

CPSC 365 / ECON 365:

Algorithms

Lecture 15: Network Flow 1

Andre Wibisono

Yale University

March 15, 2022

Last time

Dynamic Programming

Today:

- Maximum Flow Problem
- Ford-Fulkerson Algorithm

PS 5 out today, due April 5

Plan

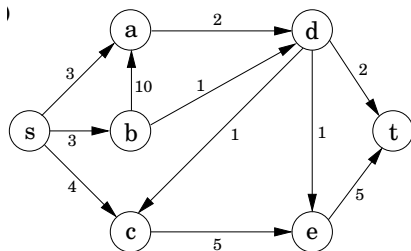
Maximum Flow Problem

Ford-Fulkerson Algorithm

Network

A **Network** is a directed graph $G = (V, E)$ with the following information:

- There is a **source** $s \in V$ with no incoming edges.
 - Assume all vertices reachable from s .
- There is a **sink** $t \in V$ with no outgoing edges.
- Each edge $e \in E$ has a **capacity** $c_e > 0$.
 - Assume capacity is a positive integer: $c_e \in \mathbb{N} = \{1, 2, 3, \dots\}$.



Flow in a Network

Let $G = (V, E)$ be a network with capacity $c = (c_e)_{e \in E}$.

A **flow** (or an $s - t$ flow) is a function $f: E \rightarrow \mathbb{R}_0$ that specifies the amount $f(e)$ to push along edge e , subject to:

1. *Capacity constraint*: $0 \leq f(e) \leq c_e$ for all $e \in E$.
2. *Conservation condition*: For each $v \in V \setminus \{s, t\}$,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

The **value** of a flow f is

$$v(f) = \sum_{e \text{ out of } s} f(e) = \sum_{e \text{ into } t} f(e).$$

Flow Notation

Given $f: E \rightarrow \mathbb{R}_0$, for $v \in V$, define:

$$f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$$

$$f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e).$$

Then capacity constraint for a flow f becomes:

$$f^{\text{in}}(v) = f^{\text{out}}(v) \quad \text{for all } v \neq s, t.$$

We can write the value of a flow f as:

$$v(f) = f^{\text{out}}(s) = f^{\text{in}}(t)$$

Maximum Flow Problem

Given a network G be a network with capacity $c = (c_e)_{e \in E}$.

Goal: Find a flow f with the maximum value $v(f)$.

maximize $v(f)$

subject to $0 \leq f(e) \leq c_e \quad \forall e \in E,$

$f^{\text{in}}(v) = f^{\text{out}}(v) \quad \forall v \neq s, t.$

How to find a maximum flow?

Idea 1: Greedy method, grow the flow

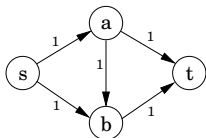
1. Start with the zero flow.
2. Repeat: Choose an appropriate path from s to t , and increase flow along the edges of this path as much as possible.

Issue: If choose a wrong path in the beginning, can get stuck.

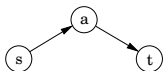
- So should allow possibility to correct previous flow.

How to find a maximum flow? Example

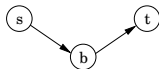
Network:



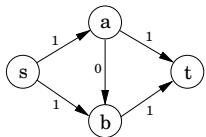
Suppose method chooses paths:



then

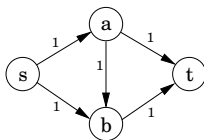


This gives the correct maximum flow with value 2:

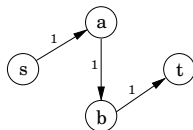


How to find a maximum flow? Example

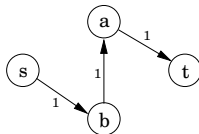
Network:



But if we first choose this path:



Then we should also allow the method to choose this path:



How to find a maximum flow?

Idea 2: Grow our flow, but allow correction (undo previous flow)

1. Start with the zero flow
2. Repeat: Choose an **augmenting path** in the **residual graph**, and **augment** the flow along the edges of this path.

Residual Graph

Let f be a flow on a network $G = (V, E)$. The **residual graph** $G_f = (V, E_f)$ is a directed graph on V with the following edges:

For each edge $e = (u, v) \in E$:

- If $f(e) < c_e$, then we add $e = (u, v)$ to E_f with capacity

$$c_f(e) = c_e - f(e) > 0.$$

We call $e \in E_f$ a **forward edge** (we can push more on $f(e)$).

- If $f(e) > 0$, then we add $e' = (v, u)$ to E_f with capacity

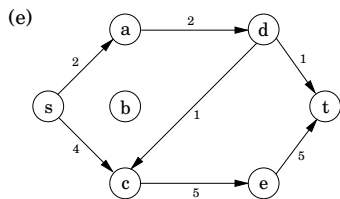
$$c_f(e') = f(e) > 0.$$

We call $e' \in E$ a **backward edge** (we can subtract some from $f(e)$).

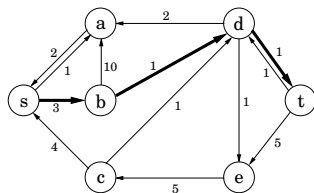
Note: Each $e \in E$ gives rise to up to 2 edges in E_f , so $|E_f| \leq 2|E|$.

Residual Graph

Current Flow



Residual Graph



Augmenting Paths

Let $G_f = (V, E_f)$ be the residual graph of a flow f on $G = (V, E)$.

- An **augmenting path** is a simple path P from s to t in G_f .
 - Simple = does not repeat nodes, so P uses $\leq n - 1$ edges.
- The **bottleneck** of P is the minimum capacity (in G_f) along the edges of P :

$$\text{bottleneck}(P) = \min_{e \in P} c_f(e)$$

- Idea: We can **augment** the flow f along P by $\beta = \text{bottleneck}(P)$.
For each $e \in P$:
 - If e is a **forward** edge, then **increase** flow $f(e)$ by β .
 - If e is a **backward** edge, then **decrease** flow $f(e)$ by β .

Augmenting the Flow

Given a flow f and an augmenting path P , define:

Augment(f, P):

Let $\beta = \text{bottleneck}(P)$

For each edge $e = (u, v) \in P$:

If e is a forward edge:

increase $f(e)$ in G by β : $f'(e) = f(e) + \beta$

Else: (e is a backward edge, so $e' = (v, u) \in E$)

decrease $f(e')$ in G by β : $f'(e') = f(e') - \beta$

Return f'

Augmenting the Flow: Properties

Let f be a flow in G , and let P be an augmenting path in G_f .
Let $f' = \text{Augment}(f, P)$.

Lemma 1: f' is also a flow in G .

Lemma 2: f' has value $v(f') = v(f) + \text{bottleneck}(P)$.

Lemma 1

Let f be a flow in G , and let P be an augmenting path in G_f .
Let $f' = \text{Augment}(f, P)$.

Lemma 1: f' is also a flow in G .

Proof: Since f' differs from f only on the edges of P , we only need to verify the capacity and conservation conditions on these edges.

Let $\beta = \text{bottleneck}(P)$, so $0 < \beta \leq c_f(e)$ for all $e \in P$.

Lemma 1 Proof

We check the capacity condition. Let $e \in P$. There are two cases:

1. If e is a forward edge, then $e \in E$, and its residual capacity is $c_f(e) = c_e - f(e)$. Then $f'(e) = f(e) + \beta$ satisfies

$$f'(e) > f(e) \geq 0$$

and

$$f'(e) \leq f(e) + c_f(e) = f(e) + (c_e - f(e)) = c_e$$

2. If $e = (u, v)$ is a backward edge, then $e' = (v, u) \in E$, and its residual capacity is $c_f(e) = f(e')$. Then $f'(e') = f(e') - \beta$ satisfies

$$f'(e') \geq f(e') - c_f(e) = f(e') - f(e') = 0$$

and

$$f'(e') \leq f(e') \leq c_{e'}$$

Therefore, the capacity condition is satisfied by f' for all $e \in E$.

Lemma 1 Proof

We check the conservation condition along the vertices in P . Let $v \in P$, $v \neq s, t$. P must enter v once, say along edge $e_1 = (u, v) \in E_f$; and P must exit v once, say along edge $e_2 = (v, w) \in E_f$. There are four cases:

- (a) If e_1 and e_2 are forward edges: Then $e_1, e_2 \in E$, and $f'(e_1) = f(e_1) + \beta$, $f'(e_2) = f(e_2) + \beta$. Then

$$\begin{aligned}(f')^{\text{in}}(v) &= f^{\text{in}}(v) + f'(e_1) - f(e_1) = f^{\text{in}}(v) + \beta \\ (f')^{\text{out}}(v) &= f^{\text{out}}(v) + f'(e_2) - f(e_2) = f^{\text{out}}(v) + \beta.\end{aligned}$$

Since $f^{\text{in}}(v) = f^{\text{out}}(v)$, we see that $(f')^{\text{in}}(v) = (f')^{\text{out}}(v)$.

- (b) If e_1 is a forward edge and e_2 is a backward edge: Then $e_1 \in E$, $f'(e_1) = f(e_1) + \beta$, and $e'_2 = (w, v) \in E$, $f'(e'_2) = f(e'_2) - \beta$. Then

$$\begin{aligned}(f')^{\text{in}}(v) &= f^{\text{in}}(v) + f'(e_1) - f(e_1) + f'(e'_2) - f(e'_2) \\ &= f^{\text{in}}(v) + \beta - \beta \\ &= f^{\text{in}}(v) = f^{\text{out}}(v) \\ (f')^{\text{out}}(v) &= f^{\text{out}}(v).\end{aligned}$$

The other two cases are similar and left as exercise.

Lemma 2

Let f be a flow in G , and let P be an augmenting path in G_f .
Let $f' = \text{Augment}(f, P)$.

Lemma 2: f' has value $v(f') = v(f) + \text{bottleneck}(P)$.

Proof: The first edge e of P must be an edge out of s in the residual graph G_f . Since P is simple, it does not visit s again. Since G has no edges entering s , the edge e must be a forward edge in G_f .

We increase the flow on e by $\text{bottleneck}(P)$, and we don't change the flow on any other edge incident to s .

Therefore, the value of f' increases by exactly $\text{bottleneck}(P)$:

$$v(f') = v(f) + \text{bottleneck}(P).$$



Plan

Maximum Flow Problem

Ford-Fulkerson Algorithm

Ford-Fulkerson

The Ford-Fulkerson Algorithm for finding maximum flow.

Max-Flow(G): $G = (V, E, c)$

Initialize $f(e) = 0$ for all $e \in E$

While there is an $s - t$ path in the residual graph G_f :

 Choose an augmenting path P in G_f

$f' = \text{Augment}(f, P)$

 Update $f \leftarrow f'$

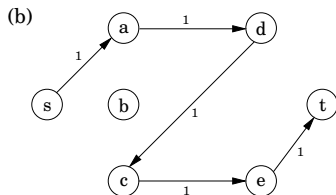
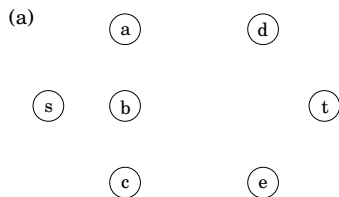
 Update the residual graph G_f to be $G_{f'}$

Return f

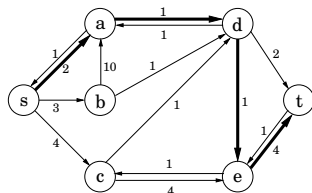
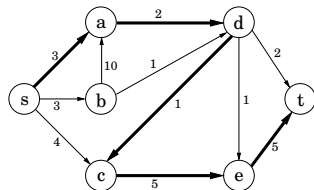
Note: Doesn't specify how to choose augmenting path; any choice works.

Ford-Fulkerson: Example

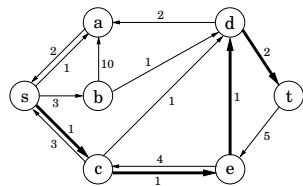
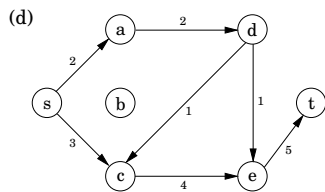
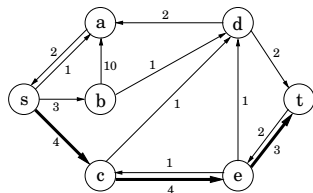
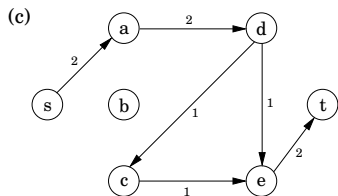
Current flow



Residual graph

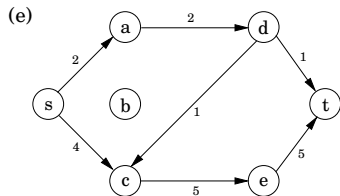


Ford-Fulkerson: Example

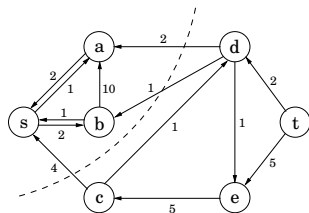
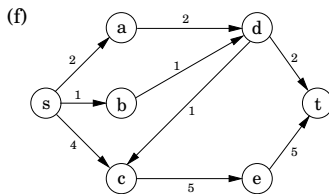
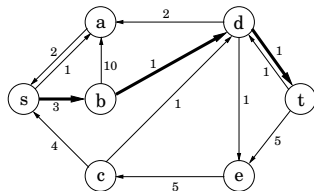


Ford-Fulkerson: Example

Current Flow



Residual Graph



Ford-Fulkerson: Properties

Let $G = (V, E)$ be a network with positive integer capacities c_e .

Lemma 3: At every intermediate stage of Ford-Fulkerson, the flow values $\{f(e) : e \in E\}$ and the residual capacities in G_f are integers.

Proof: We use induction on the number of iterations j .

The statement is true before any iterations of the While loop since we start with the zero flow.

Suppose the statement is true after j iterations.

Since the residual capacities in G_f are integers, the value $\text{bottleneck}(P)$ 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. □

Ford-Fulkerson: Properties

Let $G = (V, E)$ be a network with positive integer capacities c_e .

Let $C = \sum_{e \text{ out of } s} c_e$.

Lemma 4: The Ford-Fulkerson algorithm terminates in at most C iterations of the While loop.

Proof: By Lemma 2, each iteration of the While loop increases the value of the flow by the bottleneck ≥ 1 .

We start with a flow of value 0.

The value of the maximum flow is at most C .

Therefore, the While loop can run for at most C iterations. □

Ford-Fulkerson: Running Time

Let $G = (V, E)$ be a network with positive integer capacities c_e .

Let $C = \sum_{e \text{ out of } s} c_e$.

Theorem 1: The Ford-Fulkerson algorithm can be implemented to run in $O(|E| \cdot C)$ time.

- This is *pseudo-polynomial* time.
- Input is $b = \log C$ bits, so running time is $O(|E| \cdot 2^b)$.

Ford-Fulkerson: Running Time

Let $G = (V, E)$ be a network with positive integer capacities c_e .

Let $C = \sum_{e \text{ out of } s} c_e$.

Theorem 1: The Ford-Fulkerson algorithm can be implemented to run in $O(|E| \cdot C)$ time.

Proof: By Lemma 4, there are at most C iterations of the While loop.

In each iteration, to find an $s - t$ path in G_f , we can run BFS or DFS. Since $|E_f| \leq 2|E|$, this takes $O(|V| + |E_f|) = O(|V| + |E|) = O(|E|)$.

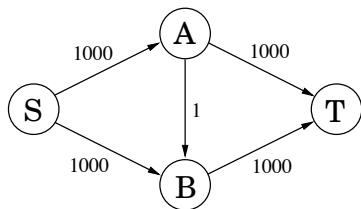
To augment the flow with the path takes $O(n)$ time.

Given the new flow f' , we build the new residual graph in $O(m)$ time: For each edge $e \in E$, we construct the forward and backward edges in $G_{f'}$.

Then the total running time is $O(|E| \cdot C)$. □

Ford-Fulkerson: Slow Example

Ford-Fulkerson can take $O(C)$ iterations if we don't choose augmenting paths carefully.



Ford-Fulkerson: Choosing Augmenting Paths

There are many good choices of augmenting paths that give polynomial-time implementation of Ford-Fulkerson:

1. Scaling max-flow: $O(|E|^2 \log C)$ time
2. Edmonds-Karp: Choose shortest augmenting path (with fewest edges): $O(|V| \cdot |E|^2)$ time

Next Time

Next time:

- Correctness: Ford-Fulkerson returns a **maximum** flow
- Max-flow min-cut theorem