Network Flow

capacity constraint: $\forall e \in E: 0 \le f(e) \le c(e)$

flow conservation: $\forall v \in V \setminus \{s,t\}$ $\sum_{e \text{ entering } v} f(e) = \sum_{e \text{ leaving } v} f(e)$

$f^{in}(v) = \sum_{e \text{ entering } v} f(e)$ $f^{out}(v) = \sum_{e \text{ leaving } v} f(e)$

$v(f) = f^{out}(s) = f^{in}(s)$ $f^{out}(u) = f^{in}(u)$ for all $u \ne s,t$

Residual Graph G_f: **Forward edge**: capacity $c(e) - f(e)$

add when capacity > 0 — **Reverse edge**: $e^{rev} = (v,u)$ capacity $f(e)$

Augmenting Path: P = an s-t path in G_f

bottle neck (P,f): smallest capacity across all edges in P

Augment: sending bottleneck(P,f) units of flow along P.

send x units of flow along: P: forward e: increase f one by x
reverse e: decrease f one by x

Ford-Fulkerson Algo:

```
MaxFlow(G):
    Set f(e)=0 for all e in G  # initialize
    while P=FindPath(s,t,Residule(G,f)) != None:
        f = Augment(f,P)
        Update Residual(G,f)
    return f
```

→ BFS to find shortest s-t path in Gf & augment

Running time: **# Augmentations**: • at every step, flow and capacity remain integers • For path P in G_f, bottleneck(P,f) > 0 implies bottleneck(P,f) ≥ 1 • Each augmentation increases flow by at least 1 • maxflow (hence max #Augmentation) is at most $C = \sum_{e \text{ leaving } s} c(e)$

Time to perform an Augmentation: • G_f has n vertices and at most 2m edges • Finding P, computing bottleneck(P,f), updating G_f • $O(m+n)$

Total Time: $O((m+n) \cdot C)$

Pseudo-polynomial time: The value of C can be exponentially large in the input length (the number of bits required to write down the edge capacity)

ways to achieve poly • Find the maximum bottleneck capacity augmenting path $O(m^2 \log C)$ • Find the shortest augmenting path $O(nm^2)$ BFS

Cuts and cuts capacity $cap(A,B) = \sum$ of capacity of edges leaving A

Th: for any flow f and any s-t cut (A,B), $v(f) = f^{out}(A) - f^{in}(A)$

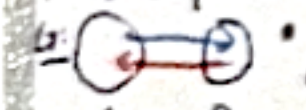
For any flow f and any s-t cut (A,B) $v(f) \le cap(A,B)$

$\Rightarrow \max_f v(f) \le \min_{(A,B)} cap(A,B) \Rightarrow$ max value of any flow ≤ min capacity of any s-t cut

Ford-Fulkerson finds maximum flow: B^* = remaining nodes $V \setminus A^*$

proof: • f = flow returned by FF • A^* - nodes reachable from s in G_f

(A^*, B^*) is a valid cut: $s \in A^*$ by definition, $t \in B^*$ when FF terminates, there's no s-t path in G_f, so $t \notin A^*$



G

• Each blue edge (u,v) must be saturated ⇒ $f^{out}(A^*) = cap(A^*,B^*)$ (otherwise G_f have forward edge, $v \in A^*$)

• Each red edge (v,u) must have zero flow ⇒ $f^{in}(A^*) = 0$

$\therefore v(f) = f^{out}(A^*) - f^{in}(A^*) = cap(A^*, B^*)$

Max Flow Min cut Theorem: In any graph, the value of the maximum flow is equal to the capacity of the minimum cut.

Find min cut: • Run FF to find a max flow f. • construct its residual G_f

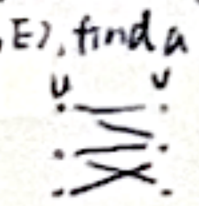
• Let A^* = set of all nodes reachable from s in G_f (BFS)

• $(A^*, V \setminus A^*)$ is a min cut. — cut is defined in G, we look at G_f

Integrality Theorem: if edge capacities are integers, then the max flow computed by FF and its variants are also integral (i.e. the flow on each edge is an integer).

cardinality matching

Bipartite Matching: Given a bipartite graph $G(U \cup V, E)$, find a max



• created a directed flow graph
  • add a source node s and target node t
  • Add edges, all of capacity 1
  • $s \to u$, for each $u \in U$, $v \to t$ for each $v \in V$
  • $u \to v$ for each $(u,v) \in E$

Proof: (Integral flow ⇒ matching) "one to one correspondence"

• Take any flow of f with value k

• The corresponding unique matching M_f = set of edges from U to V with a flow of 1

• since flow of k comes out of s, unit flow must go to k distinct vertices in U.

• From each such vertex in U, unit flow goes to a distinct vertex in V

• uses integrality theorem

Proof: (matching ⇒ integral flow)

• Take a matching $M = \{(u_1,v_1), \dots, (u_k,v_k)\}$ of size k

• construct the corresponding unique flow f_M where...

– edges $s \to u_i$, $u_i \to v_i$ and $v_i \to t$ have flow 1 for all i = 1,...,k

– The rest of the edges have flow 0

This flow has value k

perfect matching = flow with value n where $n = |U| = |V|$

$O((m+n)n)$

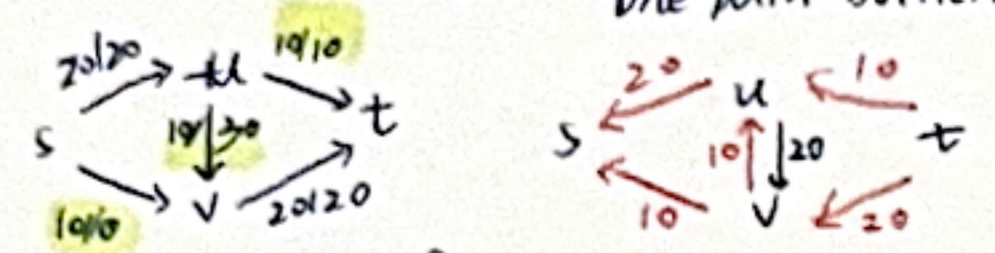
Poly strong: in # of integers but not depend on # of bits

weak: # ops in # bits

Pseudo: polynomial in the value of input integer

Residual Graph

one path bottleneck = 10



Forward edge + f
reverse edge − f

走过的 edge −10
对面的 edge +10

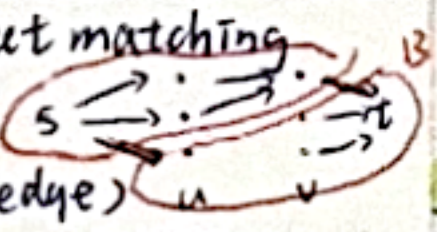
no s-t path bc no outgoing edge from s

证明 valid flow: ① capacity constraint ② Flow conservation

Hall's Marriage Theorem

G has a perfect matching iff $|N(S)| \ge |S|$ for each $S \subseteq V$

Proof: suppose G does not have a perfect matching



Hence max-flow = min cut < n

let (A,B) be the min cut

• can't have any $U \to V$ (∞ capacity edge)

• has unit capacity edge $s \to U \cap B$ and $V \cap A \to t$

$cap(A,B) = |U \cap B| + |V \cap A| < n = |U|$

so $|V \cap A| < |U \cap A|$

But $N(U \cap A) \subseteq V \cap A$ bc, cut doesn't include ∞ edge

so $|N(U \cap A)| \le |V \cap A| < |U \cap A|$ ∎

Edge Disjoint Path

Given a directed graph G = (V,E), two nodes s and t, find the maximum number of edges disjoint $s \to t$ path

Formulation: Assign unit capacity on all edges

Th: 1-1 correspondence between sets of k edge disjoint $s \to t$ path and integral flows of value k.

Proof path → flow • let $\{P_1, \dots, P_k\}$ be a set of k edge disjoint $s \to t$ path • Define flow f where f(e) = 1 whenever $e \in P_i$ for some i, and 0 otherwise • Since paths are edge disjoint flow conservation and capacity constraints are satisfied.

• Unique integral flow of value k

Proof: flow → Path • let f be an integral flow of value k

• k outgoing edges from s have unit flow

• Pick one such edge (s, u_1): By flow conservation, u_1 must have unit outgoing flow (which we haven't set up yet). Pick such an edge and continue building a path until you hit t

• repeat this for the other k−1 edges from s with unit flow.

maximum # of edge disjoint s-t paths := max flow = min cut

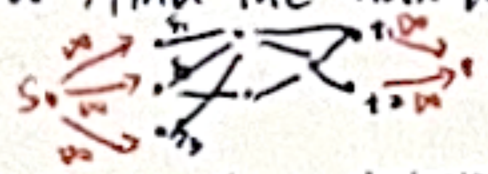
min cut = min # of edges we need to delete to disconnect s from t. (≥ & ≤)

undirected graph: create directed NF by adding each undirected edge in both direction and setting all capacity to 1

Menger's Theorem: In any directed/undirected graph, the maximum # of edge disjoint $s \to t$ path = the min # of edges whose removal disconnects s and t.

Multiple Source/Sinks Given a directed graph G = (V,E) with edge capacities $c: E \to N$, sources $s_1, \dots, s_k$ and sinks $t_1, \dots, t_k$, find the max total flow from sources to sinks

Formulation



Find max flow from s to t

claim: 1-1 correspondence between flows in two Networks.

Circulation Input: Directed graph G = (V,E)
edge capacities $c: E \to N$
Node demands $d: V \to \mathbb{Z}$

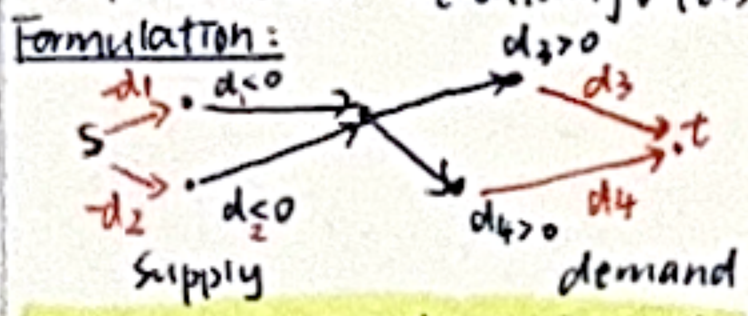
Demand / supply: > 0 take out; < 0 put flow in; = 0

output: Some circulation $f: E \to N$ satisfying

• for each $e \in E$: $0 \le f(e) \le c(e)$

• for each $v \in V$: $\sum_{e \text{ entering } v} f(v) - \sum_{e \text{ leaving } v} f(v) = d(v)$

Formulation:

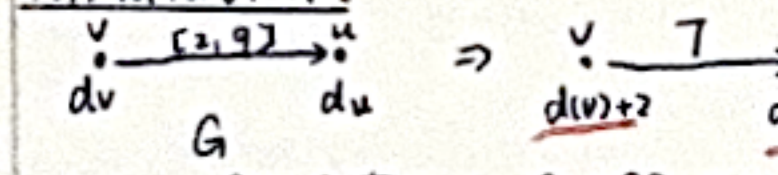


claim: G has circulation iff G' has max flow of value $\sum_{v: d(v)>0} d(v) = \sum_{v: d(v)<0} -d(v)$

Circulation with lower bound

output the: For each $e \in E$, $l(e) \le f(e) \le c(e)$

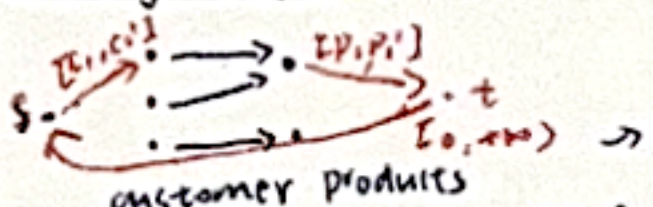
Formulation:



claim: Circulation in G iff circulation in G''

f(e) gives a valid circulation in G iff f(e) − l(e) gives a valid circulation in G'

Survey Design Formulate use circulation with lower Bound,



→ # of questions

claim: feasible survey iff feasible circulation in this network.

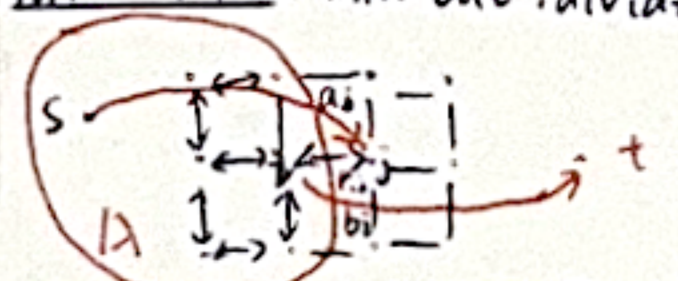
Image Segmentation

Input: • Image (2D array) • a_i = likelihood of pixel i in foreground • b_i = likelihood pixel i in background • $P_{i,j}$: penalty for seperate i,j

output: label each pixel minimize total penalty

$\text{Penalty} = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{(i,j) \in E, |A \cap \{i,j\}| = 1} P_{i,j}$

Formulation: min cut, divide the set of pixels V into (A,B)



Profit maximization Goal: Find a subset of tasks S which, subject to the precedence constraints, maximizes $\text{profits}(S) = \sum_{i \in S} P_i$

Formulate: min cut $S = A \setminus \{s\}$

Goal: cap(A,B) relate to profit(S)

precedence constraints (hard constraint are usually enforced using infinity capacity edges

construction: • Add each $(i,j) \in E$ with infinite capacity

• For each i: if $P_i > 0$, add (s,i) with capacity P_i
if $P_i < 0$, add (i,t) with capacity $-P_i$

• A finite cut exist

• If cap(A,B) is finite, then $A \setminus \{s\}$ is a valid sol

• min cap(A,B) maximize profit($A \setminus \{s\}$)

cap(A,B) = Σ all costs that we take + Σ of profit we didn't take

profit = Σ of all profitable tasks − cap(A,B)

Linear programming

convex: S is convex if $x, y \in S$, $\lambda \in [0,1]$, $\Rightarrow \lambda x + (1-\lambda) y \in S$

vertex: a point which cannot be written as a strict convex combination of any two points in the set.

feasible region of an LP is a convex set.

standard form max $c^T x$

max $c^T x$ s.t. $Ax \le b$, $x \ge 0$

s.t. $a_1^T x \le b_1$ ... $a_m^T x \le b_m$ } m constraint

$x \ge 0$ } n more constraint

≥ ⇒ − ⇒ ≤

= ⇒ ≥ & ≤

min ⇒ − ⇒ max

unconstrained variable ⇒ $x = x' - x''$, $x' \ge 0$, $x'' \ge 0$

Fail when: infeasible or unbounded

opt sol ⇒ there must be a vertex which is opt.

Simplex Algo

let v be any vertex of the feasible region

while there is a neighbor v' of v with better objective value

set v = v'

WC: exponential

slack form

$z = c^T x$, $s = b - Ax$, $s, x \ge 0$ →

max $2x_1 - 3x_2 + 3x_3$
s.t. $x_1 + x_2 - x_3 \le 7$
$-x_1 - x_2 + x_3 \le -7$
$x_1 - 2x_2 + 2x_3 \le 4$
$x_1, x_2, x_3 \ge 0$

⇒ max $2x_1 - 3x_2 + 3x_3$
$x_4 = 7 - x_1 - x_2 + x_3$
$x_5 = -7 + x_1 + x_2 - x_3$
$x_6 = 4 - x_1 + 2x_2 - 2x_3$
$x_1, x_2, \dots x_6 \ge 0$

non basic, basic, slack (= 0 ⇒ tight)

Simplex ①

start with feasible vertex

if b ≥ 0, ⇒ x = 0 is feasible

② entering variable: non basic variable with + coefficient (pivot)

$z = 3x_1 + x_2 + 2x_3$ ($x_4 \ge 0$, $x_2 = x_3 = 0$)

$x_4 = 30 - x_1 - x_2 - 3x_3 \Rightarrow x_1 \le 30$

$x_5 = 24 - 2x_1 - 2x_2 - 5x_3 \Rightarrow x_1 \le 12$

$x_6 = 36 - 4x_1 - x_2 - 2x_3 \Rightarrow x_1 \le 9$ ✓ Tightest obstacle

leaving variable

$\Rightarrow x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}$

$x_4 = 21 - \frac{3}{4}x_2 - \frac{5}{2}x_3 + \frac{x_6}{4}$ 把 x_1 用↓替换

$x_5 =$

$x_1, x_2 \dots x_6 \ge 0$

$z = 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4}$

Final: $z = 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3}$ $x_3 = x_5 = x_6 = 0$

$x_1 = 8 \dots$
$x_2 = 4 \dots$
$x_3 = 18 \dots$

$x_1 = 8$ $x_2 = 4$ $x_3 = 0$

• If enter variable has no upper bound: LP is unbounded

• If pivoting doesn't change const in z: degeneracy, inf loop
↳ prevent by perturbing b by a small random amount in each coord, or by carefully breaking ties among entering and leaving variables eg by smallest index

if b < 0:

LP1: max $c^T x$ s.t. $a_1^T x \le b_1$ ... $x \ge 0$

LP2: $a_1^T x + s_1 = b_1$ ... $x, s \ge 0$

LP3: $a_1^T x + s_1 = b_1$, $-a_2^T x - s_2 = -b_2$ ... $x, s \ge 0$

(a_i > 0:) LP4: min $\sum_i z_i$ s.t. $a_1^T x + s_1 + z_1 = b_1$, $-a_2^T x - s_2 + z_2 = -b_2$ ... $x, s, z \ge 0$

solve LP4 using simplex with the initial basic sol being $x = s = 0$, $z = |b|$ • If opt value is 0, extract a basic feasible x^* from it, use it to solve LP1 using simplex • If opt value for LP4 is > 0, the LP1 is infeasible.

RT: # vertices of a polytope can be exponential in # constraint

Dual: max $-4x - 3y' + 3y'' + 6z$ ⇒ min $2a + 10b - 10c$
$2x - y' + y'' + 3z \le 2$ ⇒ $2a + 3b - 3c \ge -4$
$3x + 2y' - 2y'' + 3z \le 10$ ⇒ $-a + 2b - 2c \ge -3$
$-3x - 2y' + 2y'' - 3z \le -10$ ⇒ $a - 2b + 2c \ge 3$
$3a + 3b - 3c \ge 6$
$a, b, c \ge 0$

max $c^T x$, $Ax \le b$, $x \ge 0$ — min $y^T b$, $y^T A \ge c^T$, $y \ge 0$

If Dual objective = Primal objective ⇒ both opt

weak: $c^T x \le y^T b$ **strong**: $c^T x^* = (y^*)^T b$

$|x| \le 3 \Rightarrow x \le 3, -x \le 3$

min $3|x| + y \Rightarrow t \ge x, t \ge -x$, min $3t + y$

min max$(3x+y, x+2y) \Rightarrow t \ge 3x+y$, $t \ge x+2y$, min t

NW as LP (忘com)
Shortest path as LP