Introduction to Algorithms

Chapter 26 : Maximum Flow

Xiang-Yang Li and Haisheng Tan

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2024

Outline of Topics

Flow Networks and Flows

A **flow network** $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.We further require that if E contains an edge (u, v),then there is no edge (v, u) in the reverse direction. If $(u, v) \notin E$, then for convenience we define $c(u, v) = 0$, and we disallow self-loops.
We distinguish two vertices in a flow network: a source s and a sink t. For convenience, we assume that each vertex lies on some path from the source to the sink. That is, for each vertex $v \in V$, the flow network contains a path $s \rightsquigarrow v \rightsquigarrow t$. The graph is therefore connected and, since each vertex other than s has at least one entering edge, $|E| \geq |V| - 1$.

Flow Networks and Flows

A **flow** in G is a real-valued function $f : V \times V \to \mathbb{R}$ that is satisfies the following two properties:

- **Capacity constraint:** For all $u, v \in V$, we require $0 \leq f(u,v) \leq c(u,v)$.
- **Flow conservation:** For all $u \in V - \{s,t\}$, we require

$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$$

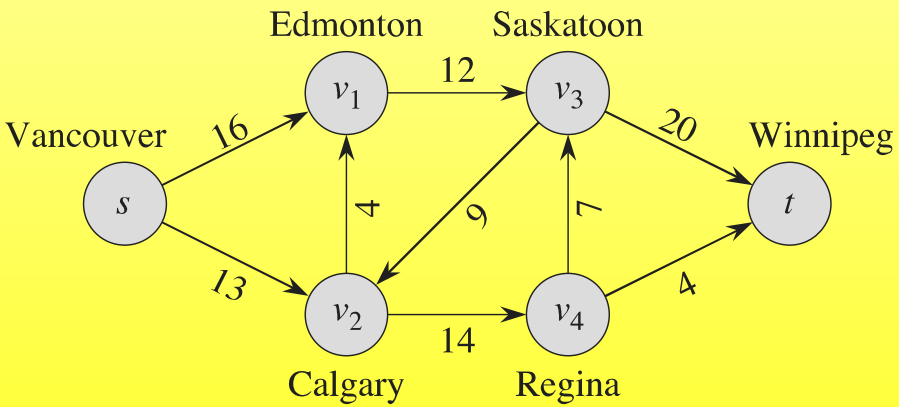
When $(u,v) \notin E$, there can be no flow from u to v, and $f(u,v) = 0$.

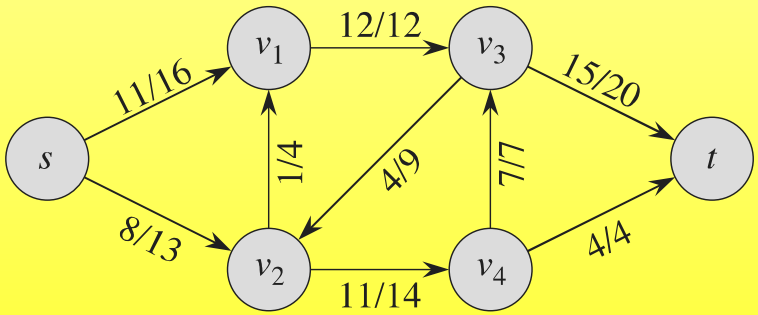
Flow Networks and Flows

We call the nonnegative quantity $f(u,v)$ the flow from vertex u to vertex v. The **value** $|f|$ of a flow f is defined as $|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$, that is, the total flow out of the source minus the flow into the source. Typically, a flow network will not have any edges into the source, and the flow into the source, given by the summation $\sum_{v \in V} f(v,s)$, will be 0. We include it, however, because when we introduce residual networks later in this chapter, the flow into the source will become significant. In the **maximum-flow problem**, we are given a flow network G with source s and sink t, and we wish to find a flow of maximum value.

Flow Networks and Flows

Left: A flow Network G
Right: A flow in G



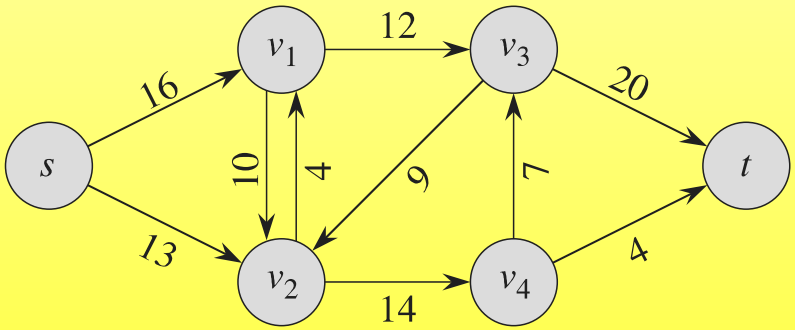


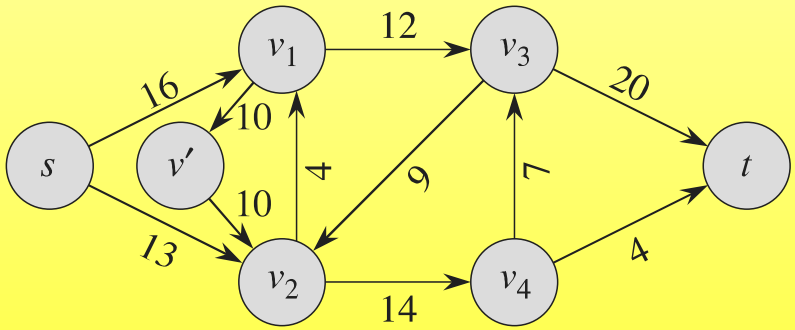
Antiparallel Edge

Suppose that the trucking firm offered Lucky Puck the opportunity to lease space for 10 crates in trucks going from Edmonton to Calgary. It would seem natural to add this opportunity to our graph. This network suffers from one problem, however: it violates our original assumption that if an edge $(v_1, v_2) \in E$, then $(v_2, v_1) \notin E$. We call the two edges (v_1, v_2) and (v_2, v_1) **antiparallel**.

We must transform the network into an equivalent one containing no antiparallel edges. We choose one of the two antiparallel edges, in this case (v_1, v_2) , and split it by adding a new vertex v' and replacing edge (v_1, v_2) with the pair of edges (v_1, v') and (v', v_2)

Antiparallel Edge



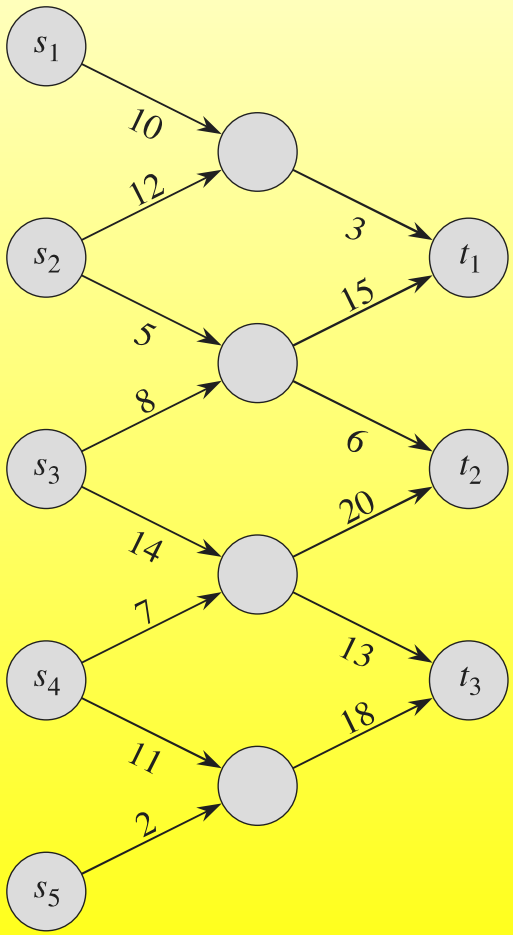


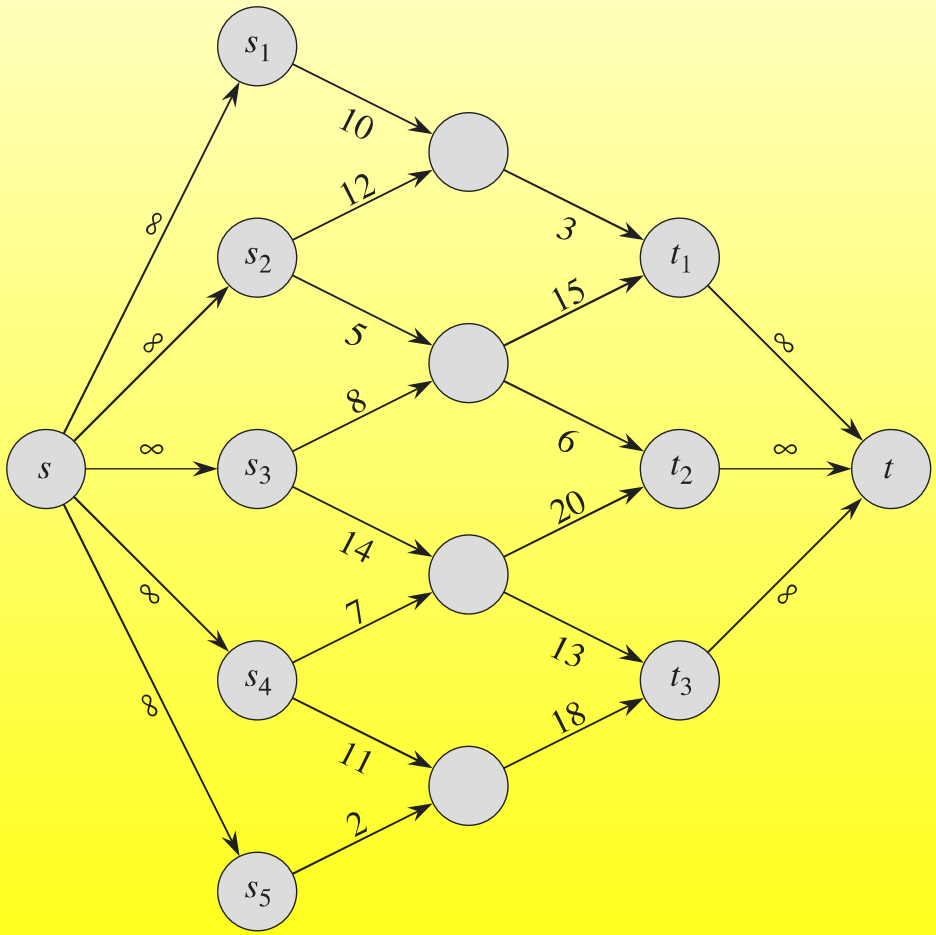
Networks with Multiple Sources and Sinks

A maximum-flow problem may have several sources and sinks, rather than just one of each. The Lucky Puck Company, for example, might actually have a set of m factories $\{S_1, S_2, \ldots, S_m\}$ and a set of n warehouses $\{t_1, t_2, \ldots, t_n\}$. Fortunately, this problem is no harder than ordinary maximum flow.

We add a **supersource** s and add a directed edge (s, s_i) with capacity $c(s, s_i) = \infty$ for each $i = 1, 2, \ldots, m$. We also create a new **supersink** t and add a directed edge (t_i, t) with capacity $c(t_i, t) = \infty$ for each $i = 1, 2, \ldots, n$.

Networks with Multiple Sources and Sinks





The Ford-Fulkerson Method

The Ford-Fulkerson method iteratively increases the value of the flow. We start with $f(u,v) = 0$ for all $u, v \in V$, giving an initial flow of value 0. At each iteration, we increase the flow value in G by finding an "augmenting path" in an associated "residual network" G_f. Once we know the edges of an augmenting path in G_f , we can easily identify specific edges in G for which we can change the flow so that we **increase** the value of the flow. Although each iteration of the Ford-Fulkerson method increases the value of the flow, we shall see that the flow on any particular edge of G may **increase** or **decrease**.

The Ford-Fulkerson Method

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

In order to implement and analyze the Ford-Fulkerson method, we need to introduce several additional concepts.

Residual Networks

Intuitively, given a flow network G and a flow f, the residual network G_f consists of edges with capacities that represent how we can change the flow on edges of G. An edge of the flow network can admit an amount of additional flow equal to the edge's capacity minus the flow on that edge. If that value is positive, we place that edge into G_f with a "residual capacity" of $c_f(u,v) = c(u,v) - f(u,v)$. The only edges of G that are in G_f are those that can admit more flow; those edges (u,v) whose flow equals their capacity have $c_f(u,v) = 0$, and they are not in G_f.

Residual Networks

The residual network G_f may also contain edges that are not in G, however. As an algorithm manipulates the flow, with the goal of increasing the total flow, it might need to decrease the flow on a particular edge.

In order to represent a possible decrease of a positive flow $f(u,v)$ on an edge in G, we place an edge (v,u) into G_f with residual capacity $c_f(v,u) = f(u,v)$. These reverse edges in the residual network allow an algorithm to *send back* flow it has already sent along an edge.

Residual Networks

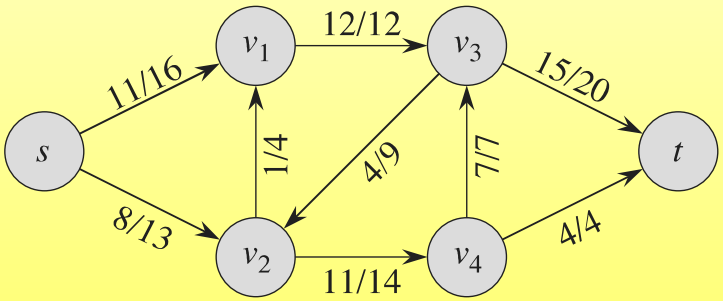
More formally, suppose that we have a flow network $G = (V, E)$ with source s and sink t. Let f be a flow in G, and consider a pair of vertices $u, v \in V$. We define the **residual capacity** $c_f(u, v)$ by

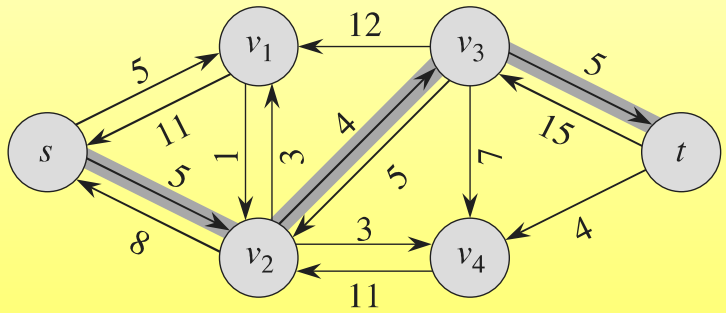
$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

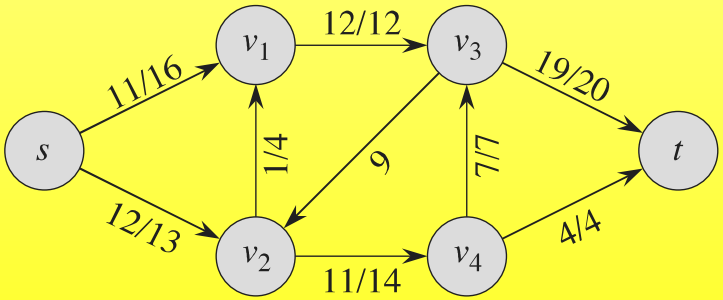
Given a flow network $G = (V, E)$ and a flow f, the **residual network** of G induced by f is $G_f = (V, E_f)$, where

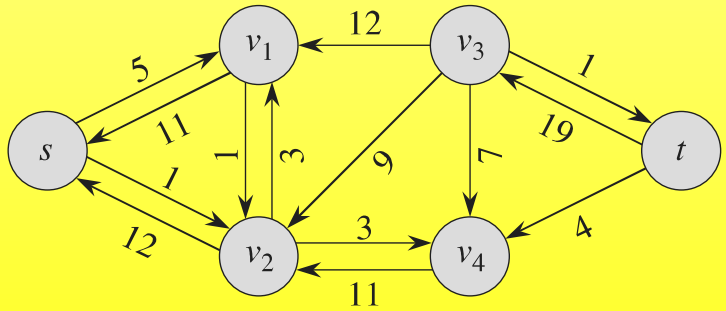
$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

Networks with Multiple Sources and Sinks









Residual Networks

The edges in E_f are either edges in E or their reversals, and thus $|E_f| \leq 2|E|$.

Observe that the residual network G_f is similar to a flow network with capacities given by c_f. It does not satisfy our definition of a flow network because it may contain both an edge (u,v) and its reversal (v,u). Other than this difference, a residual network has the same properties as a flow network, and we can define a flow in the residual network as one that satisfies the definition of a flow, but with respect to capacities c_f in the network G_f.

Residual Networks

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 the corresponding residual network G_f, we define $f \uparrow f'$, the **augmentation** of flow f by f', to be a function from $V \times V$ to $\mathbb{R}$, defined by

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

Residual Networks

Lemma 26.1: 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 function $f \uparrow f'$ is a flow in G with value $|f \uparrow f'| = |f| + |f'|$.

Augmenting Paths

Given a flow network $G = (V, E)$ and a flow f , an augmenting path p is a simple path from s to t in the residual network G_f. By the definition of the residual network, we may increase the flow on an edge (u, v) of an augmenting path by up to $c_f(u, v)$ without violating the capacity constraint on whichever of (u, v) and (v, u) is in the original flow network G.
We call the maximum amount by which we can increase the flow on each edge in an augmenting path p the residual capacity of p, given by

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

Augmenting Paths

Lemma 26.2: Let $G = (V, E)$ be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Define a function $f_p : V \times V \to \mathbb{R}$ by

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

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

Augmenting Paths

Corollary 26.3: Let $G = (V, E)$ be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Suppose that we augment f by f_p. Then the function $f \uparrow f_p$ is a flow in G with value $|f \uparrow f_p| = |f| + |f_p| > |f|$.

Cuts of Flow Networks

A **cut** (S,T) of flow network $G=(V,E)$ is a partition of V into S and $T=V-S$ such that $s\in S$ and $t\in T$. If f is a flow, then the **netflow** $f(S,T)$ across the cut (S,T) is defined to be

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

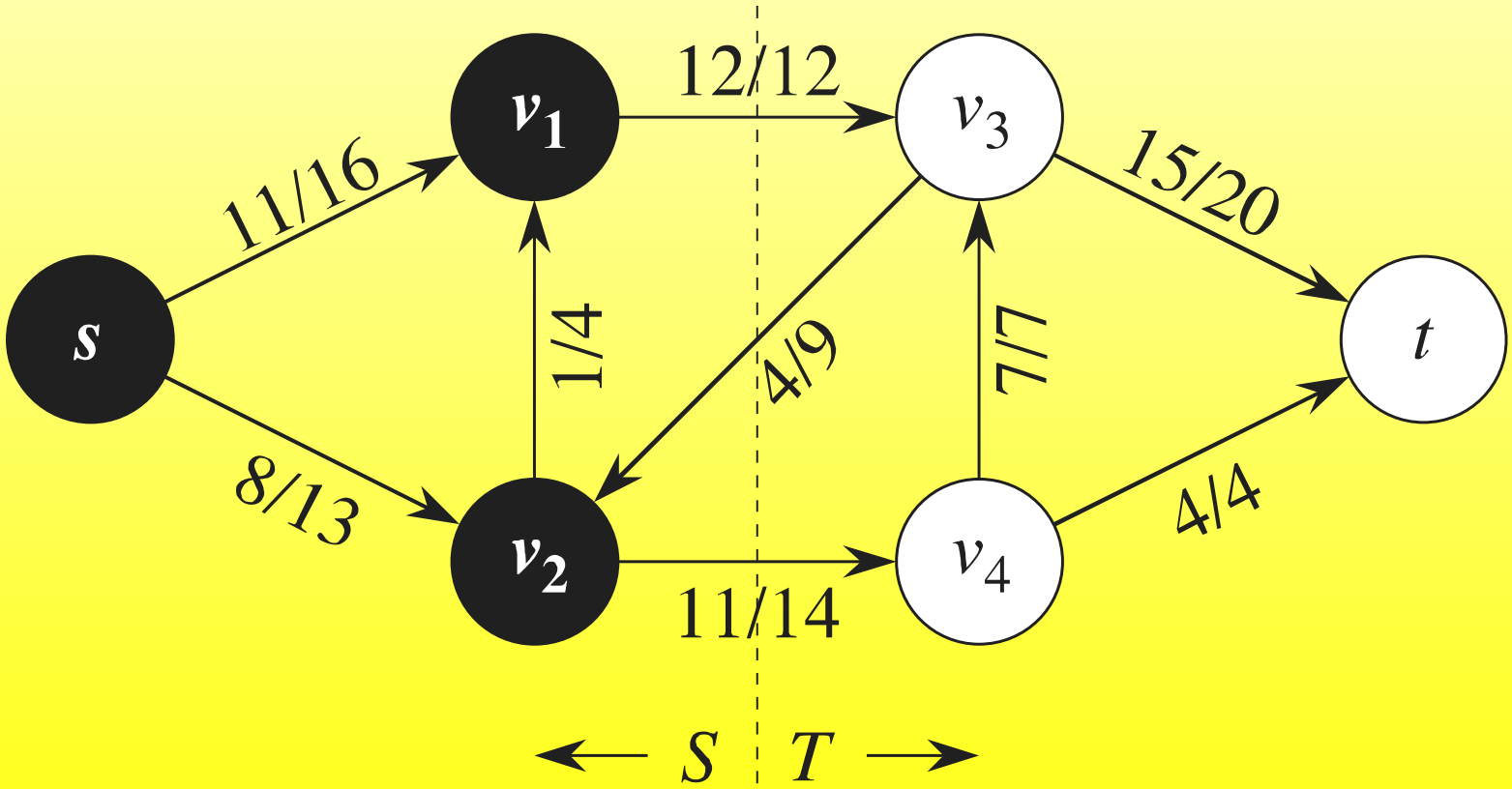
The **capacity** of the cut (S,T) is

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

A **minimum cut** of a network is a cut whose capacity is minimum over all cuts of the network.

Cuts of Flow Networks

$$f(S,T) = 12 - 4 + 11 = 19; \qquad c(S,T) = 12 + 14 = 26.$$



Cuts of Flow Networks

Lemma 26.4: Let f be a flow in a flow network G with source s and sink t, and let (S,T) be any cut of G. Then the net flow across (S,T) is $f(S,T) = |f|$.

Corollary 26.5: The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.

Theorem 26.6 (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:

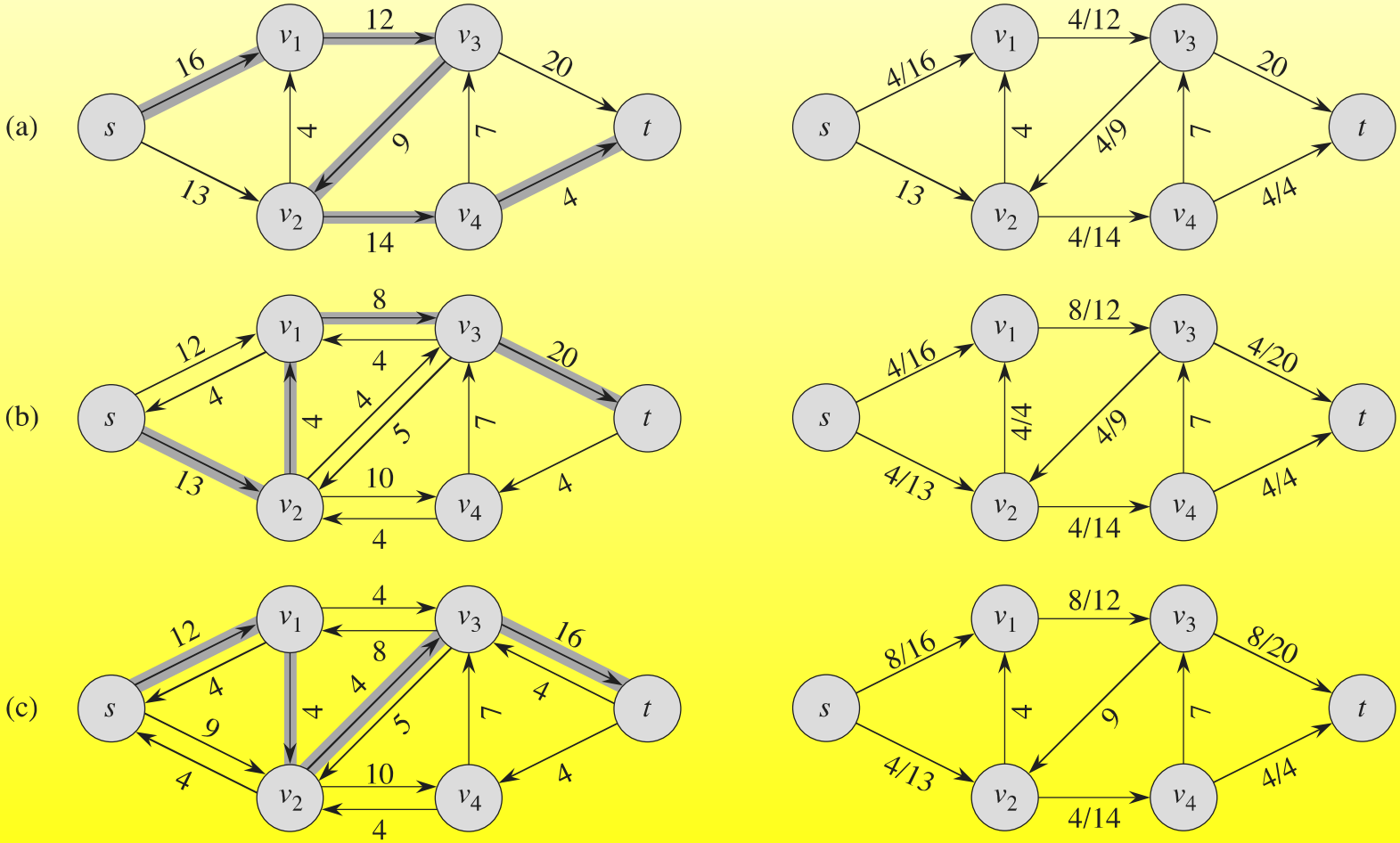
- f is a maximum flow in G.
- The residual network G_f contains no augmenting paths.
- $|f| = c(S,T)$ for some cut (S,T) of G.

The basic Ford-Fulkerson Algorithm

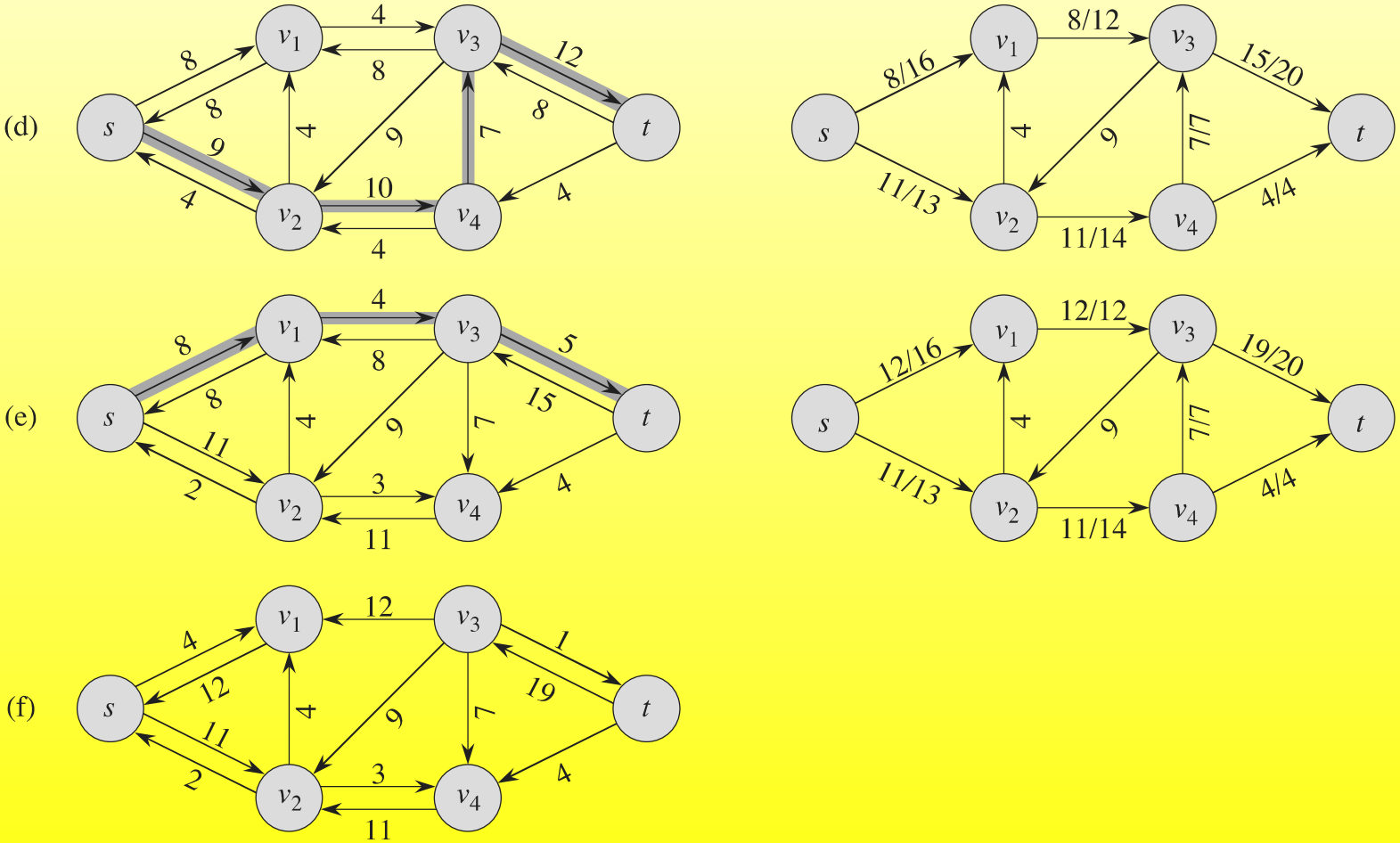
FORD-FULKERSON(G,s,t)

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

Example of Ford-Fulkerson



Example of Ford-Fulkerson



Analysis of Ford-Fulkerson

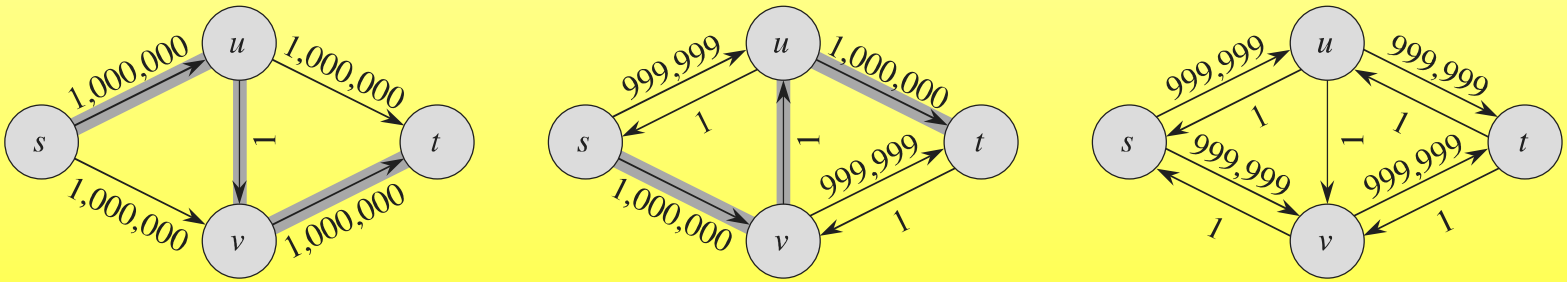
The running time of FORD-FULKERSON depends on how the augmenting path p in line 3 is determined.
If it is chosen poorly, the algorithm might not even terminate.
If the augmenting path is chosen by using a breadth-first search, the algorithm runs in polynomial time.
Most often in practice, the maximum-flow problem arises with integral capacities. (If the capacities are rational numbers, we can apply an appropriate scaling transformation to make them all integral.)

Analysis of Ford-Fulkerson

Lines 1-2 take time $\Theta(E)$.
The **while** loop of lines 3-8 is executed at most $|f^*|$ times, since the flow value increases by at least one unit in each iteration.
Assumed a data structure corresponding a directed graph $G' = (V, E')$, where $E' = \{(u,v) : (u,v) \in E \text{ or } (v,u) \in E\}$
Given a flow f on G, the edges in the residual network G_f consist of all edges (u,v) of G' such that $c_f(u,v) > 0$.
The time to find a path in a residual network is therefore $O(V+E') = O(E)$ if we use either depth-first search or breadth-first search.
Thus the total running time of FORD-FULKERSON is $O(E|f^*|)$

Example of Ford-Fulkerson

the optimal flow value $|f^*|$ is large:



The Edmonds-Karp Algorithm

We can improve the bound on FORD-FULKERSON by finding the augmenting path p in line 3 with a breadth-first search. That is, we choose the augmenting path as a shortest path from s to t in the residual network, where each edge has unit distance (weight).
We call the Ford-Fulkerson method so implemented the Edmonds-Karp algorithm. We now prove that the **Edmonds-Karp algorithm** runs in $O(VE^2)$ time.

The Edmonds-Karp Algorithm

Lemma 26.7: 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-path distance $\delta_f(s, v)$ in the residual network G_f increases monotonically with each flow augmentation.

Theorem 26.8: 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 $O(VE)$.

The Edmonds-Karp Algorithm

Because we can implement each iteration of **Ford-Fulkerson** in $O(E)$ time when we find the augmenting path by breadth-first search, the total running time of the Edmonds-Karp algorithm is $O(VE^2)$.

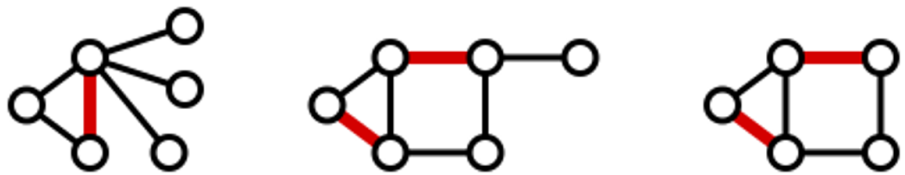
We shall see that push-relabel algorithms can yield even better bounds. The algorithm of Section 26.4 gives a method for achieving an $O(V^2E)$ running time, which forms the basis for the $O(V^3)$-time algorithm of Section 26.5.

Maximum Bipartite Matching

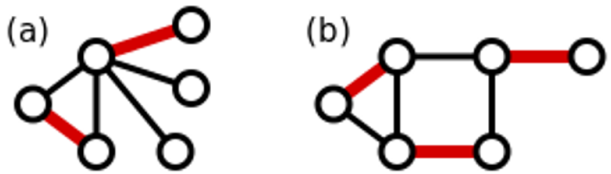
This section presents one such problem: finding a maximum matching in a bipartite graph.
In order to solve this problem, we shall take advantage of an integrality property provided by the Ford-Fulkerson method. We shall also see how to use the Ford-Fulkerson method to solve the maximum-bipartite-matching problem on a graph $G = (V, E)$

Matching

(Matching). *Given a graph $G = (V, E)$ a* matching *is a subset of edges $M \subseteq E$, such that no two edges in M are adjacent (i.e., where no node is adjacent to two edges in the matching). A matching is* maximal *if no edge can be added without violating the above constraint. A matching of maximum cardinality is called* maximum. *A matching is called* perfect *if each node is adjacent to an edge in the matching.*



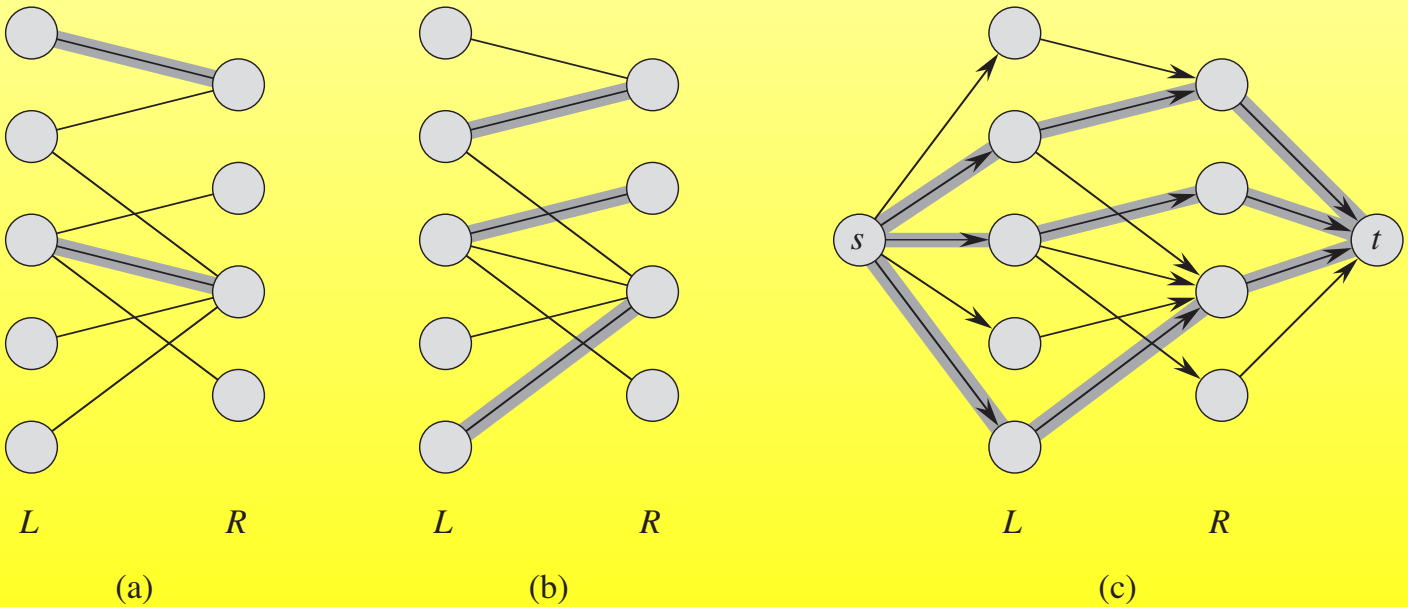
maximal



maximum

Bipartite Graph

The vertex set can be partitioned into $V = L \cup R$, where L and R are disjoint and all edges in E go between L and R.
We assume that every vertex in V has at least one incident edge.



Finding a maximum bipartite matching

Use the Ford-Fulkerson method to find a maximum matching in an undirected bipartite graph $G = (V, E)$ in time polynomial in $|V|$ and $|E|$.

The trick is to construct a flow network in which flows correspond to matchings, as shown in Figure (c).

We define the **corresponding flow network** $G' = (V', E')$ for the bipartite graph G:

source s and sink t be new vertices not in V, $V' = V \cup \{s, t\}$

If $V = L \cup U$, the directed edges of G' is

$$E' = \{(s,u) : u \in L\} \cup \{(u,v) : (u,v) \in E\} \cup \{(v,t) : v \in R\}$$

The maximum-bipartite-matching problem

To complete the construction, we assign unit capacity to each edge in E'. Since each vertex in V has at least one incident edge, $|E| \geq |V|/2$: $|E| \leq |E'| = |E| + |V| \leq 3|E|$,and so $|E'| = \Theta(E)$
The following lemma shows that a matching in G corresponds directly to a flow in $G's$ corresponding flow network G'.

Lemma 26.9

Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cup R$, and let $G' = (V', E')$ be its corresponding flow network. If M is a matching in G, then there is an integervalued 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 $|M| = |f|$.

Theorem 26.10 (Integrality theorem)

If the capacity function c takes on only integral values, then the maximum flow f produced by the Ford-Fulkerson method has the property that $|f|$ is an integer. Moreover, for all vertices u and v, the value of $f(u,v)$ is an integer.

Corollary 26.11

The cardinality of a maximum matching M in a bipartite graph G equals the value of a maximum flow f in its corresponding flow network G'.

Maximum bipartite matching

Given a bipartite undirected graph G, we can find a maximum matching by the following steps:

- creating the flow network G'
- running the Ford-Fulkerson method, and directly obtaining a maximum matching M from the integer-valued maximum flow f found.

Since any matching in a bipartite graph has cardinality at most $\min(L,R) = O(V)$, the value of the maximum flow in G' is $O(V)$
So we can find a maximum matching in a bipartite graph in time $O(VE') = O(VE)$ (since $|E'| = \Theta(E)$)