Network Flows (Cont...)

RAMESH K. JALLU

LEC-10

DT. 08/02//22

Lemma

- The new flow f' obtained is indeed a flow in G
- ***Proof***: We need to verify the *capacity constraint* and *flow conservation* properties
- Since f' differs from f only on edges of p, it is sufficient to check the properties only on these edges and vertices that lie on the path p
- Let (u, v) be any edge on p
  - If (u, v) is a forward edge, then $f'(u, v) = f(u, v) + b = f(u, v) + c_f(u, v) = c(u, v)$
  - If (u, v) is a back edge, then $f'(u, v) = f(u, v) - b = f(u, v) - f(u, v) = 0 \leq c(u, v)$
- Therefore, every edge on p satisfies capacity constraints

Proof (Cont …)

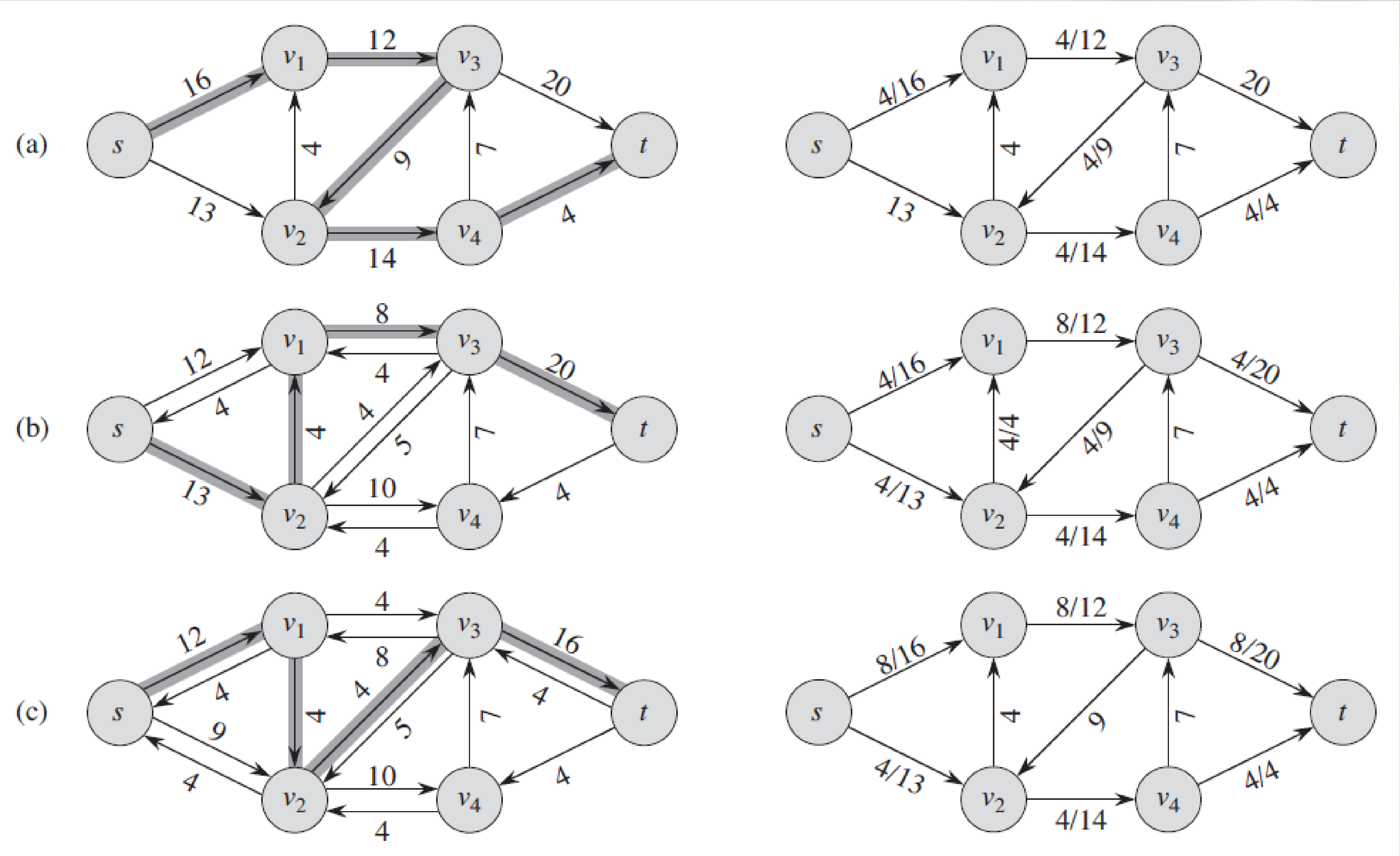
- Let v be any internal node on p
- We can verify that the change in the amount of flow entering v is the same as the change in the amount of flow exiting v
  - Since f satisfied the conservation condition at v, so must f'
- Technically, there are four cases to check, depending on whether the edge of p that enters v is a forward or backward edge, and whether the edge of P that exits v is a forward or backward edge
- However, each of these cases is easily worked out, and I leave them to you as an excursive

Algorithm(Recap)

```
FORD-FULKERSON-METHOD(G, s, t)
initialize flow f to 0
while there exists an augmenting path p in the residual network G_f
    augment flow f along p
return f
```

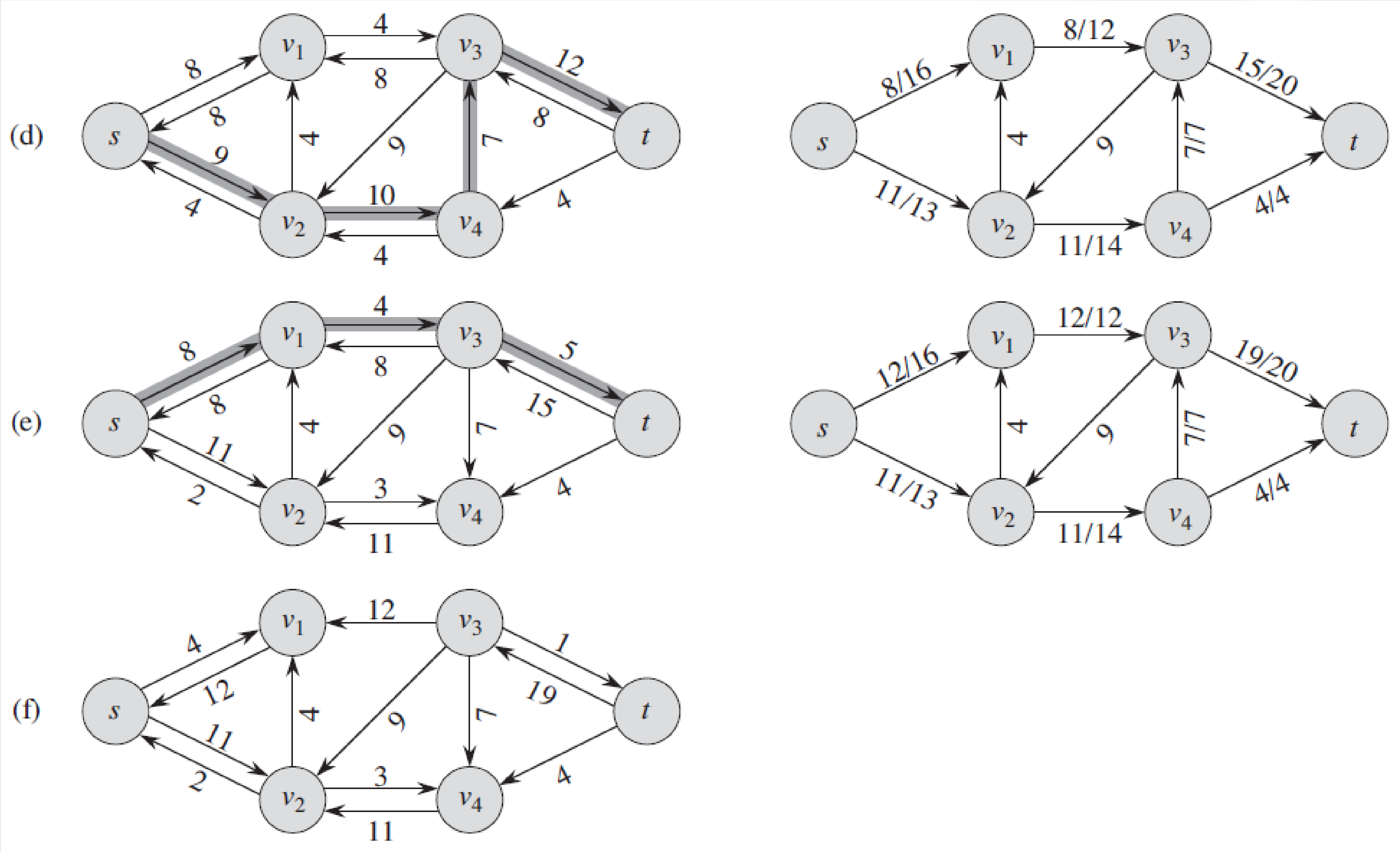
```
FORD-FULKERSON(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
while there exists a path p from s to t in the residual network G_f
    c_f(p) = min {c_f(u, v) : (u, v) is in p}
    for each edge (u, v) in p
        if (u, v) ∈ E
            (u, v).f = (u, v).f + c_f(p)
        else (v, u).f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

Example



(a)
s
v1
v2
v3
v4
t
16
13
12
4
9
14
7
20
4
4/16
13
4/12
4
4/9
4/14
7
20
4/4
(b)
12
4
13
8
4
4
4
5
10
4
7
20
4
4/16
4/13
8/12
4/4
4/9
4/14
7
4/20
4/4
(c)
12
4
9
4
4
8
4
4
5
10
4
7
16
4
4
8/16
4/13
8/12
4
9
4/14
7
8/20
4/4

Example



Analysis

- If all capacities are integers, then at every intermediate stage of the Ford-Fulkerson Algorithm, the flow values $\{f(u, v)\}$ and the residual capacities in G_f are integers
- ***Proof***: The statement is clearly true before any iterations of the while loop
- Now suppose it is true up to j iterations
- Since all residual capacities in G_f are integers, the value *bottleneck*(p, f) for the augmenting path found in iteration $j + 1$ will be an integer
- Thus the flow f will have integer values, and hence so will the capacities of the new residual graph

Cont ...

- The flow value ***strictly increases*** when we apply an augmentation
- ***Proof***: Let f be a flow in G, and let p be a simple s-t path in G_f. Then, $|f\,'| = |f| + bottleneck(p, f\,)$
- The first edge of p must be an edge out of s in the residual graph G_f
- Since the path is simple, it does not visit s again
- Since G has no edges entering s, the first edge must be a forward edge
- We increase the flow on this edge by $bottleneck(p, f\,)$, and we do not change the flow on any other edge incident to s
- Therefore, the value of $f\,'$ exceeds the value of f by $bottleneck(p, f\,)$

Termination of the algorithm

- Suppose, as previous, that all capacities in the flow network G are integers. Then, the algorithm terminates in at most C iterations, where C is the sum of the capacities of the edges leaving s
- ***Proof*****:** Observe that no flow in G can have value greater than C
- By previous observation, the flow value increases in each iteration; it increases by at least 1 in each iteration
- Since it starts with the value 0, and cannot go higher than C, the while loop in the algorithm can run for at most C iterations
- Therefore, the algorithm is guaranteed to terminate after a finite iterations

Running time

- The algorithm can be implemented to run in $O(C|E|)$ time under the assumptions that the capacities are integers
- *Proof*: The algorithm depends on the residual graph
- The residual graph G_f can be constructed from $G = (V, E)$ in $O(|V| + |E|)$
- We can maintain G_f as a adjacency list; we will have two linked lists for each node v, one containing the edges entering v, and one containing the edges leaving v
- To find an s-t path in G_f, we can use BFS or DFS which run in $O(|V| + |E|)$

Cont ...

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

- By our assumption, every node in G lies on some s-t path, and thus every node has at least one incident edge $|E| \geq |V| - 1$
- The *for* loop at line number 5 takes time proportional to the length of p, and which is at most $O(|V|)$
- Therefore, the total running time is $O(|E|) + C(O(|V| + |E|) + O(|V|) + O(|V|))$ $= O(C|E|)$

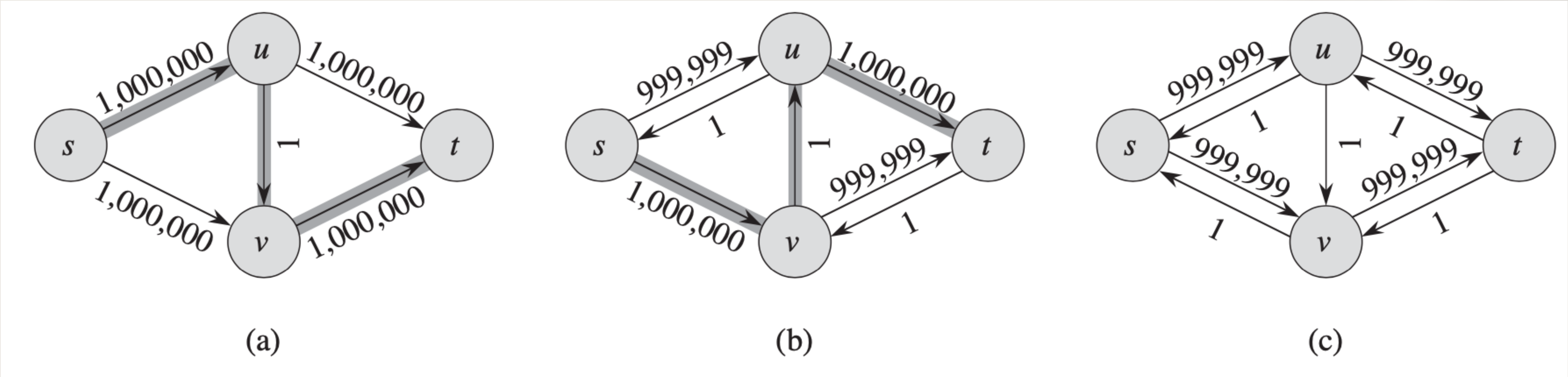
Remarks

- In practice, the maximum-flow problem often arises with integral capacities
- If the capacities are rational numbers, we can apply an appropriate scaling transformation to make them all integral
- If f^* denotes a maximum flow in the transformed network, then a straightforward implementation of FORD-FULKERSON executes the while loop of lines 3–8 at most $|f^*|$ times, since the flow value increases by at least one unit in each iteration
- The Ford-Fulkerson method might *fail to terminate* only if edge capacities are *irrational numbers*

Choosing good augmenting paths

- Recall that augmentation increases the value of the maximum flow by the bottleneck capacity of the selected path
- We saw that any way of choosing an augmenting path increases the value of the flow
- This led to a bound of C on the number of augmentations, where C is the sum of the capacities of the edges leaving s
- When C is not very large, this can be a reasonable bound; however, it is very weak when C is large
- To get a sense for how bad this bound can be, consider the example graph in the next slide

Example



u
s
v
t
1,000,000
1,000,000
1
1,000,000
1,000,000
(a)
u
s
v
t
999,999
1
1,000,000
1
1,000,000
999,999
1
(b)
u
s
v
t
999,999
1
999,999
1
1
999,999
1
999,999
1
(c)

- Thank you!