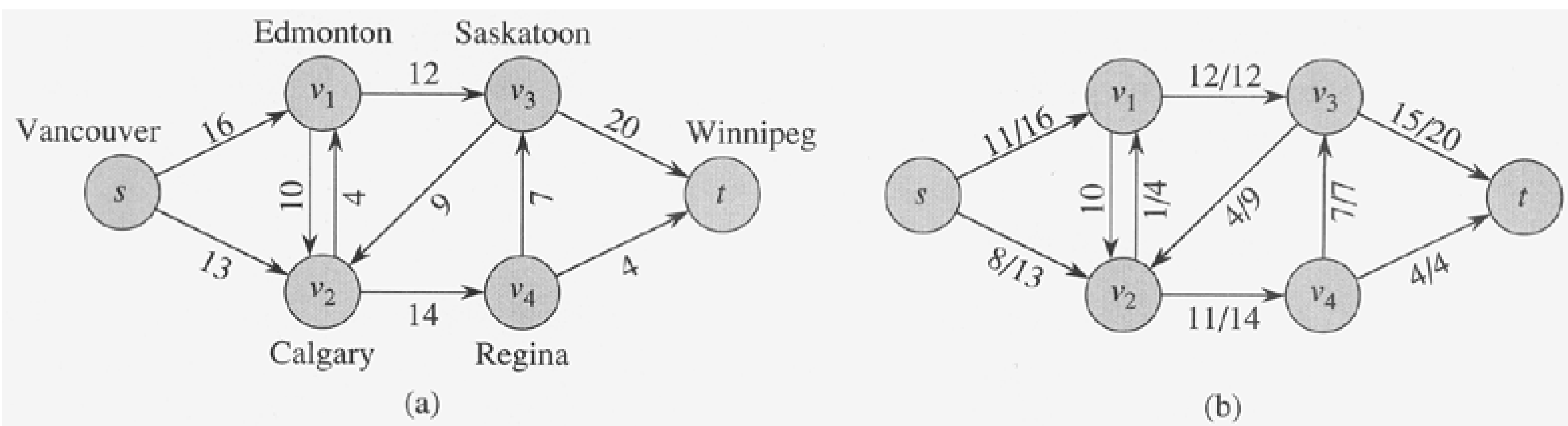
Flow Network

- A *positive flow* is a function $f: V \times V \rightarrow R$ s.t.,
  - Capacity constraint:
    - For all $u, v \in V$, $0 \leq f(u, v) \leq c(u, v)$



  - Flow conservation constraint:
    - For all $u \in V - \{s, t\}$, $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$

Flow-in equals flow-out

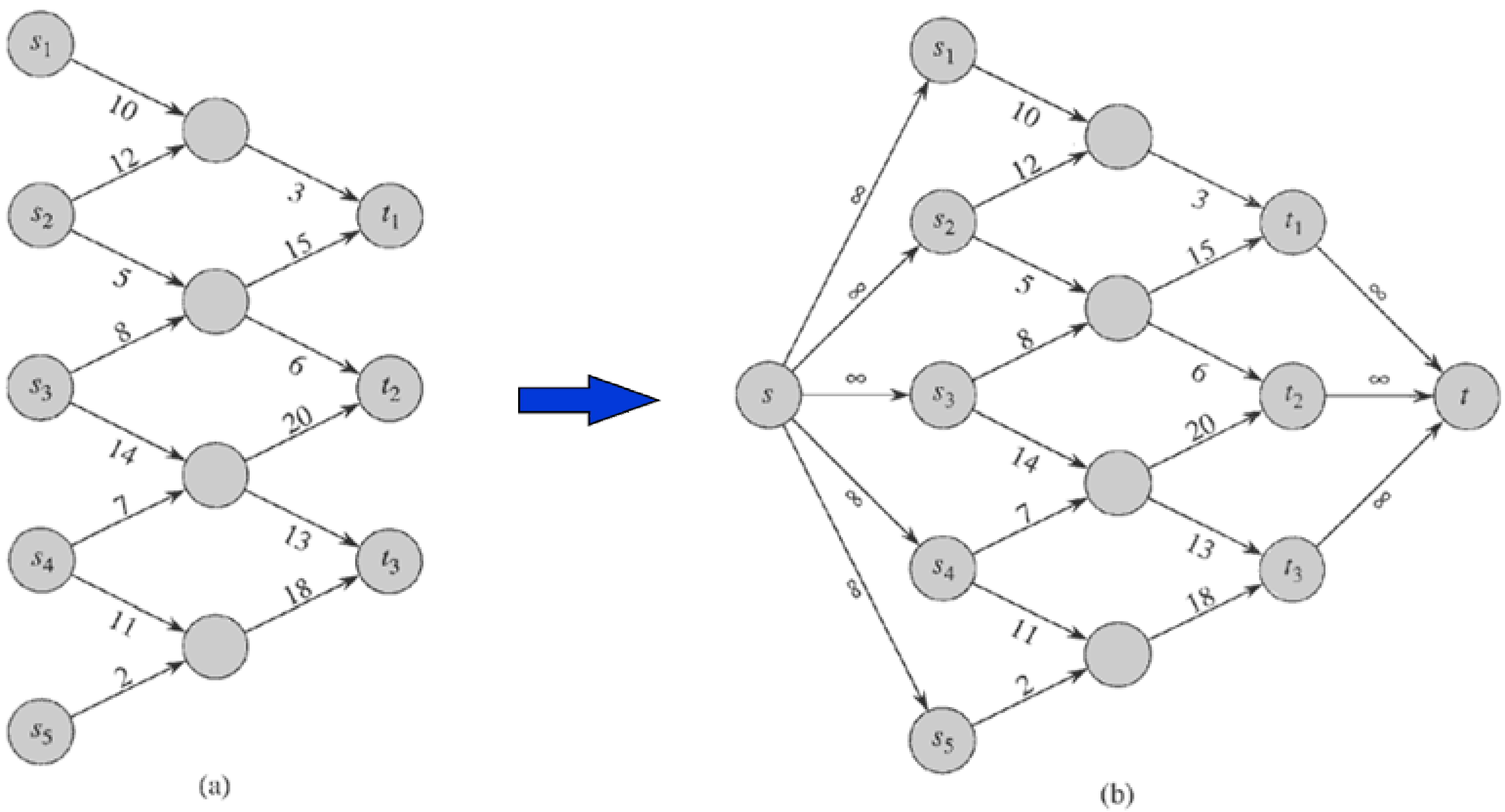


Maximum-Flow Problem

- Given a flow f, the value of f is
  - $|f| = \sum f(s, u)$: *i.e.*, total flow out of the source
- Maximum-flow problem:
  - Compute a flow of maximum value
- Multiple sources/sinks
  - Convert to single source/sink problem by adding one supersource and one supersink
- Anti-parallel edges or Two-way edges
  - Transform the network into an equivalent one containing no anti-parallel edges by adding a new vertex

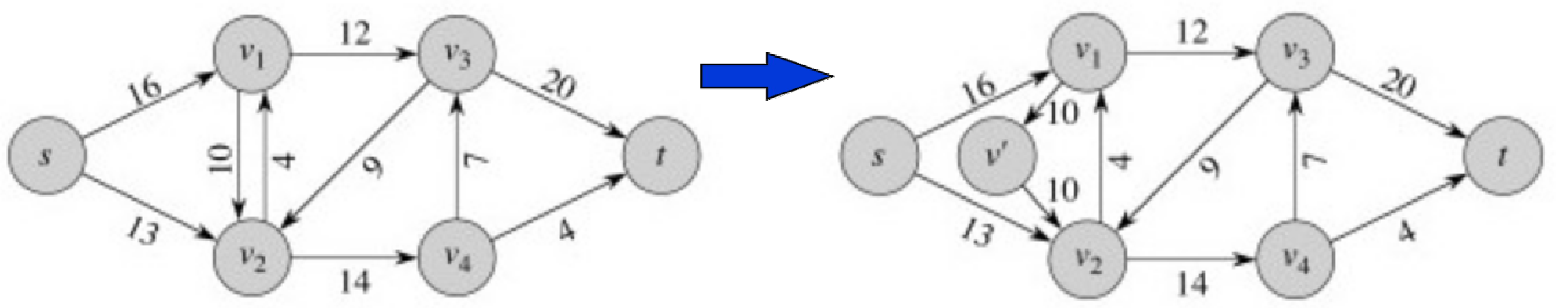
Maximum-Flow Problem

- Multiple sources/sinks
  - Convert to single source/sink problem by adding one supersource and one supersink



Maximum-Flow Problem

- Anti-parallel edges or Two-way edges
  - Transform the network into an equivalent one containing no anti-parallel edges by adding a new vertex and two edges having the same capacity as one of the anti-parallel edges.



(a) (b)

  - Flow networks without anti-parallel edges are easier to explain and process. It is not surprising if anti-parallel edges are avoided or excluded or disallowed for the sake of simplicity in many situations.

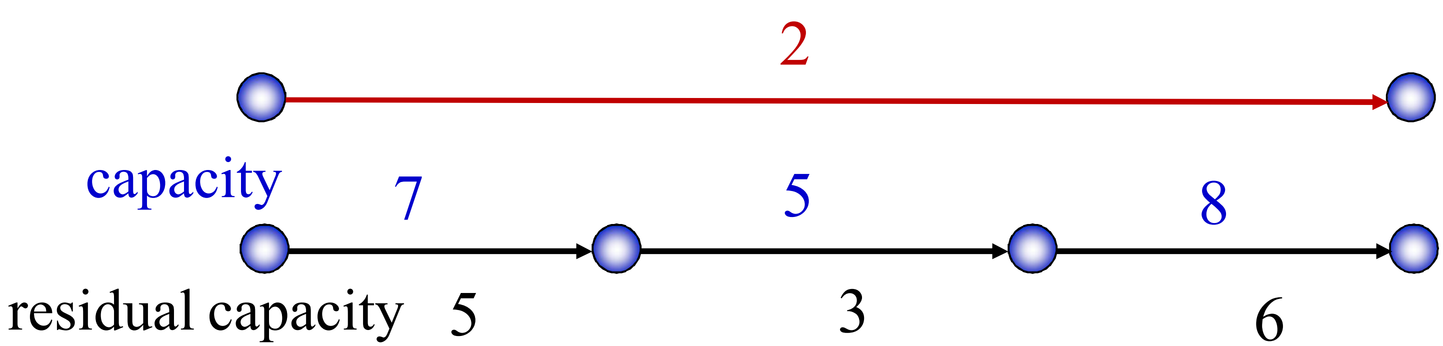
Maximum-Flow Problem

FORD-FULKERSON-METHOD(G, s, t)

initialize flow f to 0
while there exists an augmenting path p
    do augment flow f along p
return f

Residual Network

- Given a flow network and a flow, the residual network consists of edges that can admit more network flow.
- $G = (V, E)$: a flow network with source s and sink t
- f : a flow in G
- The amount of additional network flow from u to v before exceeding the capacity $c(u, v)$ is the residual capacity of (u, v), given by: $c_f(u, v) = c(u, v) - f(u, v)$



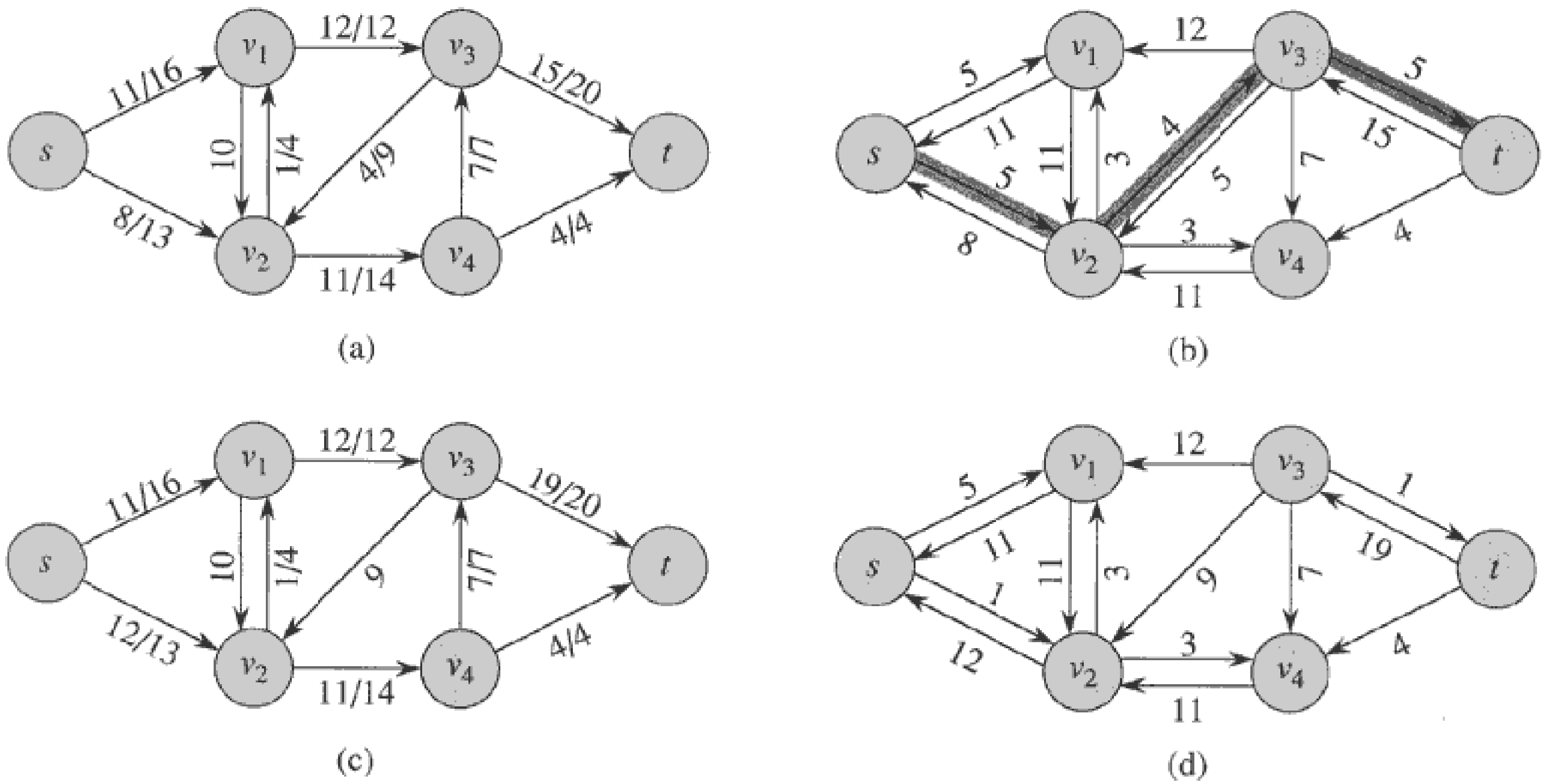
The residual capacity of the path is 3

Residual Network

- Given a flow f in a network $G = (V, E)$
  - The residual capacity between $u, v \in V$
    - $c_f(u, v) = c(u, v) - f(u, v)$ ≥ 0 !
  - Residual network $G_f = (V, E_f)$
    - where $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$
  - Residual network G_f may also contain edges that are not in G
  - Residual capacity, $c_f(u, v)$ is defined by

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (u,v) \notin E \\ 0 & \text{otherwise} \end{cases}$$

Residual Network: Example



(a) (b) (c) (d)

Each edge in G corresponds to at most two edges in residual network: $|E_f| \le 2|E|$

Residual Network

- A flow in a residual network provides a roadmap for adding flow to the original flow network.
  - If f is a flow in G and f' is a flow in G_f, we define $f \uparrow f'$, the *augmentation* of flow f by f'

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

- Pushing flow on the reverse edge in the residual network is known as *cancellation*.

Residual Network

- Lemma
  - Let $G = (V, E)$ be a flow network with source s and sink t, and let f be a flow in G.
  - Let G_f be the residual network of G induced by f, and let f' be a flow in G_f. Then the flow sum $f+f'$ is a flow in G with value $|f + f'| = |f| + |f'|$
  - $f + f'$: the flow in the same direction will be added.

    the flow in different directions will be cancelled.

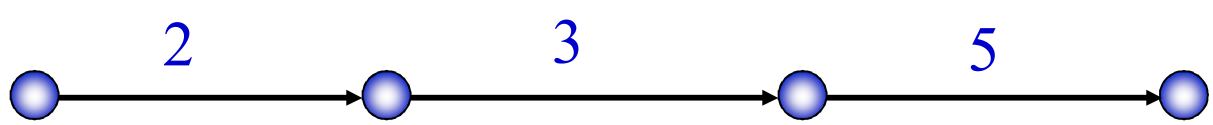
This suggests that we can improve current flow by computing a new flow for its residual network, and add it upon original one

Augmenting Path

- Given a flow f in a flow network $G = (V, E)$, an *augmenting path* p is a simple path from source s to sink t in the residual network G_f.
- How much extra flow can we push on an augmenting path p ?

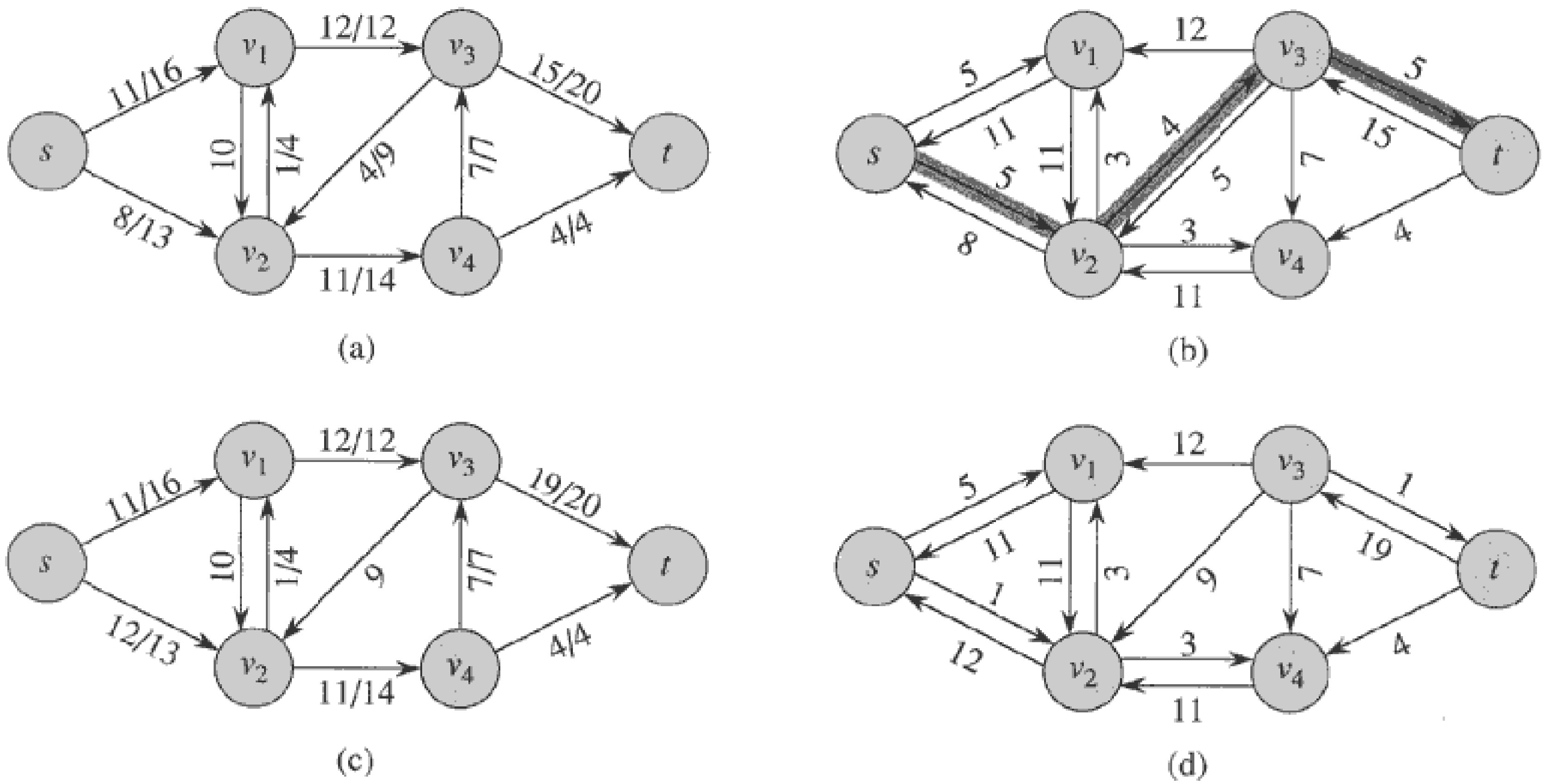
The maximum amount by which we increase the flow on each edge in an augmenting path p is the *residual capacity* of p, given by

$$c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is on } p\}$$



The residual capacity is 2

Augmenting Path: Example



Lemma: Augmenting -> Flow

Lemma:

Given flow network G, flow f in G, residual network G_f. Let p be an augmenting path in G_f. Define $f_p : V \times V \to \mathbf{R}$:

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \text{ is on } p \,, \\ -c_f(p) & \text{if } (v,u) \text{ is on } p \,, \\ 0 & \text{otherwise} \,. \end{cases}$$

Then f_p is a flow in G_f with value $|f_p| = c_f(p) > 0$.

Corollary:

Given flow network G, flow f in G, and an augmenting path p in G_f, define f_p as in lemma, and define $f' : V \times V \to \mathbf{R}$ by $f' = f + f_p$. Then f' is a flow in G with value $|f'| = |f| + c_f(p) > |f|$.

Cuts of Flow Networks

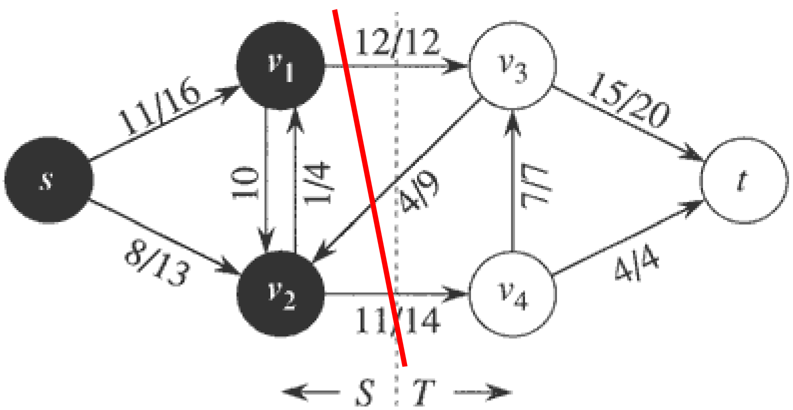
- A *cut* (S, T) of flow network $G = (V, E)$
  - is a partition of V into S and $T = V - S$, s.t. $s \in S$ and $t \in T$
  - The net flow $f(S, T)$ across cut (S, T) is

$$f(S,T) = \sum_{u \in S, v \in T} f(u,v) - \sum_{v \in T, u \in S} f(v,u)$$

  - The capacity $c(S, T)$ of cut (S, T) is

$$c(S,T) = \sum_{u \in S,\, v \in T} c(u,v)$$

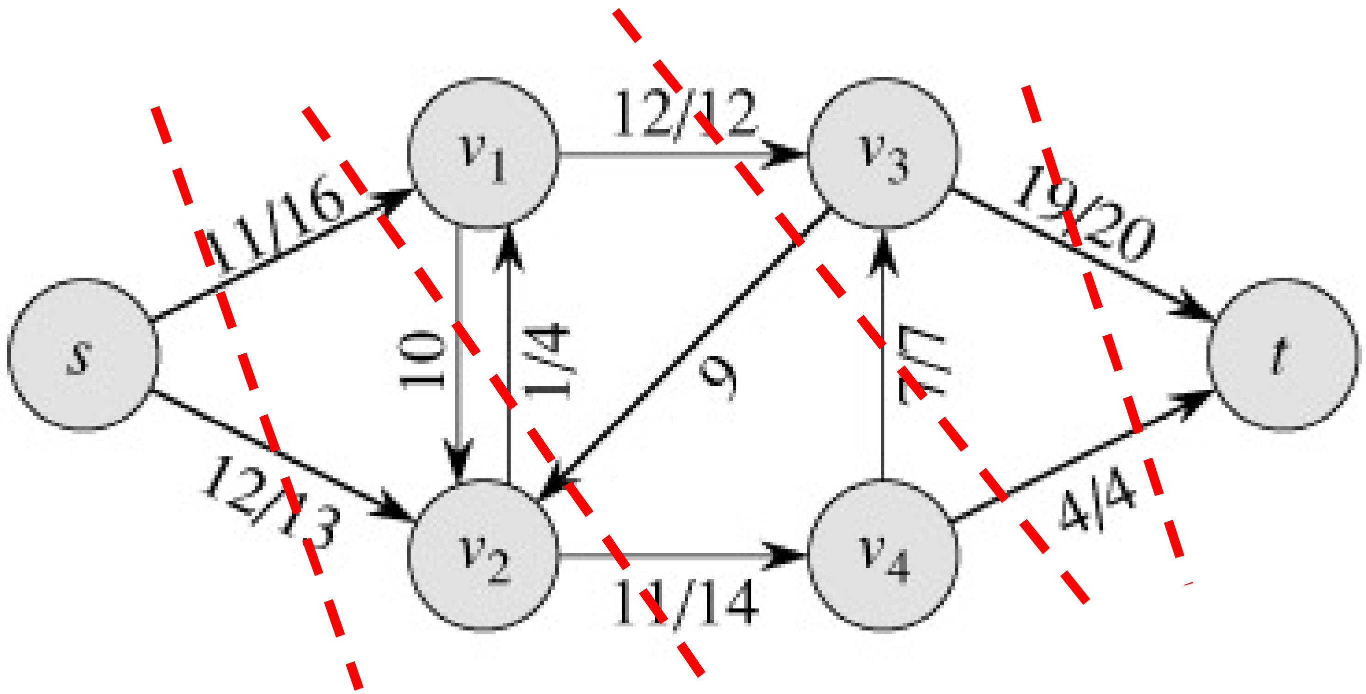
- A minimum-cut is a cut whose capacity is minimum over all cuts



$f(S, T) = 12 + 11 - 4 = \mathbf{19}$
$c(S, T) = 12 + 14 = \mathbf{26}$

Cuts of Flow Networks

- The net flow across any cut is the same and equal to the flow of the network |f|.



Cuts of Flow Networks

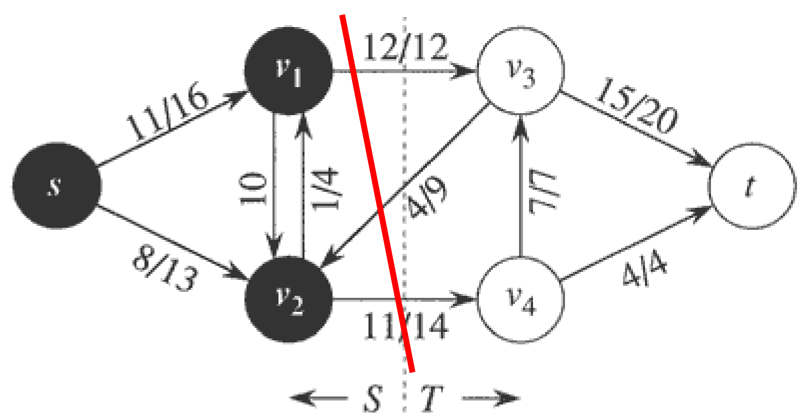
Lemma:

Let f be a flow in a network G with source s and sink t, and let (S, T) be a cut of G. Then the net flow across (S, T) is $f(S, T) = |f|$.

Corollary:

The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.

The value of any flow $\leq$ the capacity of any cut



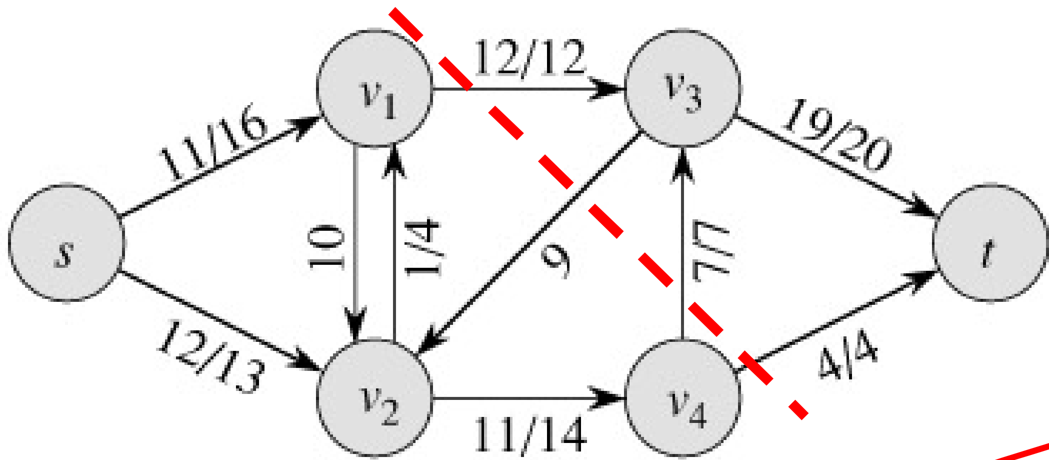
Max-flow Min-cut Theorem

- If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:
  (1) f is a maximum flow in G
  (2) The residual network G_f contains no augmenting paths
  (3) $|f| = c(S, T)$ for some cut (S, T) in G

- Proof:
  - (1) => (2)
  - (2) => (3)
  - (3) => (1)

The value of Maximum Flow =
the Capacity of Minimum Cut

Max-flow Min-cut Theorem

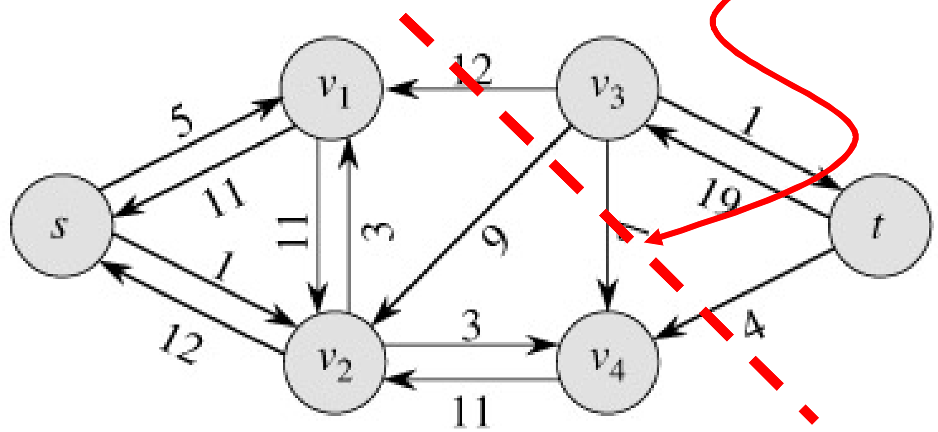


Resulting Flow = 23

No augmenting path:

Maxflow = 23

(e)



Residual Network

Ford-Fulkerson Algorithm

FORD-FULKERSON(G, s, t)

for each edge $(u, v) \in E[G]$ do

$$\left.\begin{array}{l} f[u, v] = 0 \\ f[v, u] = 0 \end{array}\right\} O(E)$$

?

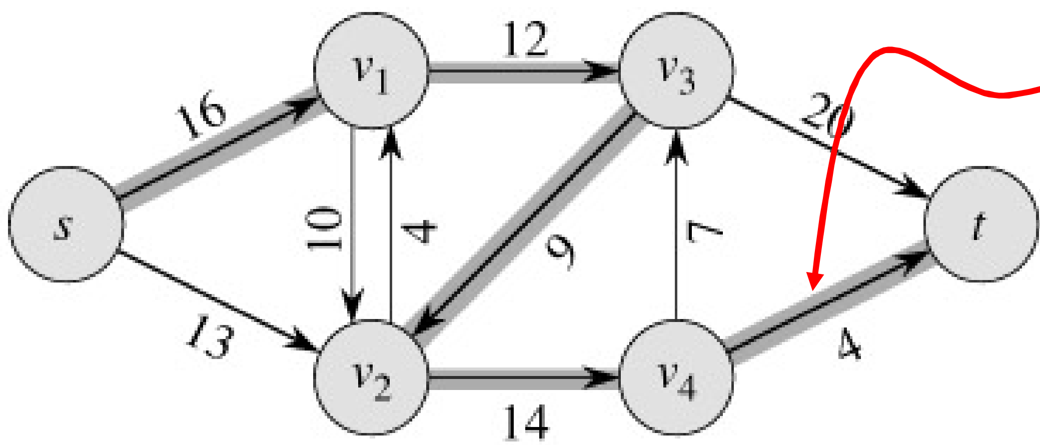
while there exists a path P from s to t in the residual network G_f do

$c_f(P) = \min\{c_f(u, v) : (u, v) \text{ is in } P\}$

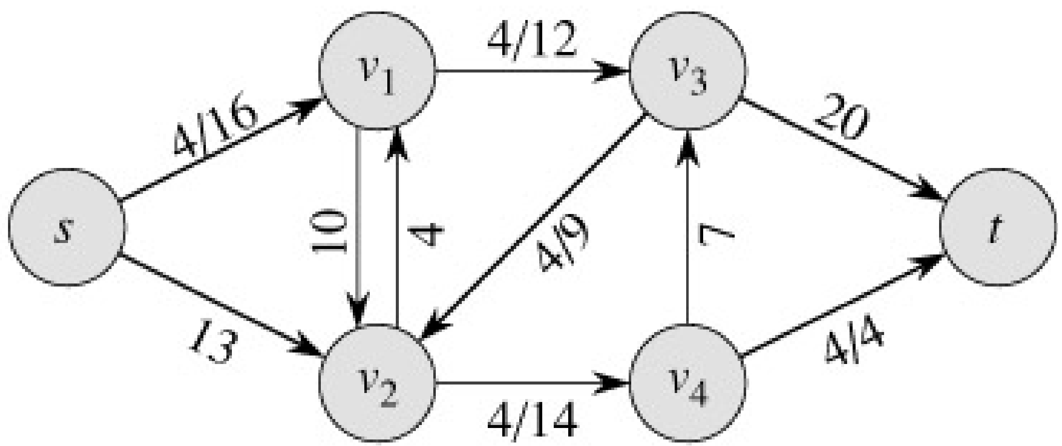
$$\left.\begin{array}{l} \text{for each edge } (u, v) \text{ in } P \text{ do} \\ \quad \text{if } (u, v) \in E[G] \\ \quad\quad f[u, v] = f[u, v] + c_f(P) \\ \quad \text{else } f[v, u] = f[v, u] - c_f(P) \end{array}\right\} O(E)$$

Ford-Fulkerson Algorithm: Example

augmenting path



Original Network

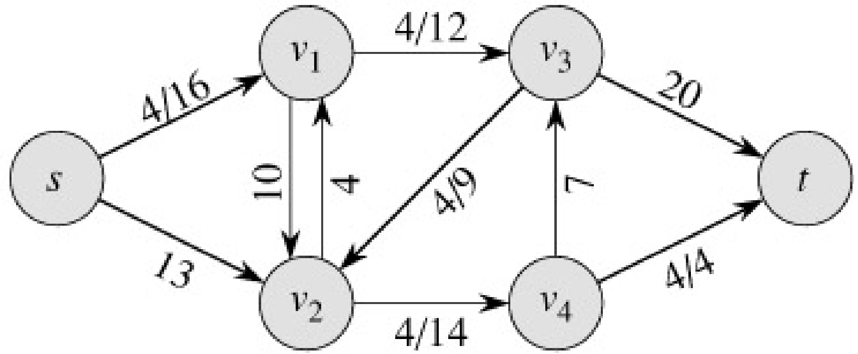


Flow Network

Resulting Flow = 4

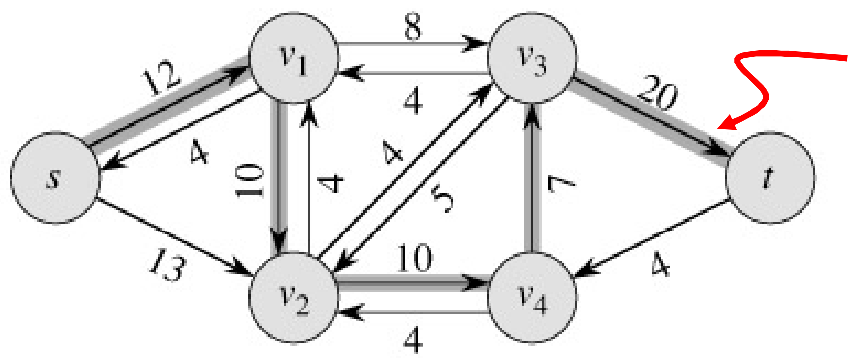
Ford-Fulkerson Algorithm: Example

Flow Network



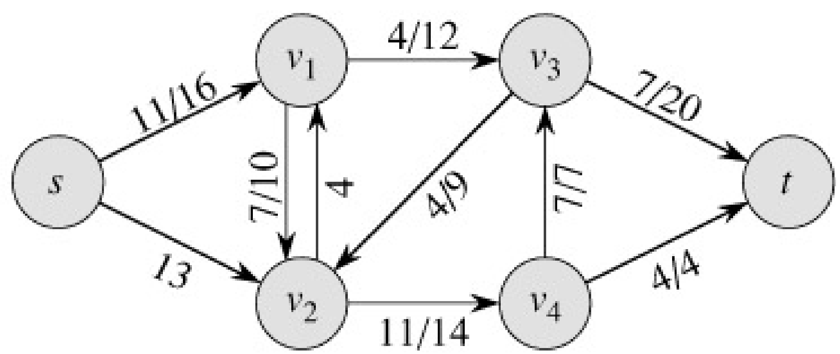
Resulting Flow = 4

Residual Network



augmenting path

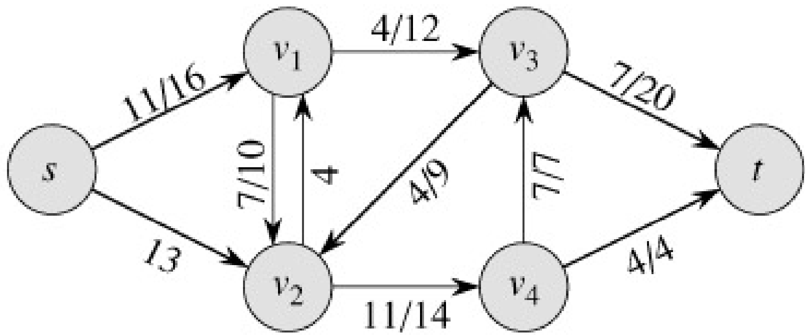
Flow Network



Resulting Flow = 11

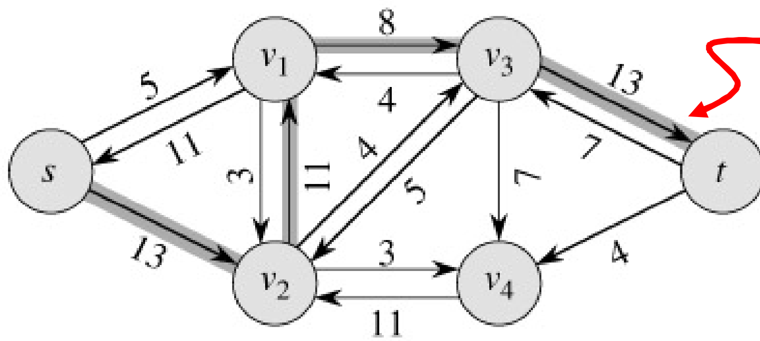
Ford-Fulkerson Algorithm: Example

Flow Network



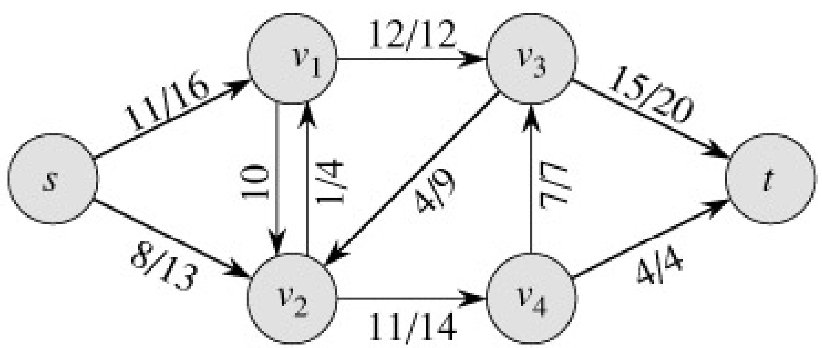
Resulting Flow = 11

Residual Network



augmenting path

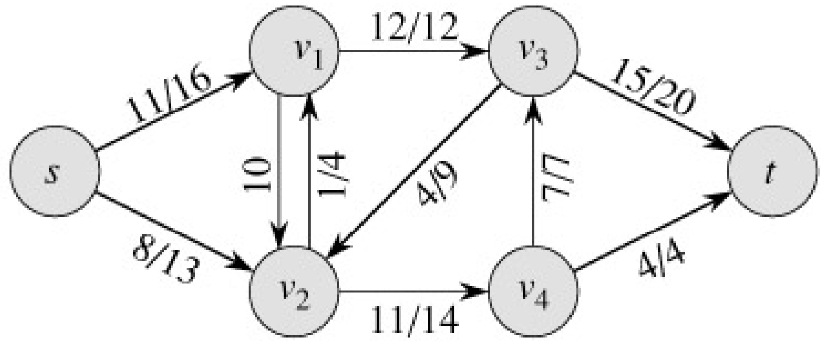
Flow Network



Resulting Flow =19

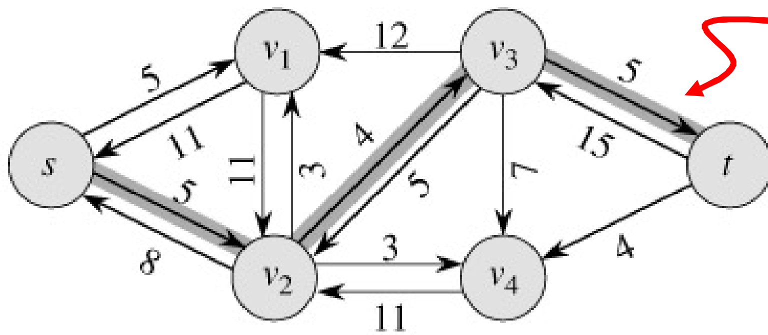
Ford-Fulkerson Algorithm: Example

Flow Network



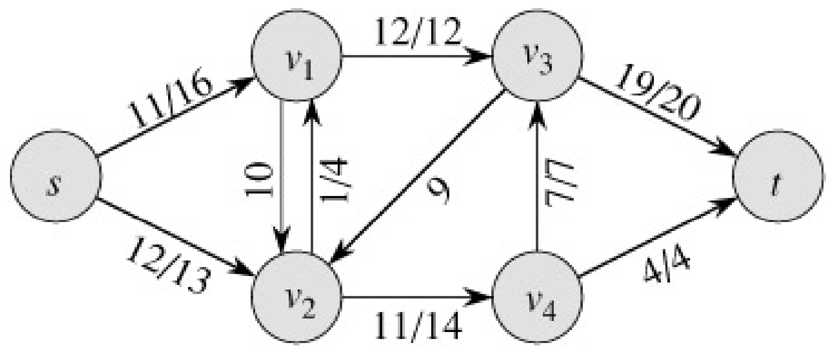
Resulting Flow =19

Residual Network



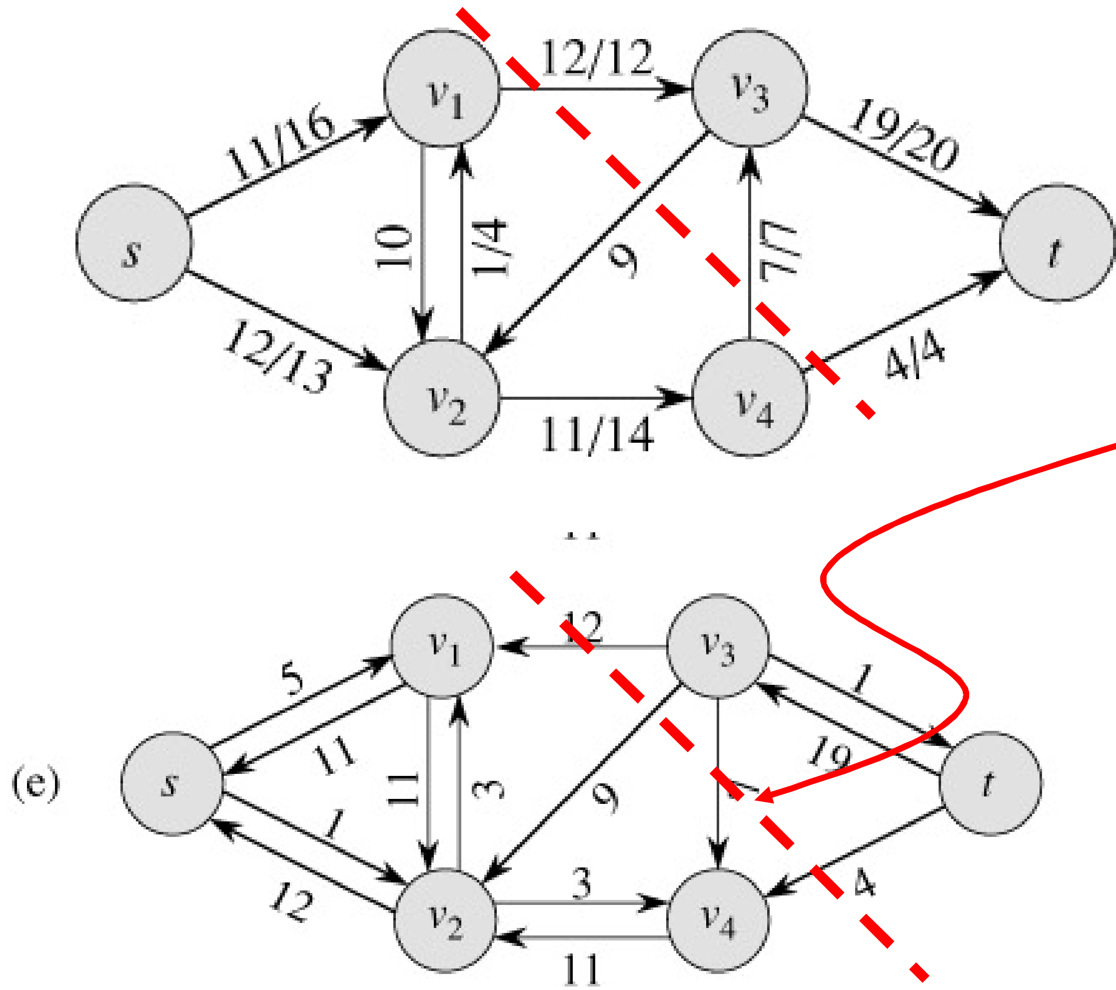
augmenting path

Flow Network



Resulting Flow =23

Ford-Fulkerson Algorithm: Example



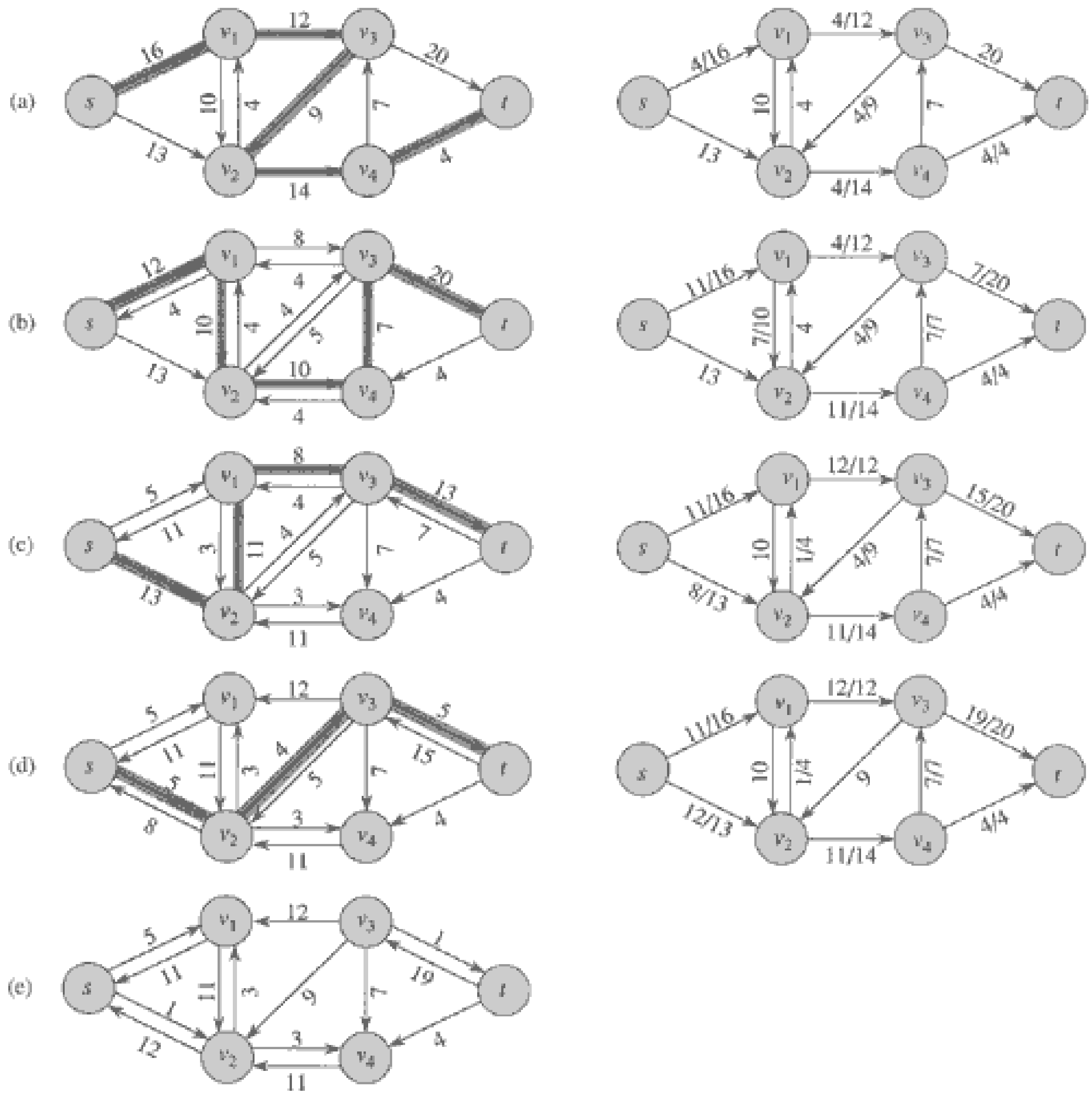
Resulting Flow = 23

No augmenting path:

Maxflow = 23

Residual Network

Ford-Fulkerson Algorithm: Example



Ford-Fulkerson Algorithm: Analysis

- Performance obviously
  - depends on the augmenting paths found at each iteration

- If edge capacities are integers (or, rational numbers [*apply an appropriate scaling transformation to make them all integral*]):
  - Then the algorithm returns max-flow
  - The algorithm runs in polynomial time

- If edge capacities are irrational numbers:
  - Then the algorithm might not even terminate
  - It need not even converge to the maximum value

Ford-Fulkerson Algorithm: Integral Capacities

```
FORD-FULKERSON(G, s, t)
1  for each edge (u, v) ∈ E[G]
2       do f[u, v] ← 0
3          f[v, u] ← 0
4  while there exists a path p from s to t in the residual network G_f
5       do c_f(p) ← min{c_f(u, v) : (u, v) is in p}
6          for each edge (u, v) in p
7              do f[u, v] ← f[u, v] + c_f(p)
8                 f[v, u] ← −f[u, v]
```

O(E)

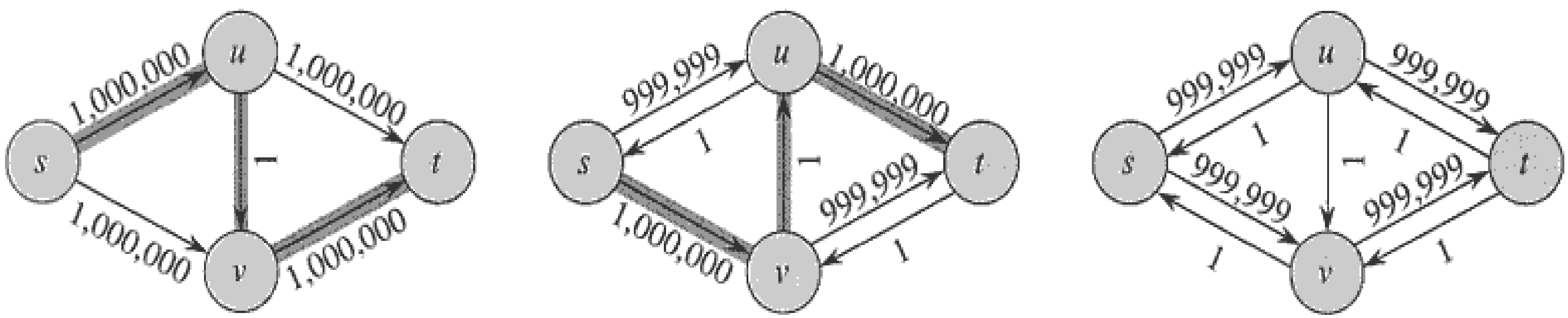
O(E)

If each $c(e)$ is an *integer*, then time complexity is $O(E\,|f^*|)$, where f^* is the maximum flow found by the algorithm

- Lines 1-3 take $O(E)$ time.
- The while loop of Line 4 is executed at most $|f^*|$ times, since the value of the flow increases by at least 1 at each iteration.
  - Each iteration of the while loop takes $O(E)$ time if either depth-first or breadth-first search is used to find a path in the residual network.
- Therefore, total time = $O(E + E\,|f^*|) = O(E\,|f^*|)$.

Ford-Fulkerson Algorithm: Integral Capacities

- Ford-Fulkerson algorithm runs in $O(E\,|f^*|)$ time, where f^* is the maximum flow found by the algorithm
- Not really polynomial in $|V|$ and $|E|$
  - Depends on $|f^*|$



Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Edmonds-Karp Algorithm

- A small fix to the Ford-Fulkerson algorithm makes it work in polynomial time.
- Select the augmenting path using **breadth-first search** on residual network.
- The augmenting path p is the shortest path from s to t in the residual network (treating all edge weights as 1).
- Runs in time $O(V E^2)$.

```
FORD-FULKERSON(G, s, t)
for each edge (u, v) ∈ E[G]
     do f[u, v] ← 0
        f[v, u] ← 0
while there exists a path p from s to t in the residual network G_f
     do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
        for each edge (u, v) in p
            do f[u, v] ← f[u, v] + c_f(p)
               f[v, u] ← −f[u, v]
```

Edmonds-Karp Algorithm

- Use the Ford-Fulkerson framework
- Implement the computation of augmenting path
  - by using breadth-first search
  - *i.e*, a shortest-linkage path (in no. of edges) from s to t in residual network
- Enable us to bound the time complexity
  - Mainly: the number of iterations

- Time complexity of Edmonds-Karp algorithm is $O(V E^2)$
  - The number of iterations is $O(V E)$
  - Each iteration needs $O(E)$

Edmonds-Karp Algorithm: Observations

- *Lemma*: If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source s and sink t, then for all vertices $v \in V - \{s, t\}$, the shortest distance $\delta_f(s, v)$ in the residual network G_f increases monotonically with each flow augmentation.

  Proof:

- *Theorem*: If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source s and sink t, then the total number of flow augmentations performed by the algorithm is $O(V E)$.

  Proof:

- Time complexity of Edmonds-Karp algorithm is $O(V E^2)$

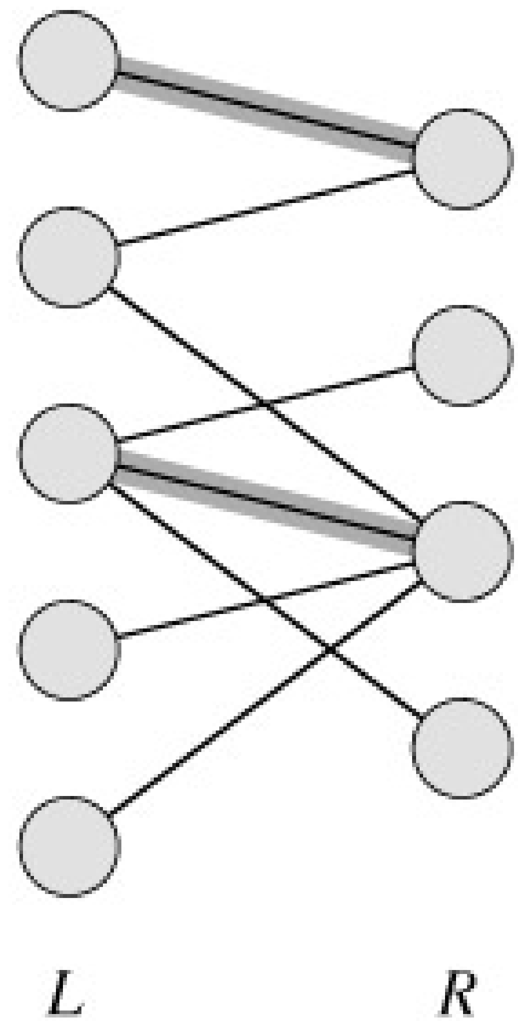
An Application of Max Flow:

Maximum Bipartite Matching

Bipartite Graph

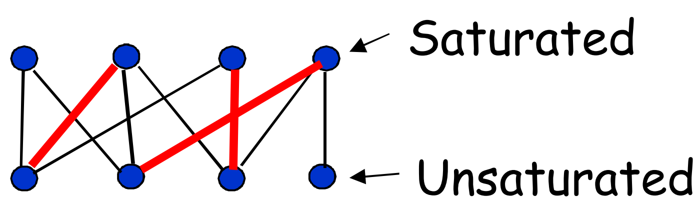
- A bipartite graph is a graph $G = (V, E)$ in which V can be partitioned into two parts L and R such that every edge in E is between a vertex in L and a vertex in R.

- e.g. vertices in L represent skilled workers and vertices in R represent jobs. An edge connects workers to jobs they can perform.

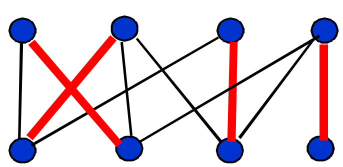


Matching

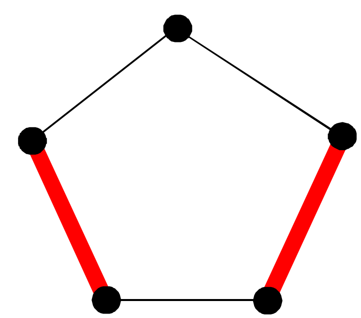
- A matching in a simple graph G is a set of edges with no shared endpoints.
- The vertices incident to the edges of a matching M are said to be saturated by M; the others are unsaturated.
- A perfect matching in a graph is a matching that saturates every vertex.
- A maximum matching is a matching of maximum size among all matchings in the graph.



A matching



Perfect Matching

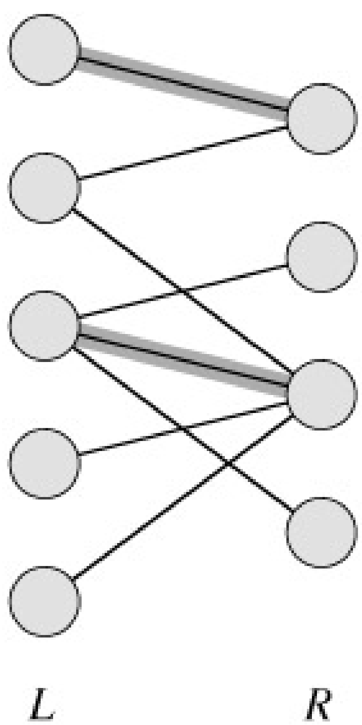


Maximum Matching
Not a perfect matching

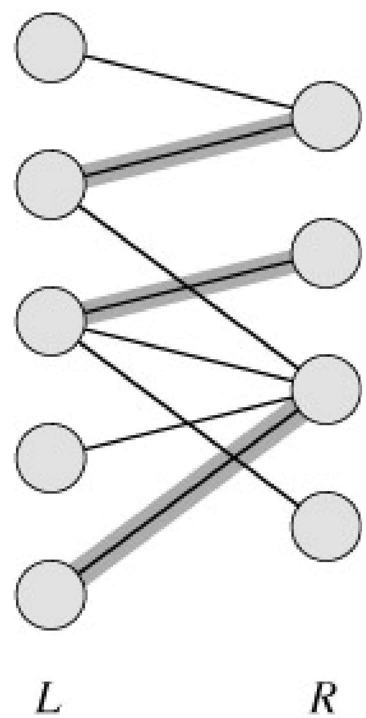
Maximum Matching

- A **maximum matching** is a matching of maximum cardinality (maximum number of edges).

not maximum

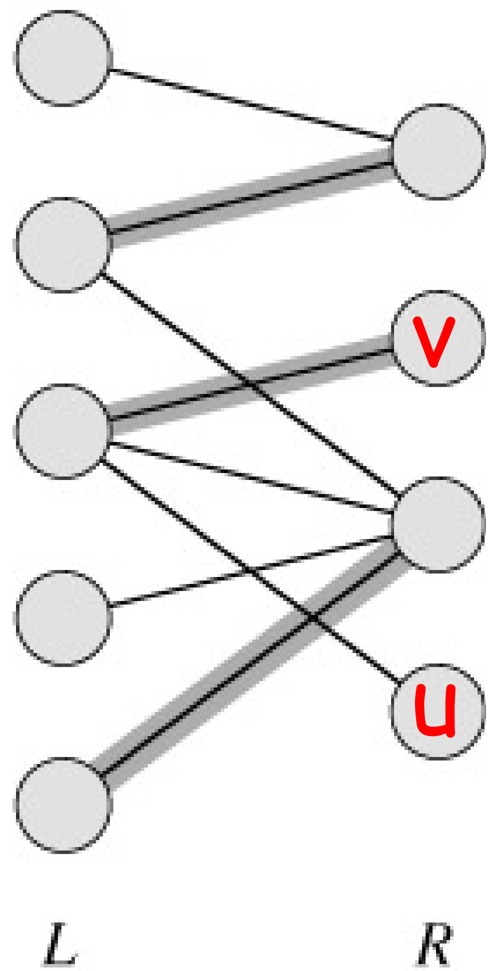


maximum



Maximum Matching

- No matching of cardinality 4, because only one of *v* and *u* can be matched.
- In the workers-jobs example a max-matching provides work for as many people as possible.

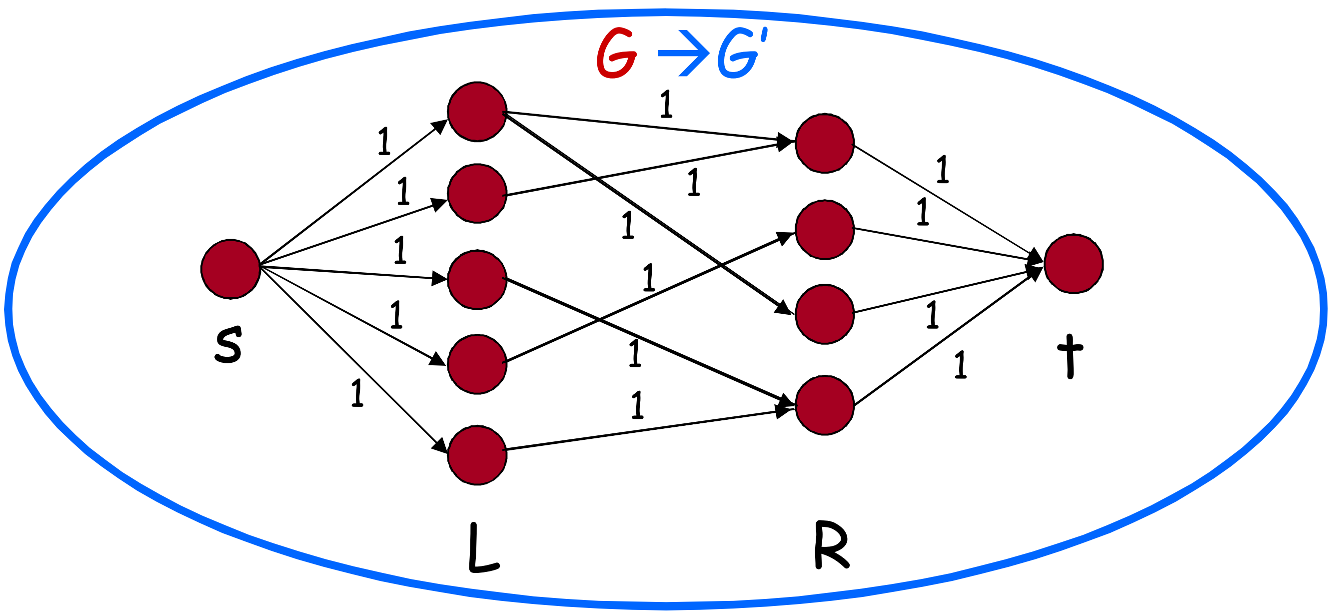


Solving the Maximum Bipartite Matching Problem

- Reduce the maximum bipartite matching problem on graph **G** to the max-flow problem on a corresponding flow network **G′**.
- Solve using Ford-Fulkerson algorithm.

Corresponding Flow Network

- To form the corresponding flow network **G'** of the bipartite graph **G**:
  - Add a source vertex s and edges from s to L.
  - Direct the edges in E from L to R.
  - Add a sink vertex t and edges from R to t.
  - Assign a capacity of 1 to all edges.
- Claim: max-flow in **G'** corresponds to a max-bipartite-matching on **G**.



Solving Bipartite Matching as Max Flow

Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cup R$.

Let $G' = (V', E')$ be its corresponding flow network.

If M is a matching in G,

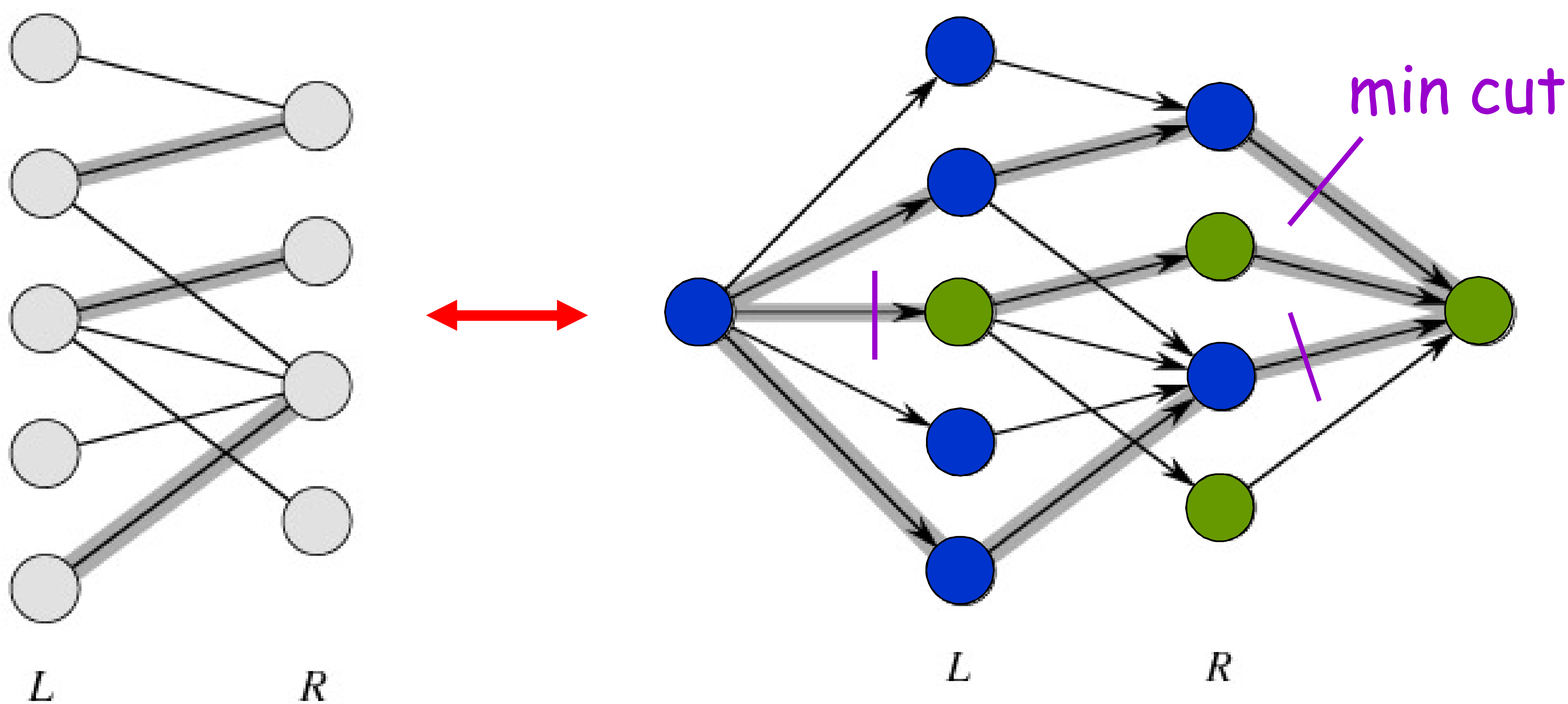
then there is an integer-valued flow f in G' with value $|f| = |M|$.

Conversely if f is an integer-valued flow in G',

then there is a matching M in G with cardinality $|M| = |f|$.

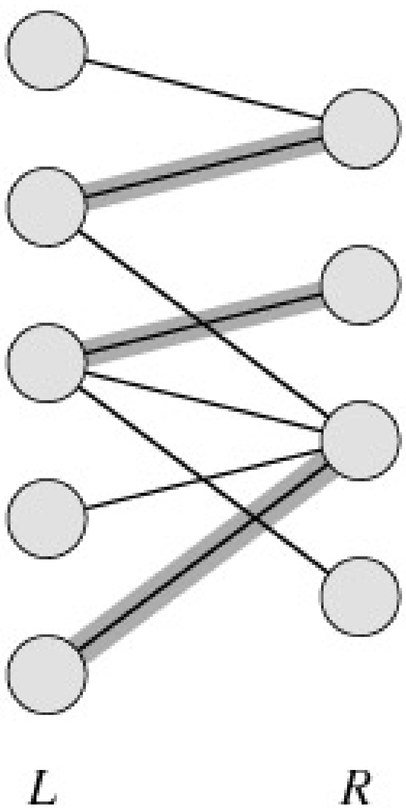
Thus $\max |M| = \max(\text{integer } |\text{f}|)$

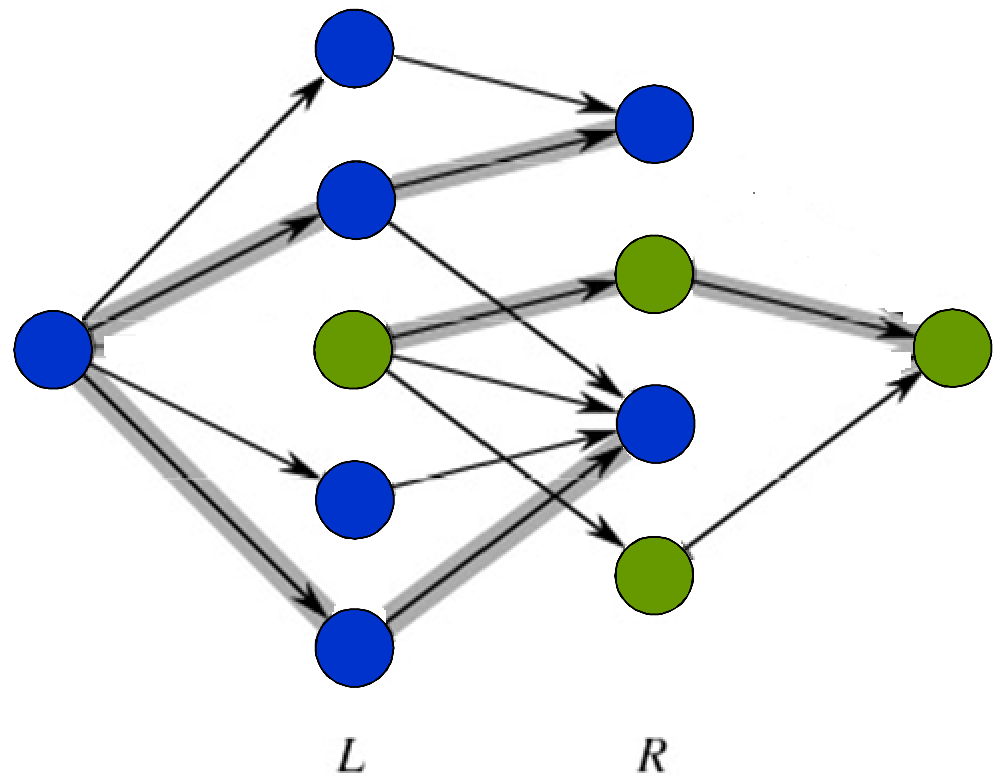
Example



|M| = 3 ←→ max flow =|f|= 3

Example





$|M| = 3 \quad \leftrightarrow \quad \text{max flow} = |f| = 3$

Conclusion

- Network flow algorithms allow us to find the maximum bipartite matching fairly easily.

- Similar techniques are applicable in other combinatorial design problems.