CSC 373 - Algorithm Design

Greedy Method

- Backtracking = recursive brute force
- Divide and Conquer → e.g. mergesort

Interval Scheduling

input : $\{I_1 \ldots I_n\}$, $I_i = [s_i, t_i]$

output : max num of disjoint intervals.

(optimal substructure) → for greedy to work optimally, every solution to a subpb must be part of solution to pb → constraint search space (tree)

alg :

```
S = ∅
for i=1 to n
    add next interval with earliest finish time that doesn't intercept previous one to S.
return S.
```

proof : exchange argument

Let S be sol° returned by alg, S' be OPT sol°.

Let S_{i-1} denote S at start of ith itera°

hyp : S_{i-1} can be extended to OPT sol°.

Assume it is true at start of loop i.

$$S_i = \begin{cases} S_{i-1} & \text{if } I_i \text{ intersects interval in } S_{i-1} \\ S_{i-1} \cup \{I_i\} & \text{o/w} \end{cases}$$

? ⇒ S_i can be extended to OPT sol°

Minimum Spanning Forest

MST(G): connected weighted undirected graph G $\mapsto$ a minimum spanning tree of G

MSF(G): (un)~~connected~~ weighted undirected graph G $\mapsto$ a MSF of G.

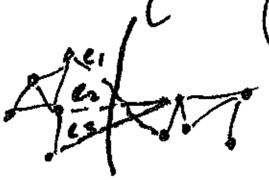
tree = connected forest

Prim's alg (MST)

1. choose arbitrary vertex v and add it to an empty tree T
2. grow the tree by adding the vertex connected by an edge to T, such that that edge is smallest such edge

$\rightarrow O(|V|^2)$

$\rightarrow$ Greedy works bc of cut-property: for a given cut C of a w. graph, the smallest edge in C is in every MST of it.



$\min(e_1, e_2, e_3) \in MST$

means I abuse syntax

Kruskal's alg (MSF)

1. Sort all edges by weight, create a forest F where every node is a root.
2. grow the forest by adding the smallest weighted edge that connects 2 trees in the forest (i.e. avoid cycles)

$\rightarrow O(|V|^2)$

Note [CLRS]: Greedy method yields optimal solution on several struct., one of which is called a Matroid (eg. job sched - cf 2420), which exhibits optimal substructure & greedy choice property –

$\mathcal{M} = (S, I)$ finite set, indep. subsets of S.

Dynamic programming

Shortest path

input : $G = (V, E)$, weighted directed graph and $s, t \in V$

output : a shortest path $s \rightsquigarrow t$ (or all shortest paths starting at source)

Dijkstra's alg: (Greedy) (all edges NON-negative)

```
for i = 1 to n
    d[i] = +∞
    p[i] = Nil
d[s] = 0

H = BuildHeap(V,d)
while H ≠ ∅
    v = H.extractMin()
    for u in Edgelist[v]
        if d[u] > d[v] + w_vu then            ┐ relaxations
            d[u] = d[v] + w_vu                │
            p[u] = v   # add v to the 'path' to u ┘
            H.decreaseKey(u, d[u])   → only predecessor needed
```

Redux

shortest path algs

aka Floyd-Warshall

- Floyd ⟼ all pairs ⟼ $O(|V|^3)$ — neg edges
- Bellman-Ford ⟼ single source ⟼ $O(|V||E|)$ — neg edges

DP

- Dijkstra ⟼ single source ⟼ $O(|V| + |V| \log |V|)$ } Greedy

Longest path

simple path = no repeated vertices –

No subproblem optimality → brute force → NP-complete

Relaxation ("it may seem strange that the term 'relaxation' is used for an operation that tightens an upper bound.") cf [CLRS]

[CLRS] "the process of relaxing an edge (u,v) consists of testing whether we can improve the shortest path to v found so far by going through u, if so update". The use of the term 'relaxation' is historical

Bellman-Ford alg:

(Dynamic prog)

(edges can be negative)

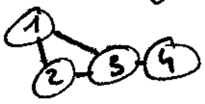
```
for i = 1 to n
    d[i] = +∞
    π[i] = NIL   # predecessor list
d[s] = 0

for i = 1 to n-1
    for each (u,v) in E
        Relax(u,v) # see Dijkstra

checks for negative cycles → ( for each (u,v) in E
    if d[v] > d[u] + w_vu then FAIL
```

All pairs shortest paths

→ single-source shortest paths: predecessor subgraph is a tree of shortest paths from source

→ all-pairs shortest paths: ——— is a forest:

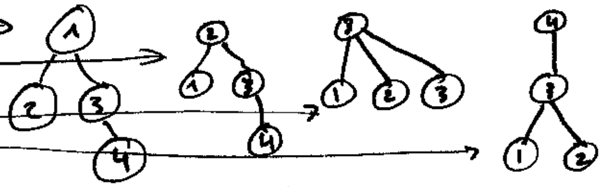
eg. 4 nodes



predecessor matrix:

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & NIL & 1 & 1 & 3 \\ 2 & 2 & NIL & 2 & 3 \\ 3 & 3 & 3 & NIL & 3 \\ 4 & 3 & 3 & 4 & NIL \end{array}$$

⇒



Two DP formulations (recurrences):

Min weight of any path i⇝j that contains at most m vertices

$$l_{ij}^{(m)} = \begin{cases} 0 & \text{if } m=0 \ \& \ i=j \\ \infty & \text{if } m=0 \ \& \ i\neq j \\ \min\limits_{1\le k\le n} \left(l_{ik}^{(m-1)} + w_{kj} \right) & \text{if } m \geq 1 \end{cases}$$

More intuitive $O(|V|^4)$

weight of a shortest path from i⇝j s.t. all intermediate vertices (p-i-j) are in the set {1..k}

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k=0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right) & \text{if } k \geq 1 \end{cases}$$

saves time $O(|V|^3)$

Floyd-Warshall alg:

```
for k = 1 to n
    let $D^{(k)}$ be a new matrix # $D^{(k)} = (d_{ij}^{(k)})$
    for i = 1 to n
        for j = 1 to n
            $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
```

Matrix multiplies (Matrix-chain parenthisation)

e.g. 2 matrices $A_{l \times m} \wedge B_{m \times n} \rightarrow AB_{l \times n}$ needs $l\, n \times m$ computations

3 mats, $A(BC)$ ① $(AB)C$ ② with $A_{100\times 5}\, B_{5\times 100}\, C_{100\times 1000}$

A   BC   AB   C

① → 550000 comps

② → 1001000 comps

input : $M_1 \dots M_n$, $M_{i\, d_i d_{i+1}}$

output : min. num of comps

Let $m_{1,n}$ be OPT sol°, then $m_{1,n} = \min\limits_{1 \leq k \leq n-1} (m_{1,k} + d_1 d_{k+1} d_{n+1} + m_{k+1,n})$

Use DP : keep ~~ka~~ 2Dim array of $m_{i,j}$:

```
alg: for i = 1 to n-1
        for j = 1 to n  ←  { $m_{ij} = +\infty$
                             if $i=j$ then $m_{ij} = 0$
            for k ~~f~~= i to j-1
                if $x = m_{i,k} + m_{k+1,j} + d_i d_{k+1} d_{j+1} < m_{ij}$
                    $m_{ij} = x$
```

Flow Networks

Input : G, dir. weight. conn. ; $s, t \in V$

Output : Max flow ($\Leftrightarrow$ min cut)

Def: a flow network is a graph $G = (V,E)$,

(1) such that every edge has a nonnegative capacity $c(u,v) \geq 0$ (as a convention 0 is when there is no edge) ;

(2) such that all edges are directed and there is no self loops nor recurrent connections ;

(3) and such that s and t are distinguished as source and sink resp.

a flow is a func.

$f : E \rightarrow \mathbb{R}^+$

$f(e) \in [0, e.w]$ (1)

$\forall v \in V, v \neq s \wedge v \neq t \Rightarrow$
$f^{in}(v) = f^{out}(v)$ (2)

capacity constraint (1)

$\forall e \in E, f(e) \in [0, e.w]$

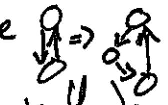
flow conservation (2)

$\forall v \in V, v \neq s,t \Rightarrow f^{in}(v) = f^{out}(v)$

Maximum flow pb: determine maximum flow $f^{Max} \begin{cases} = \max f^{out}(s) \\ = \max f^{in}(t) \end{cases}$

→ Any directed graph (→ w/ at least 1 source & 1 sink) s.t. (1) holds can be reduced to a flow network:

[CLRS] . antiparallel edges (2) if there are connection both ways, add an extra node



. supersource/sink (3) if there are multiple sources(/sinks), add a single one that feeds in(/out to) them.

Ford-Fulkerson Method

→ Residual networks

Def: The residual net of a flow net G with flow f, denoted G_f, is the graph induced by the residual capacity c_f of each edge of G. The residual capacity is constructed by (1) $c_f(u,v) = c(u,v) - f(u,v)$ (the capacity remaining/unused by f) and (2) adding an edge in the other direction $c_f(v,u) = f(u,v)$ (the flow through $u \to v$) to allow an alg to cancel out some existing flow.

Def: Once given a flow f' in the residual net G_f we can augment flow f as such: $(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E \\ 0 & \text{o/w} \end{cases}$

Def: A cut (S,T) in flow net G is a cut such that $s \in S$, $t \in T$. The net flow across the cut is $\boxed{f(S,T) = \sum_{u \in S}\sum_{v \in T} f(u,v) - \sum_{u \in S}\sum_{v \in T} f(v,u)}$ (in the residual net, we allow recurrent edges — while the capacity of a cut is just $\boxed{c(S,T) = \sum_{u \in S}\sum_{v \in T} c(u,v)}$ ← upper bound on

Thm: Max-Flow Min-Cut the max flow in G is equal to the cut c_{min} of G, s.t. $\forall$ cutset C of G, $c_{min} = \min(c)_{c \in C}$. ($f$ is max iff G_f has no augmenting paths, a path that would augment f: $\forall f'$ of G_f, $(f \uparrow f') \leq f$)

Ford-Ferkuson method:

implementation of path-finding is not given.

```
for each (u,v) ∈ E
    (u,v).f = 0
while ∃ a path p from s ⇝ t in G_f   # while an augmenting path remains, do f↑f'
    c_f(p) = min_{(u,v) in p} (c_f(u,v))   # residual capacity of p
```

```
for each (u,v) in p
    if (u,v) ∈ E
        (u,v).f = (u,v).f + c_f(p)
    else (v,u).f = (v,u).f - c_f(p)
```

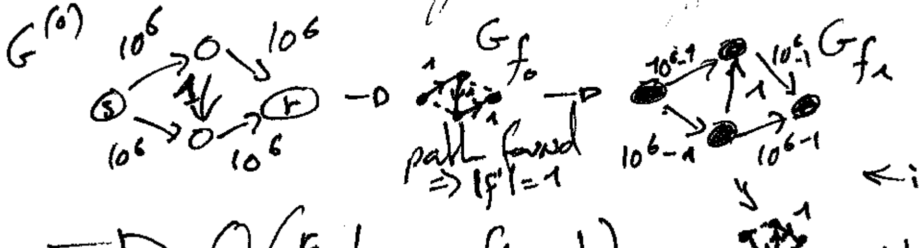
add residual cap. of augmenting path in G_f to corresponding edges in original flow net

if an edge in p is in the other direction than one in G, that means it is cancelling out existing flow.

→ running time will depend on implementation of finding an augmenting path in the residual net.

Thm: Integral flow: if capacities are integers, then max flow ∈ N.
(Ford-Felkerson Might not converge o/w)

Naive Ford-Ferkeson alg: worst case could be bad, eg.



reminds me of Braess's paradox CSC200

← imagine you keep finding the worst paths ⇒ 10^6 iterations

→ O(E |max flow|)

Edmonds-Karp alg: Use BFS to find a shortest (unweighted) path in the (partial) residual net. → $O(|V||E|^2)$

proof:
- correctness (of Ford-Felkerson) ⇒ end up w/ min cut if all capacities ∈ N
- correctness (of Edmonds-Karp) follows from Fo-Felk.
- termination (of ——):
  - call the edge w/ min capacity in an augmenting (also shortest) path critical (i.e. equal to |f'|)
  - lemma: any edge in V×V can become critical n/2 times.

i.e. it is across the min cut of path p.

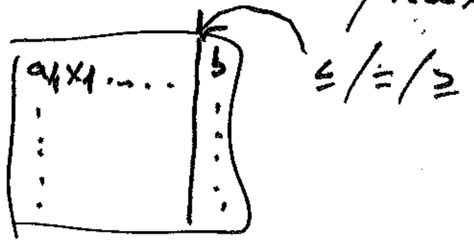
proof: once augmented, the critical edge (u,v) disappears from the residual net and can only reappear if (v,u) ever appears on an augmenting path.
In which case, $s \overset{p_1^{(2)}}{\rightsquigarrow} v \xrightarrow{+1} u \overset{p_2^{(2)}}{\rightsquigarrow} t$ is shortest ⇒ $s \rightsquigarrow u = s \rightsquigarrow v + 1$ shortest
But from this lemma (*) the shortest path from $s \overset{p_1'}{\rightsquigarrow} v$ is greater or equal to the shortest path $s \overset{p_1^{(1)}+1}{\rightsquigarrow} v$ found previously. Actually if (u,v) reappears then $s \overset{p_2^{(3)}}{\rightsquigarrow} u = s \overset{p_2^{(1)}}{\rightsquigarrow} u + 2$

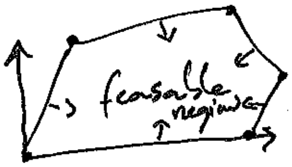
lemma (*): at each iteration of Ed-Karp, shortest paths from source in residual net monotonically increase
proof: [redacted]

suppose v contradic

Linear & Integer Programming

LP
- variables $(x_i) \in \mathbb{R}$
- objective $\sum c_i x_i$, $c_i \in \mathbb{R}$
- constraints: min / max





2D-polytope

IP → NP-hand

Def: Feasable region: all values of x_i's that satisfy every constraint.

- empty → infeasable LP
- unbounded → unbounded LP
- bounded → normal LP

Algs
- simplex
- other algs

Applications: network flow

→ Given network $N = (V, E)$ capacities $c(e), \forall e \in E$

→ Create linear program with variables: $f_e, \forall e \in E$

obj. f: $\max \sum_{s,v \in E} f_{s,v}$

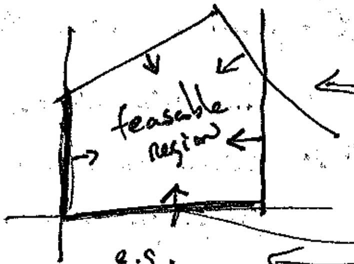
→ Constraints:

$\forall e \in E, \; f_e \leq c(e)$ | capacity constraint

$f_e \geq 0$

9

→ formulating ~~as~~ LP:



e.g.

2D convex polytope (i.e. convex polygon) 2-polytope

MAX/MIN obj func

under linear constraints
1 ———— $\geq$
2 ———— $\leq$
3 ————

with vars $\geq$?

NON strict inequalities

→

Max/min $c^T x$

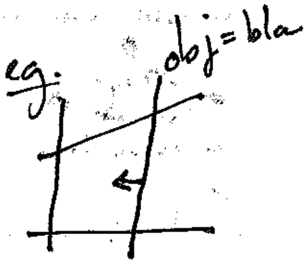
under $Ax \leq b$

with $x \geq d$

→ simplex = convex n-polytope (i.e. the convex hull of its n+1 vertices) → cf CSC2420

Simplex alg: some form of gaußian elimination on a simplex. If the matrix of constraints respects certain properties (e.g. positive semi-definite) then the runtime is poly, but exp. in general

finds a local optimum, which, by linearity is also global min



feasable region

- empty ⇒ infeasable LP
- unbounded ⇒ if obj func 'closes' the polytope then unbounded LP (otherwise infeasable)
- bounded ⇒ bounded LP

→ Approximation algorithms

(see CSC2420) { approx alg: KNOWN approx ratio
approx scheme: also takes ε as input an can thus tradeoff runtime for precision

FPTAS (fully polytime ap.sch. means also poly in $(\frac{1}{\varepsilon})$)

→ hardness of approximation: NP-complete pbs have different possible approximations (I guess the polytime reduction itself hides this)

e.g.

- MIN vertex cover & max matching: have cst approx of 2.
- knapsack (0-1): has approx of $(1+\varepsilon)$ in $O(n^3/\varepsilon)$
- TSP: NO FPTAS known! (ORP=NP)

"weakly NPC"

"strongly NPC"

weak NP-completeness

(vs) strong

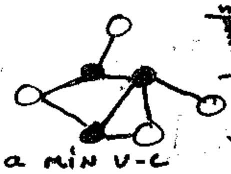
NO pseudo-polytime alg

knapsack admits a pseudo-polytime alg. (polytime in the size [as an integer, i.e. ~~logarithm~~ of the numeric value encoded in binary] of input, but not in its representation as a bin. string) (a way to measure the real length of the input) that uses DP.

→ vertex cover example

Side-note : graph pb 'dualities' —Yes→ • primal-dual thm (i.e. one lower-bounds the other)

—NO→ • dual space in linAlg (inf. function spc)

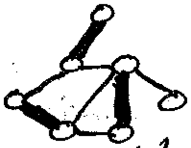


a min v-c

Vertex cover : ↑ lower bounds

set of vertices C such that:
$\forall e \in E$, e is incident to at least 1 $v \in C$

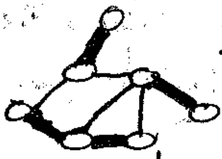
Matching (indep. edge set):
set of edges M such that:
$\forall v \in V$, v is incident to at most 1 $e \in M$



a max match

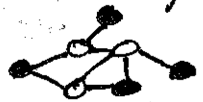
edge cover : ↑ lower bounds

set of edges C such that:
$\forall v \in V$, v is incident to at least 1 $e \in C$



a min edge c.

indep. set :
set of vertices I such that:
$\forall e \in E$, e is incident to at most 1 $v \in I$



a max i.s.

By primal-dual:

- The max matching (dual→max) gives a lower bound on the min vertex cover (primal→min)

⚠ the solutions are not equal in general ⇒ duality gap

these 2 dualities share some symmetry —o— (VS) o—o

1) Naive approx : pick an arbitrary edge (u,v) from the graph and delete all edges incident to (u) or (v), repeat until the graph ~~is empty~~ has no more edges (the vertices picked out will be a VC).

↳ provides a 2 - approx alg

proof : consider all the edges picked out, call that set A. then A is a maximal matching, thus $|A| \le |OPT|$. Since we are always collecting 2 vertices, we also know that the size of the output is $|C| = 2|A|$. by using our lower bound, $|C| = 2|A| \le 2\,|OPT|$

$$\Rightarrow \frac{|C|}{|OPT|} \le 2$$

Note :
a maximal matching is a matching M such that adding any edge to M that isn't already in it would make M non-matching
↳ (local optimum)

→ if you are wondering, the minimum maximal matching (global optimum) is NP - complete

2) LP approx :

IP ↗

obj : $\min \sum_{i=0}^{n} x_i$ — corresp. to each vertex
cons:
|V| lines ⇒ $0 \le x_i \le 1$ for each vertex
|E| lines ⇒ $x_i + x_j \ge 1$ for each edge(i,j)
with: $x \in \mathbb{N}$

- set up an equivalent IP and relax it.
- solve the relaxed LP
- Create a cover by $C = \{ v_i \in V \mid x_i^* \ge 1/2 \}$ ← sol to LP

↳ gives a cover bc constraint $x_i + x_j \ge 1$ ⇒ at least one of x_i^* or x_j^* will be $\ge 1/2$

↳ also provides a 2-approx alg