CS 341: ALGORITHMS

Lecture 16: max flow

Readings: CLRS 26.2

Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

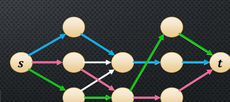
FLOWS AND PATHS

FLOWS AND PATHS

- **Edge-disjoint paths** problem
- Input: digraph $G = (V, E)$ and two vertices $s, t \in V$
- Output: A maximal number of edge-disjoint paths in G
- Paths P_1 and P_2 are **edge-disjoint** if they do not share any edges

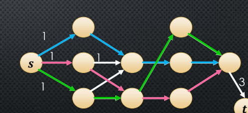
This is a special case of the **maximum flow** problem

… where the union of paths defines a **flow**



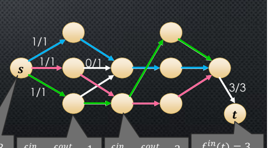
s-t FLOWS

- Let $G = (V, E)$ be a digraph where each edge $e \in E$ has a **capacity** $c(e) > 0$



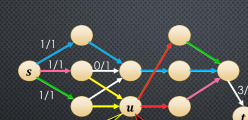
s-t FLOWS

- Let $G = (V, E)$ be a digraph where each edge $e \in E$ has a **capacity** $c(e) > 0$
- An s-t flow assigns a number $f(e)$ to each edge satisfying:
  - Capacity constraints
    - $0 \le f(e) \le c(e)$ for each edge
  - Conservation of flow
    - $f^{in}(v) = f^{out}(v)$ for $v \notin \{s,t\}$
  - **Source** $f^{in}(s) = 0$
  - **Sink** $f^{out}(t) = 0$



This is the **value** of the flow

DEFINING $f^{in}(u)$ AND $f^{out}(u)$



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

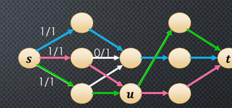
$f^{out}(u) = \sum_{e \text{ out of } u} f(e)$

MAX s-t FLOW

- Input: digraph $G = (V, E)$ with capacities $c(e)$ for $e \in E$, and two vertices s, t
- Output: a flow from s to t with maximum value i.e., that maximizes $f^{out}(s)$
- Motivation
  - Liquid flowing through pipes
    Current through electrical networks
    Internet/telephony traffic routing
  - Also useful for seemingly unrelated problems (next time)

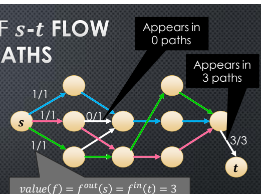
CONNECTION BETWEEN FLOWS AND PATHS

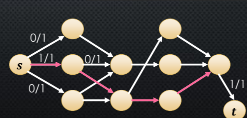
- In this example, max flow is 3
- Note max flow is limited by the sum of capacities **out of** s
  - … and **into** t
- Flows vs paths
  - a flow can always be decomposed into "capacity-disjoint" paths



LEMMA 1: DECOMPOSITION OF s-t FLOW INTO CAPACITY-DISJOINT s-t PATHS

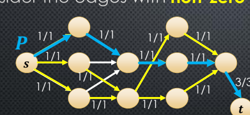
- Let f be an s-t flow where $f(e)$ is an integer for each $e \in E$, $f^{in}(s) = 0$ and $value(f) = k$
- Then there are s-t paths $P_1, P_2, \dots, P_k$ such that each **edge** e appears in $f(e)$ of these **paths**
- Proof sketch by induction
  - Base case: when k=1 there is only one path





INDUCTIVE STEP

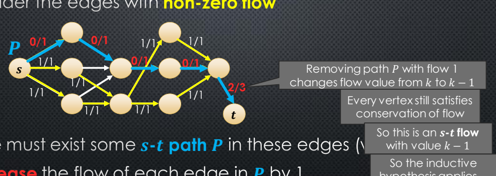
- Suppose lemma holds for $k-1$, show it holds for k (where $k \geq 2$)
- Consider the edges with **non-zero flow**



- There must exist some ***s*-*t* path *P*** in these edges (why?)
- **Decrease** the flow of each edge in ***P*** by 1

INDUCTIVE STEP

- Suppose lemma holds for $k-1$, show it holds for k (where $k \geq 2$)
- Consider the edges with **non-zero flow**

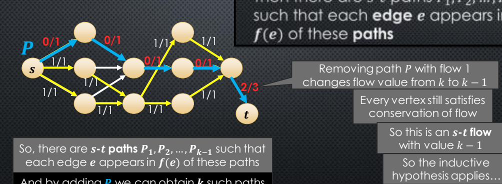


- There must exist some ***s*-*t* path *P*** in these edges (w
- **Decrease** the flow of each edge in ***P*** by 1

INDUCTIVE STEP

Lemma

- Let f be an s-t flow where $f(e)$ is an integer for each $e \in E$, $f^{in}(s) = 0$ and $value(f) = k$
- Then there are s-t paths $P_1, P_2, \dots, P_k$ such that each **edge** e appears in $f(e)$ of these **paths**



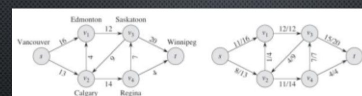
EXAMPLE APPLICATION OF LEMMA 1

- Given a flow of value k where $f(e) \in \{0,1\}$ for all $e \in E$
- The lemma says the flow f can be decomposed into k edge-disjoint paths
- So if our goal is to find k edge-disjoint paths we can just focus in finding such a flow instead
  - (so we don't need to worry about which edges belong to which paths during the algorithm)
- Can extract paths from such a flow by repeatedly doing: BFS on the non-zero flow edges, identifying an s-t path, and decrementing the flows along that path

13

HOME EXERCISE

- Find a decomposition of the following flow into capacity-disjoint paths



14

FLOWS AND CUTS

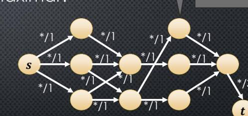
15

UPPER BOUNDING THE MAX FLOW FOR G

- What is a good upper bound on the value of a flow?
  - And how do we know a flow is maximal?
- Trivial upper bound
  - Sum of capacities of all edges
- Slightly better
  - $\min\left(c^{out}(s), c^{in}(t)\right)$ where $c^{out}(s) = \sum_{e \text{ out of } s} c(e)$ and $c^{in}(t) = \sum_{e \text{ into } t} c(e)$
  - Tightly bounds max flow in this case…

$\sum c(e) = 17$

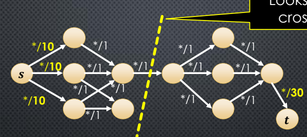
But max flow is 3…



16

BUT WHAT ABOUT THIS CASE?

Looks like an edge crossing a cut…



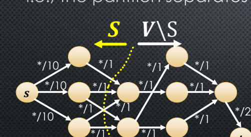
Estimate using $\min\left(c^{out}(s), c^{in}(t)\right) = 30$

But real answer is 1…

17

DEFINITIONS: AN s-t CUT AND ITS CAPACITY

- An s-t cut is a partition (S,V\S) where $s \in S$ and $t \in V\backslash S$
  - i.e., the partition separates s and t



(Recall S does not need to be connected)

Let $\delta^{out}(S)$ be the set of edges **directed out from S**

$\delta^{out}(S) = \{(u,v) \in E : u \in S, v \in V\backslash S\}$

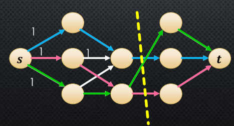
The **capacity** of the cut is the sum of the capacities of these edges

$c^{out}(S) = \sum_{e \in \delta^{out}(S)} c(e)$

18

UPPER BOUNDING EDGE-DISJOINT PATHS BY S-T CUT

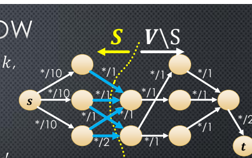
- For the edge-disjoint paths problem, where $c(e) = 1$ for all e, cut capacity is just the **number of edges crossing the cut**
- If an s-t cut S has at most k edges crossing the cut, then are at most k edge-disjoint s-t paths, since each s-t path has an edge crossing the cut



19

GENERALIZING TO MAX FLOW

Lemma 2: if an s-t cut S has capacity k, the value of every flow must be $\leq k$



- Proof sketch: for contra assume a flow with value $k' > k$
- By earlier lemma, a flow with value k' can be decomposed into k' capacity-disjoint paths each w/flow 1
- Each such path crosses the cut, and consumes one unit of the cut's capacity (up to k' in total)
- But the cut's capacity is only k, so the paths are not capacity-disjoint! Contradiction.

20

COROLLARY: MAX FLOW ≤ MIN s-t CUT

- Recall lemma 2: if an s-t cut S has capacity k, the value of every flow must be $\leq k$
- This holds for **any** s-t cut
- Including the s-t cut S with the minimum capacity
- So, **max s-t flow ≤ min capacity over all possible s-t cuts**

- In fact, it turns out max flow is **exactly** the min cut capacity
  - So we can solve max flow by finding a min cut…

21

MIN s-t CUT PROBLEM

- Input: digraph $G = (V, E)$ with capacities $c(e) > 0$ for $e \in E$, and two vertices s, t
- Output: an s-t cut S with minimal capacity $c^{out}(S)$
- This is a natural and useful problem on its own, and we will see some other interesting applications soon…

22

MAX-FLOW MIN-CUT THEOREM

- **Theorem 3:** every max s-t flow has value equal to the capacity of a min s-t cut
- One of the most beautiful and important results in combinatorial optimization and graph theory
- Diverse applications in CS and math
- We give an **algorithmic proof** of this theorem
  - (showing that one algorithm solves both max-flow and min-cut at the same time)

23

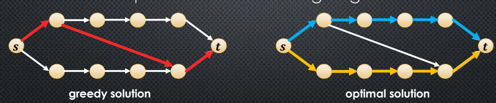
FORD-FULKERSON METHOD

Algorithm development
(mixed together with proof of max-flow min-cut theorem)

24

NAÏVE ALGORITHM ATTEMPT

- For simplicity, try edge-disjoint path problem first (unit capacities)
- Greedy idea: find a shortest s-t path (to use few edges), then repeat on the remaining edges



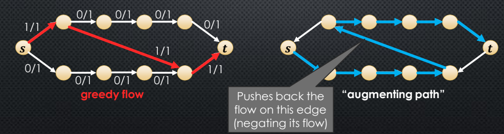
- Difficult for greedy is to **decide** on a path **permanently**
- Unclear how to find a path that **belongs** in the optimal solution

25

FORD-FULKERSON METHOD

Same Ford as in Bellman-Ford :)

- Ford-Fulkerson is a more general "local search" algorithm which can **undo** previous decisions to improve the flow
- Greedy flow can be improved by "pushing back" some flow using an **augmenting path** through a **residual graph**

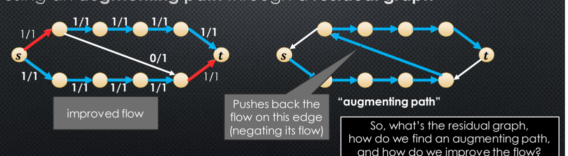


26

FORD-FULKERSON METHOD

Same Ford as in Bellman-Ford :)

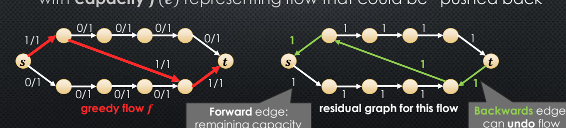
- Ford-Fulkerson is a more general "local search" algorithm which can **undo** previous decisions to improve the flow
- Greedy flow can be improved by "pushing back" some flow using an **augmenting path** through a **residual graph**



27

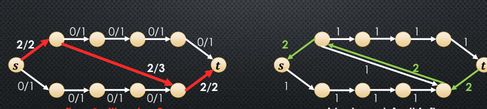
RESIDUAL GRAPH

- A **residual graph** R_f is defined for a **given flow** f and **graph** G
- R_f has the same vertices as G
- For each edge $e = uv$ in G,
  - If $f(e) < c(e)$, then R_f contains a **forward** edge (u, v) with the **remaining capacity** $c(e) - f(e)$
  - If $f(e) > 0$, then R_f contains a **backwards** edge (v, u) with **capacity** $f(e)$ representing flow that could be "pushed back"



ANOTHER EXAMPLE RESIDUAL GRAPH

- Recall: for each edge $e = uv$ in G,
  - If $f(e) < c(e)$, then R_f contains a **forward** edge (u, v) with the **remaining capacity** $c(e) - f(e)$
  - If $f(e) > 0$, then R_f contains a **backwards** edge (v, u) with **capacity** $f(e)$ representing flow that could be "pushed back"



29