Analysis of Algorithms, I

CSOR W4231.002

Eleni Drinea
Computer Science Department

Columbia University

Tuesday, April 7, 2015

Outline

Today

1. Recap
2. Flow networks
   - Applications
3. The residual graph and augmenting paths
4. The Ford-Fulkerson algorithm for max flow
5. Faster algorithms for max flow

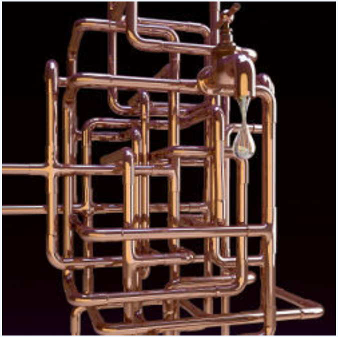
Review of the last lecture

A union-find data structure for maintaining disjoint sets.

- Implementation: maintain sets as directed rooted trees;
  - `Makeset`: worst-case time is $O(1)$
  - `Find`: worst-case time is $O(\log n)$
  - `Union`: worst-case time is $O(\log n)$
- *Improved* implementation by using **path compression**; amortized time for a sequence of $2m$ `Find` operations: $O((m+n)\log^* n)$
  - this is not the tightest possible analysis but it is already fairly subtle

Today

Modeling transportation networks



Source: Communications of the ACM, Vol. 57, No. 8

Can model a fluid network or a highway system by a graph:
edges carry *traffic*, nodes are *switches* where traffic gets diverted.

Flow networks

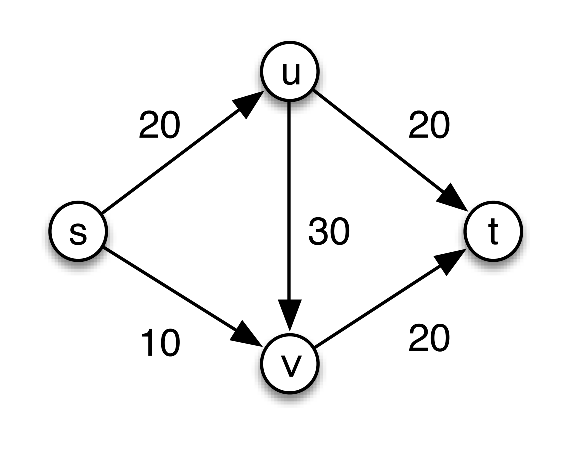
A flow network $G = (V, E)$ is a directed graph such that

1. Every edge has a capacity $c(e) \geq 0$. *A1: integer capacities*
2. There is a single source $s \in V$. *A2: no edge enters s*
3. There is a single sink $t \in V$. *A3: no edge leaves t*

Two more assumptions for the purposes of the analysis

- *A4: if $(u, v) \in E$ then $(v, u) \notin E$.*
- *A5: Every $v \in V - \{s, t\}$ is on some s-t path.*
  Hence G has $m \geq n - 1$ edges.

An example flow network



Flows

Given a flow network G, an s-t flow f in G is a function

$$f : E \to R^+$$

Intuitively, the flow $f(e)$ on edge e is the amount of *traffic* that edge e carries.

Two kinds of constraints that every flow must satisfy

1. Capacity constraints: for all $e \in E$, $0 \leq f(e) \leq c(e)$.

2. Flow conservation: for all $v \in V - \{s, t\}$,

$$\sum_{(u,v) \in E} f(u, v) = \sum_{(v,w) \in E} f(v, w) \tag{1}$$

   In words, the flow into node v equals the flow out of v, or

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

A cleaner equation for flow conservation constraints

Define

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

So we can rewrite equation (1) as: for all $v \in V - \{s, t\}$

$$f^{\text{in}}(v) = f^{\text{out}}(v) \tag{2}$$

The value of a flow

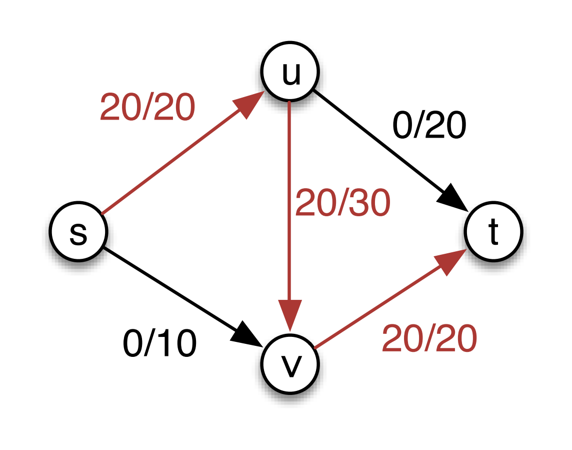
Definition 1.

The value of a flow f, denoted by $|f|$, is

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

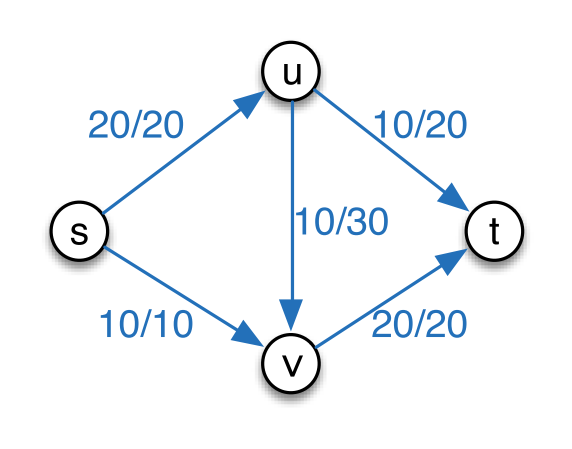
Can show that $|f| = f^{\text{in}}(t)$. *(exercise)*

An example flow of value 20



A flow f of value 20.

Another flow, of value 30



A (max) flow of value 30.

Max flow problem

Input: (G, s, t, c) such that

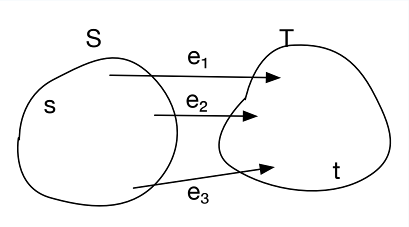
- $G = (V, E)$ is a flow network;
- $s, t \in V$ are the source and sink respectively;
- c is the (integer-valued) capacity function.

Output: a flow of maximum possible value

s-t cuts in flow networks

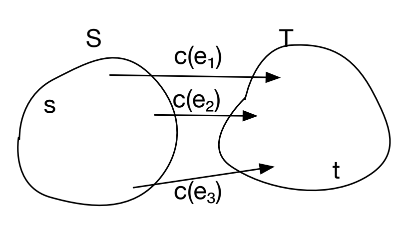
Definition 2.

An s-t cut (S, T) in G is a partition of the vertices into two sets S and T, such that $s \in S$ and $t \in T$.



A natural upper bound for the max value of a flow

- Flow f must cross (S, T) to go from source s to sink t.
- So it uses some (at most all) of the capacity of the edges crossing this cut.



- So, intuitively, the value of the flow cannot exceed

$$\sum_{\text{e out of } S} c(e)$$

Max flow and min cut

Definition 3.

The capacity $c(S, T)$ of an s-t cut (S, T) is defined as

$$c(S, T) = \sum_{\text{e out of } S} c(e).$$

$\triangle$ Note asymmetry in the definition of $c(S, T)$!

So, *intuitively*, the value of the max flow is upper bounded by the capacity of *every* cut in the flow network, that is,

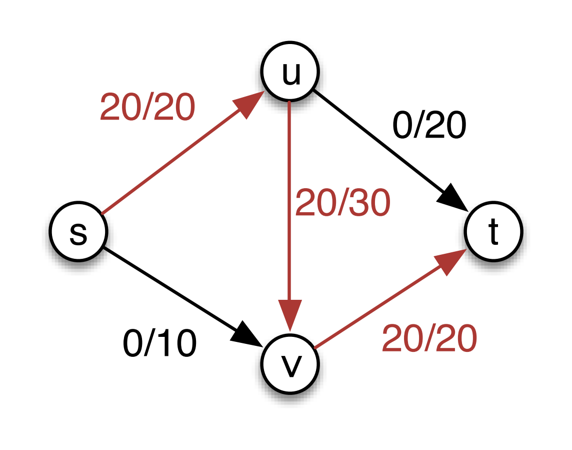
$$\max_f |f| \leq \min_{(S,T) \text{ cut in } G} c(S, T) \tag{3}$$

Applications of max-flow and min-cut

- Min-cut
  - find a set edges of smallest capacity whose deletion disconnects the network
- Max-flow
  - Bipartite matching *(next lecture)*
  - Airline scheduling
  - Baseball elimination
  - Distribution of goods to cities
  - Image segmentation
  - Survey design
  - ...

Today

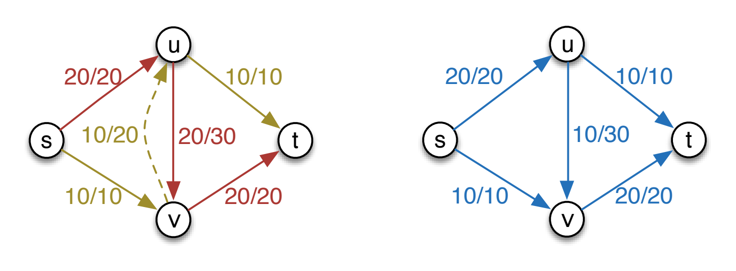
"Undoing" flow



A flow f of value 20.

Would like to undo 10 units of flow along (u, v) and divert it along (u, t).

Pushing flow back



- Push back 10 units of flow along (v, u).
- Send 10 more units from s to t along edges $(s, v), (v, u), (u, t)$.
- New flow f' (on the right) with value 30.

Pushing flow forward and backward

By pushing flow back on (v, u) we created an s-t path on which we are pushing flow

- Forward, on edges with leftover capacity (e.g, on (s, v))
- Backward, on edges that are already carrying flow so as to divert it to a different direction (e.g., on (u, v)).

The residual graph G_f

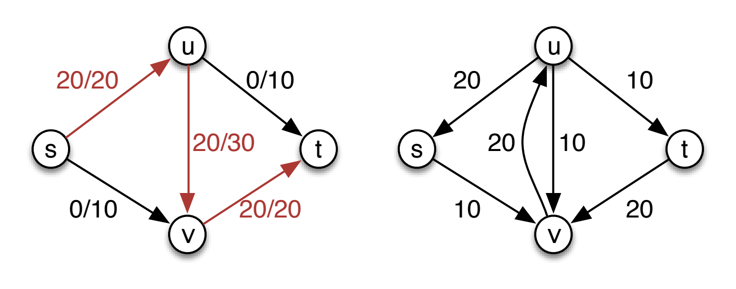
Definition 4.

Given flow network G and flow f, the residual graph G_f has

- the same vertices as G;
- for every edge $e = (u, v) \in E$ such that $f(e) < c(e)$, an edge $e = (u, v)$ with capacity $c_f(e) = c(e) - f(e)$ (forward edge);
- for every edge $e = (u, v) \in E$ such that $f(e) > 0$, an edge $e^r = (v, u)$ with capacity $c_f(e^r) = f(e)$ (backward edge).

So G_f has $\leq 2m$ edges.

Example residual graph



Left: a flow f of value 20.
Right: the residual graph G_f for this flow.

Augmenting paths

The residual graph G_f provides a roadmap for augmenting f.

1. Let P be a simple s-t path in G_f.
2. Augment f by pushing extra flow on P.

How much extra flow can we push on P without violating capacity constraints in G_f?

- Let $c(P)$ be the capacity of path P defined as the minimum residual capacity of **any** edge of P.

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

- The maximum amount of flow we can safely push on **every** edge of P is $c(P)$.

The augmented flow f'

Let P be an augmenting path in the residual graph G_f.
Augmented flow f' is as follows:

1. For a **forward** edge $e \in P$

$$f'(e) = f(e) + c(P)$$

2. For a **backward** edge $e^r = (u, v) \in P$, let $e = (v, u) \in G$

$$f'(e) = f(e) - c(P)$$

3. For $e \in E$ but not in P, $f'(e) = f(e)$.

Fact 5 (1).

f' is a flow.

Pseudocode

Augment(f, P)

for each edge $(u, v) \in P$ **do**

    if $e = (u, v)$ is a forward edge **then**

        $f'(e) = f(e) + c(P)$

    else

        $f'(v, u) = f(v, u) - c(P)$

    end if

end for

Return f'

Today

The Ford-Fulkerson algorithm

Ford-Fulkerson($G = (V, E, c), s, t$)

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

    end for

    while there is an s-t path in G_f **do**

        Let P be a simple s-t path in G_f

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

        Update $f = f'$

        Update $G_f = G_{f'}$

    end while

    Return f'

Running time analysis

The algorithm terminates if the following claims are both true

1. **Claim 1:** every iteration of the while loop returns a flow increased by an integer amount; and
2. **Claim 2:** there is a finite upper bound to the flow.

Proof of Claim 2.

Let U be the largest edge capacity. Then

$$|f| \leq \sum_{e \text{ out of } s} c(e) \leq nU$$

$\square$

f increases by an integer amount after Augment(f, P)

Proof of Claim 1.

It follows from the following facts.

Fact 6 (2).

During execution of the Ford-Fulkerson algorithm, the flow values $\{f(e)\}$ and the residual capacities in G_f are all integers.

Fact 7 (3).

Let f be a flow in G and P a simple s-t path in G_f with residual capacity $c(P) > 0$. Then after Augment(f, P)

$$|f'| = |f| + c(P) \geq |f| + 1.$$

$\square$

f increases by an integer amount after Augment(f, P)

Proof of Fact 3.

Recall that $|f| = f^{\text{out}}(s)$.

1. Since P is an s-t path, it contains an edge out of s, say (s, u).
2. Since P is simple, it does not contain any edge entering s (P is in G_f, where there are edges entering s!): otherwise, s would be visited again.
3. Since no edge enters s in G, (s, u) is a forward edge in G_f, thus the flow on this edge is updated to $f(s, u) + c(P) \geq f(s, u) + 1$.
4. Since no other edge going out of s is updated, it follows that the value of f' is $|f'| = |f| + c(P) \geq |f| + 1$.

$\square$

Running time of Ford-Fulkerson

1. Fact 3 guarantees at most nU iterations.
2. The running time of each iteration is bounded as follows:
   - $O(m + n)$ to create G_f using adjacency list representation
   - $O(m + n)$ to run BFS or DFS to find the augmenting path
   - $O(n)$ for Augment(f, P) since P has at most $n - 1$ edges

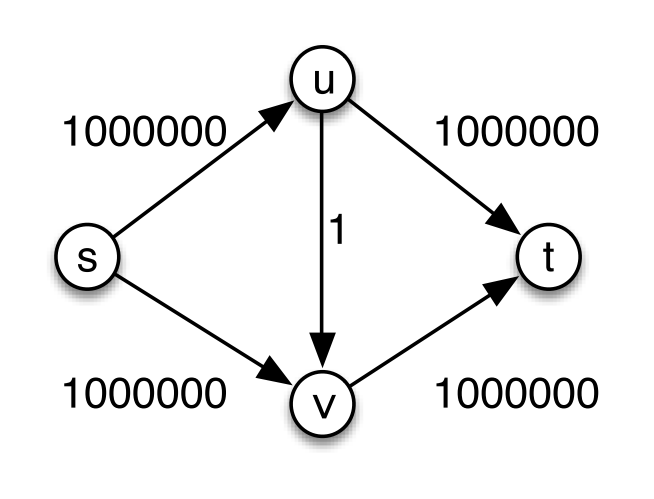
   $\Rightarrow$ Hence one iteration requires $O(m)$ time

The running time of Ford-Fulkerson is $O(mnU)$.

Remark 1.

This is a pseudo-polynomial time algorithm: it is **not** *polynomial in the description of the input U*

Problems with pseudo-polynomial running times



Today

1. Recap
2. Flow networks
   - Applications
3. The residual graph and augmenting paths
4. The Ford-Fulkerson algorithm for max flow
5. Faster algorithms for max flow

Algorithms

- Can be made polynomial: use BFS instead of DFS
  - Edmonds-Karp: $O(nm^2)$
- Unit capacities: $O(\min(\sqrt{m}, n^{2/3})m)$
- Integral capacities: $O(\min(\sqrt{m}, n^{2/3})m \log(n^2/m) \log U)$ [GoldbergRao1998]
- Real capacities: $O(nm \log(n^2/m))$
  - Improved: $O(nm)$ [Orlin2013]