Lecture 17 Maximum Flow

MCS/CS 401 Computer Algorithms I — July 27, 2022
Instructor: Danko Adrovic

Lecture Outline

- Maximum Flow
  - Flow networks and flows
  - Dealing with multiple sources and sinks, and antiparallel edges
  - The Ford-Fulkerson method
  - Residual networks
  - Augmenting paths
  - Cuts of flow networks
  - The basic Ford-Fulkerson algorithm
  - Analysis of Ford-Fulkerson algorithm

1 Maximum Flow

Practical motivation to study maximum-flow problems comes from the need to determine the the maximum amount of some material that can be moved through a network of, for example, pipes, or roads, or even data lines. Our main way to represent problems of these types is going to use weighted, directed graphs in which vertices represent locations and weighted edges the distances between adjacent locations. In addition to the notion of single source s, we will introduce an additional concept, namely that of sink t.

Specifically, in the maximum-flow problem, we wish to compute the greatest rate at which we can move material from the source to the sink without violating any capacity constraints. It is one of the simplest problems concerning flow networks and, as we will see in this lecture, this problem can be solved by efficient algorithms, particularly, the Ford-Fulkerson algorithm.

1.1 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 additionally require that if E contains an edge (u, v), then there is no edge (v, u) in the reverse direction (there is a way to get around this restriction, however). 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, $|E| \geq |V| - 1$. See the figure below as an illustration.

Formally, we define the flows in the following way. Let $G = (V, E)$ be a flow network with a capacity function c. Let s be a source of the network, and let t be a sink. A **flow** in G is a real-valued function $f : V \times V \to \mathbb{R}$ that 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 cannot be any flow from u to v, and $f(u, v) = 0$.

The nonnegative quantity $f(u, v)$ is called 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 lecture, 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.

An illustration of these concepts can be observed in the figure below. The figure (a) shows a *flow network* and the figure (b) the *flow*.

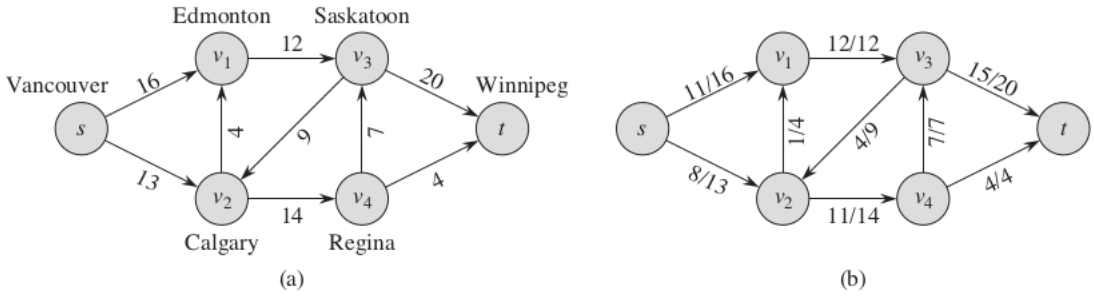

Figure 26.1 **(a)** A flow network $G = (V, E)$ for the Lucky Puck Company's trucking problem. The Vancouver factory is the source s, and the Winnipeg warehouse is the sink t. The company ships pucks through intermediate cities, but only $c(u, v)$ crates per day can go from city u to city v. Each edge is labeled with its capacity. **(b)** A flow f in G with value $|f| = 19$. Each edge (u, v) is labeled by $f(u, v)/c(u, v)$. The slash notation merely separates the flow and capacity; it does not indicate division.

1.2 Dealing with multiple sources and sinks, and antiparallel edges

It is fairly straight-forward to convert a network with multiple sources and sinks to an equivalent network with a single source and a single sink.

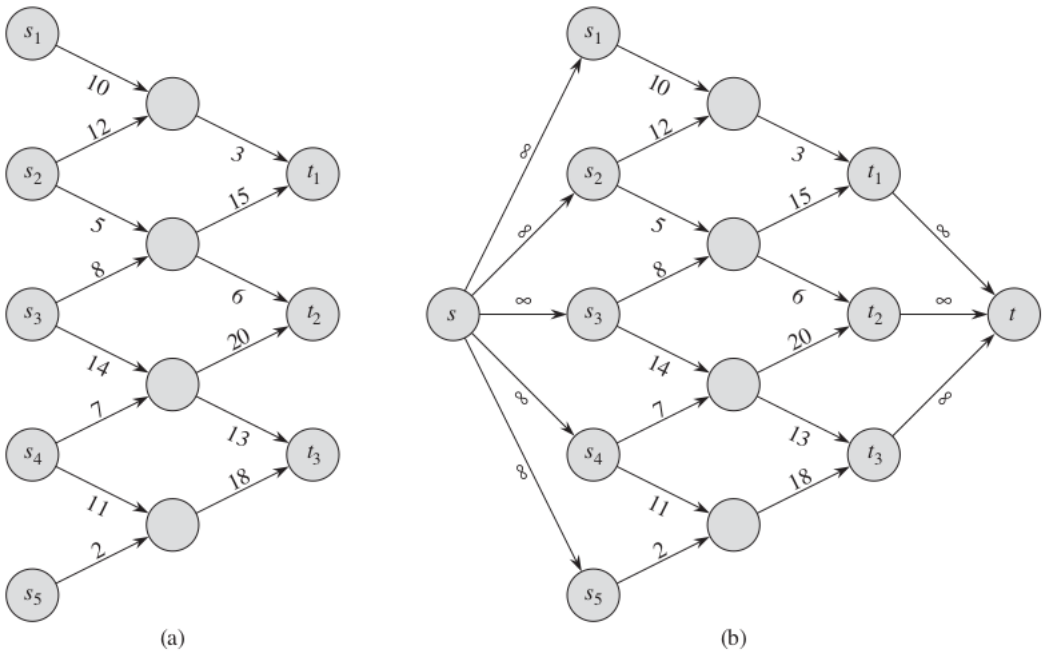

Figure 26.3 Converting a multiple-source, multiple-sink maximum-flow problem into a problem with a single source and a single sink. **(a)** A flow network with five sources $S = \{s_1, s_2, s_3, s_4, s_5\}$ and three sinks $T = \{t_1, t_2, t_3\}$. **(b)** An equivalent single-source, single-sink flow network. We add a supersource s and an edge with infinite capacity from s to each of the multiple sources. We also add a supersink t and an edge with infinite capacity from each of the multiple sinks to t.

It is equally straight-forward to converting a network with antiparallel edges to an equivalent one with no antiparallel edges and then let the algorithm run on the new version of the network flow.

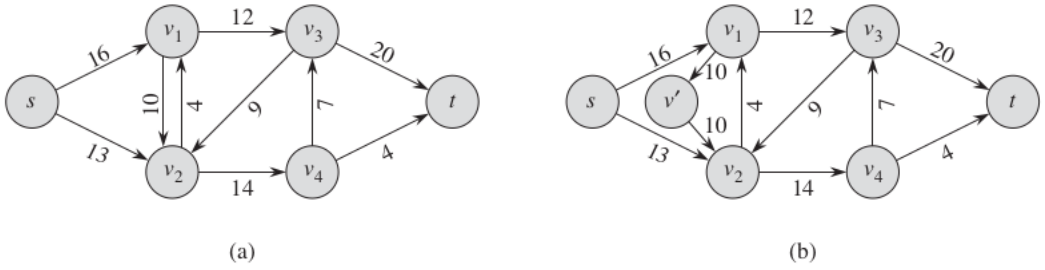

Figure 26.2 Converting a network with antiparallel edges to an equivalent one with no antiparallel edges. **(a)** A flow network containing both the edges (v_1, v_2) and (v_2, v_1). **(b)** An equivalent network with no antiparallel edges. We add the new vertex v', and we replace edge (v_1, v_2) by the pair of edges (v_1, v') and (v', v_2), both with the same capacity as (v_1, v_2).

1.3 The Ford-Fulkerson method

Before we turn our attention to a Ford-Fulkerson algorithm, of which there are several with different running times, we will focus on the generic Ford-Fulkerson method. This will give us the general idea of how the subsequent algorithms run and it will allow us to introduce

the terminology and concepts that the algorithms rely upon.

```
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
```

1.4 Residual networks

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.

The residual network G_f may also contain edges that are not in G! 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 (u, v) into G_f with residual capacity $c_f(u, v) = f(u, v)$ that is, an edge that can admit flow in the opposite direction to (u, v), at most canceling out the flow on (u, v). These reverse edges in the residual network allow an algorithm to send back flow it has already sent along an edge. Sending flow back along an edge is equivalent to decreasing the flow on the edge, which is a necessary operation in many algorithms.

Let $G = (V, E)$ denote a flow network 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

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

In other words, the amount of additional flow we can send from u to v before exceeding the capacity $c(u, v)$ is called the **residual capacity** of edge (u, v). Only one of these cases applies at a time for each ordered pair of vertices, because of our assumption that $(u, v) \in E$ implies $(v, u) \notin E$.

.

In addition, given a flow network $G = (V, E)$ and flow f, the residual network of G induced by f is $G_f = (V, E_f)$, where $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$. That is, each **residual edge** can admit a flow that is greater than 0.

1.5 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.

The shaded path in figure (b) below is an augmenting path. Treating the residual network G_f in the figure as a flow network, we can increase the flow through each edge of this path by up to 4 units without violating a capacity constraint, since the smallest residual capacity on this path is $c_f(v_2, v_3) = 4$. 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 $c_f(p) = min\{c_f(u, v) : (u, v) \text{ is on } p\}$.

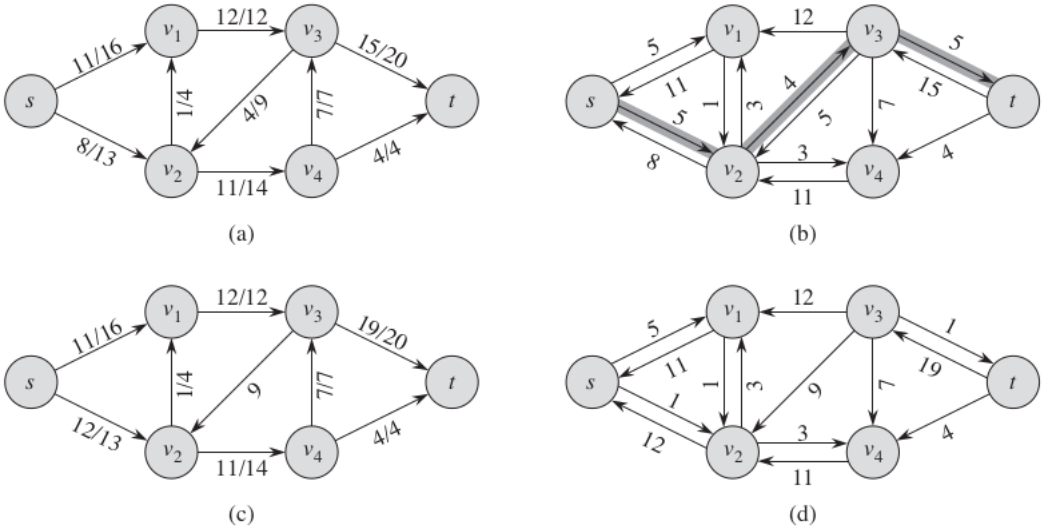

Figure 26.4 **(a)** The flow network G and flow f of Figure 26.1(b). **(b)** The residual network G_f with augmenting path p shaded; its residual capacity is $c_f(p) = c_f(v_2, v_3) = 4$. Edges with residual capacity equal to 0, such as (v_1, v_3), are not shown, a convention we follow in the remainder of this section. **(c)** The flow in G that results from augmenting along path p by its residual capacity 4. Edges carrying no flow, such as (v_3, v_2), are labeled only by their capacity, another convention we follow throughout. **(d)** The residual network induced by the flow in (c).

The following Lemma makes the above argument precise.

Lemma 1. *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*

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

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

In figure above (c) shows the result of augmenting the flow f from (a) by the flow f_p in (b), and (d) shows the resulting residual network.

1.6 Adding back edges is necessry

Addition of back-edges is absolutely necessary. Consider the case where we do not add back edges. Let's say that the algorithm selects the path t, C, B, t (in red) and that that path does not lead to a maximum $|f|$. Then, if we remove the minimum capacity along the path (ie. 1), we have effectively cut-off the possiblity of exploring paths s, A, B, t or s, C, D, t, which may contain the maximum flow. However, by adding back edges, we can explore the other paths and find maximum flow.

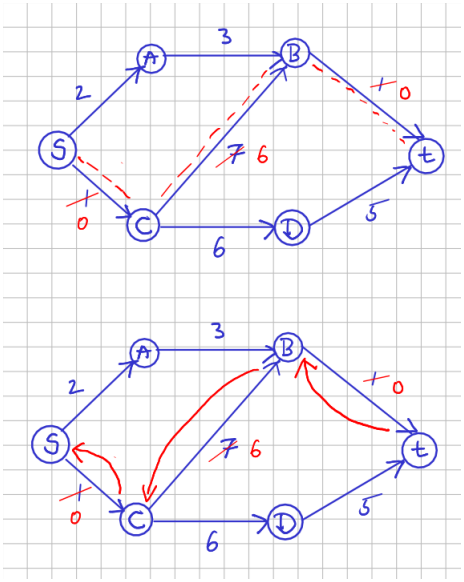

Figure 1: The necessaty of back edges: we need to be able to explore all paths.

1.7 Cuts of flow networks

The Ford-Fulkerson method repeatedly augments the flow along augmenting paths until it has found a maximum flow. How do we know that when the algorithm terminates, we have actually found a maximum flow? The max-flow min-cut theorem tells us that a flow is maximum if and only if its residual network contains no augmenting path. To understand this theorem, though, we must first explore the notion of a cut of a flow network.

A **cut** of flow network $G = (V, E)$ is a partition of V into S and $T = V - S$, such that $s \in S$ and $t \in T$. (This definition is similar to the definition of cut that we used for minimum spanning trees.) If f is a flow, then the **net flow** $f(S, T)$ across the cut (S, T) is defined to be

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

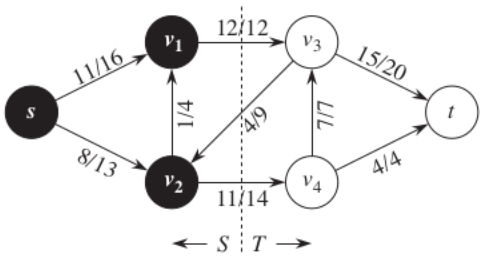

Figure 26.5 A cut (S, T) in the flow network of Figure 26.1(b), where $S = \{s, v_1, v_2\}$ and $T = \{v_3, v_4, t\}$. The vertices in S are black, and the vertices in T are white. The net flow across (S, T) is $f(S, T) = 19$, and the capacity is $c(S, T) = 26$.

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

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

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

With this last piece about the flow in a flow network, we can state the following theorem, which connects all these pieces together.

Theorem 1 (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 path.*
3. *$|f| = c(S, T)$ for some cut (S, T) of G.*

1.8 The basic Ford-Fulkerson algorithm

The *FORD-FULKERSON* algorithm simply expands on the *FORD-FULKERSON-METHOD* pseudo code given earlier. In each iteration of the Ford-Fulkerson method, we find some augmenting path p and use p to modify the flow f. When no augmenting paths exist, the flow f is a maximum flow.

```
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)
```

Example

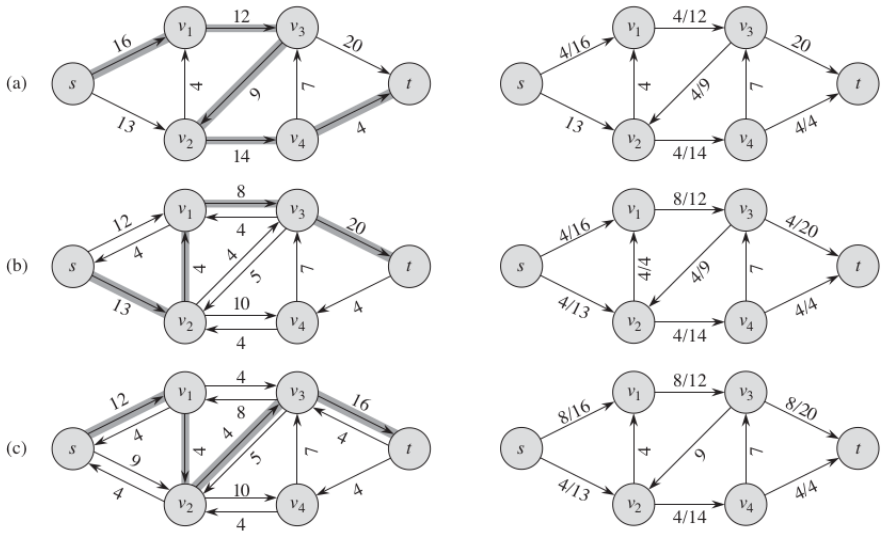

Figure 26.6 The execution of the basic Ford-Fulkerson algorithm. **(a)–(e)** Successive iterations of the **while** loop. The left side of each part shows the residual network G_f from line 3 with a shaded augmenting path p. The right side of each part shows the new flow f that results from augmenting f by f_p. The residual network in (a) is the input network G.

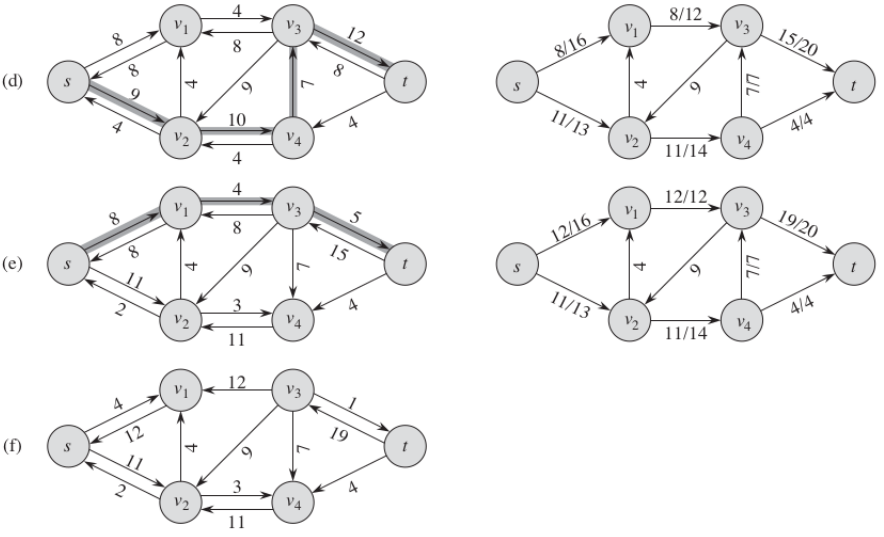

Figure 26.6, continued **(f)** The residual network at the last **while** loop test. It has no augmenting paths, and the flow f shown in (e) is therefore a maximum flow. The value of the maximum flow found is 23.

1.9 Analysis of Ford-Fulkerson algorithm

The running time of *FORD-FULKERSON* algorithm depends on how we find the augmenting path p in line 3. If we choose it poorly, the algorithm might not even terminate: the value of the flow will increase with successive augmentations, but it need not even converge to the maximum flow value. However, if we find the augmenting path by using a breadth-first search, however, the algorithm runs in polynomial time.

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* algorithm 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.

We can perform the work done within the while loop efficiently if we implement the flow network $G = (V, E)$ with the right data structure and find an augmenting path by a linear-time algorithm. For example, 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, where $E' = \{(u, v) : (u, v) \in E \text{ or } (v, u) \in E\}$. Each iteration of the while loop thus takes $O(E)$ time, as does the initialization in lines 1-2, making the total running time of the *FORD-FULKERSON* algorithm $O(E\,|f^*|)$.