Graph Theory

Fundamentals:

Graph $G = (V, E)$:

Finite nonempty set $V = (v_1, \ldots, v_m)$ of <u>nodes</u> or <u>vertices</u>

$E = \{e_1, \ldots, e_n\}$ each element is a subset of V of size 2

Elements called <u>edges</u>.

Bipartite

G is bipartite if there is a partition of V into disjoint sets V_1 and V_2 such that each edge joins a node in V_1 to a node in V_2.

eg assignment (Workers i, Jobs j)

Simple graph

Unless otherwise stated assume:

no <u>parallel edges</u>, ie there is at most one edge between any pair of vertices

no <u>loops</u>; ie no edges of the form (i, i)

Such graphs called <u>simple</u>.

Adjacency

$e_v \equiv (i, j)$ v meets or is incident to $i \in V, j \in V$;

i, j are endpoints of e_v.

Can represent graph by vertex-edge incidence matrix $A = (a_{ij})_{m \times n}$

$$a_{ij} = \begin{cases} 1 & \text{if } e_j \text{ is incident to node } i \\ 0 & \text{otherwise.} \end{cases}$$

Eg: Matching problem.

No. of ones in each col = 2.

No. of ones in ~~each~~ row i = no. of edges incident to node i

= degree $\delta(i)$ of ~~vert~~ node i.

Degree of graph = $\min \{\delta(i) : i \in V\}$.

Graph is called complete if it contains all edges,

so degree of every ~~edge~~ node is $n-1$.

Written K_n.

Vertex-vertex adjacency matrix $m \times n$ $A' = (a'_{ij})$

$$a'_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Paths, cycles

A node sequence $v_0, v_1, \dots, v_k$, $k \geq 1$ is a v_0-v_k walk if $(v_{i-1}, v_i) \in E$ for $i = 1, \dots, k$.

v_0 : origin v_k : destination $\{v_1, \dots, v_{k-1}\}$: intermediate nodes

length of walk is k.

Can also represent as $(e_1, \dots, e_k)$ where $e_i = (v_{i-1}, v_i)$

A walk is called a path if there are no node repetitions.

A v_0-v_k walk is called closed if $v_k = v_0$.

A closed walk is a cycle (circuit) if $k \geq 3$ and $v_0, v_1, \dots, v_{k-1}$ is a path.

A graph is acyclic if it contains no cycles.

Prop If the simple graph G has degree $\delta \geq k \geq 1$ then G has a path of length k.

Algorithm to find such a path:

$i = 1$

Select $v_0 \in V$, let $S = \{v_0\}$

(*) Select $v_i \in V \setminus S$ such that $\{v_{i-1}, v_i\} \in E$ ← arguably can do this.

let $S \leftarrow S \cup \{v_i\}$

$i \leftarrow i+1$

If $i < k+1$ go to (*), otherwise stop. $(v_0, v_1, \dots, v_k)$ is a path of length k //

(Delay to after connected stuff)

Thm G is bipartite $\iff$ G contains no odd cycles

Proof Exercise. (Prove $\Rightarrow$, start $\Leftarrow$ by partitioning according to distance from some vertex)

Connected

$u, v \in V$ are connected in $G = (V, E)$ if $\exists$ a (u,v)-path in G

This binary relation partitions V into the components of G.

G is connected if no of components $= 1$.

Subgraph

For $U \subseteq V$, let $E(U) = \{(i,j) : i \in U, j \in U, (i,j) \in E\}$

$E(U)$ is set of edges with both endpoints in U.

If $V' \subseteq V$, $E' \subseteq E(V')$ then $G' = (V', E')$ is a subgraph of G

G' is the subgraph induced by V' if $E' = E(V')$.

G' is a spanning subgraph if $V' = V$.

Trees

An acyclic graph is called a forest.

A connected forest is a tree.

A spanning tree of G is a spanning subgraph that is a tree.

Bondy & Murty

(Diestel & Thomassian)

Prop Let $G = (V, E)$ be a graph on n nodes. The following statements are equivalent

1) G is a tree
2) There is a unique path between each pair of nodes
3) G contains $n-1$ edges and is connected
4) G contains $n-1$ edges and is acyclic
5) G is connected and acyclic.

Corollary

a) If $G = (V, E)$ is a tree and $e' \notin E$ then $G' = (V, E \cup \{e'\})$ contains exactly one cycle (Fundamental cycle)

b) If C is the edge set of the cycle of G' and $e^* \in C \setminus \{e'\}$, then $C^* = (V, E \cup \{e'\} \setminus \{e^*\})$ also is a tree.

Digraph, A directed graph or digraph $D = (V, A)$ consists of a finite nonempty set $V = \{1, \dots, m\}$ of vertices/nodes and a set $A = \{e_1, e_2, \dots, e_n\}$ of whose elements are ordered subsets of V of size 2 called arcs.

Node-arc incidence matrix of a digraph D with m nodes and n arcs is the $m \times n$ matrix A with

$$a_{ij} = \begin{cases} 1 & \text{if } e_j = (k, i) \text{ for some } k \in V \setminus \{i\} \\ -1 & \text{if } e_j = (i, k) \text{ for some } k \in V \setminus \{i\} \\ 0 & \text{otherwise} \end{cases}$$

NB: $e^T A = 0$, so rows of A are linearly dependent.

Directed walk, directed path, directed cycle.

Shortest path problem

Have a digraph $D = (V, A)$,

weight function $w : A \to \mathbb{R}$

designated origin and destination nodes 1 and n, respectively.

Weight of a 1-n path is sum of arc weights over all arcs in path.

Problem: find a 1-n path of minimum weight.

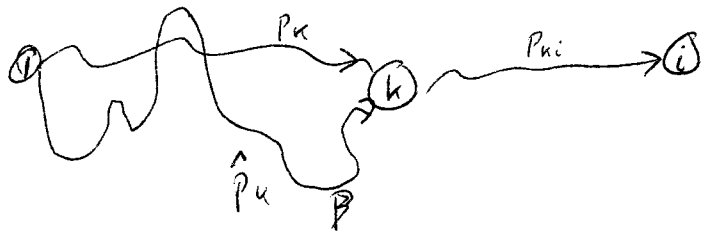
First: assume all weights w are nonnegative

Algorithm will solve slightly more general problem of finding shortest paths from node 1 to all other nodes.

Proposition Suppose k is an intermediate node on a minimum-weight 1-i path p.
Then the 1-k subpath p_k of p_i is a minimum weight 1-k path.

Proof Let $w(p)$ be weight of path p. Proof is by contradiction.

Let $\hat{p}_k$ be ~~another~~ a 1-k path with $w(\hat{p}_k) < w(p_k)$



Let $p_i = (p_k, p_{ki})$. Then $\hat{p}_i := (\hat{p}_k, p_{ki})$ is a 1-i walk and

$w(\hat{p}_i) = w(\hat{p}_k) + w(p_{ki}) < w(p_k) + w(p_{ki}) = w(p_i)$ ✗ because $\hat{p}_i$ contains

Let $g(i)$ be weight of minimum 1-i path, and define $g(1) = 0$.

Dijkstra's Minimum-Weight Path Algorithm

Step 1 (Initialization).

$g(1) = 0$, $U = \{1\}$, $h(j) = \begin{cases} w_{1j} & \text{if } (1,j) \in A \\ \infty & \text{otherwise} \end{cases}$

Step 2 (Update U) Let $i = \arg\left(\min_{j \in V \setminus U} h(j)\right)$

(If minimum is not unique, select any i that achieves the minimum)

Set $U \leftarrow U \cup \{i\}$, $g(i) = h(i)$. If $U = V$, STOP.

Step 3 (Update h)

For all $j \in V \setminus U$ with $(i,j) \in A$, $h(j) \leftarrow \min\{g(i) + w_{ij}, h(j)\}$.

Return to step 2.

Algorithm only determines weights.

To determine path, keep record of the node before j on the path that has weight $h(j)$

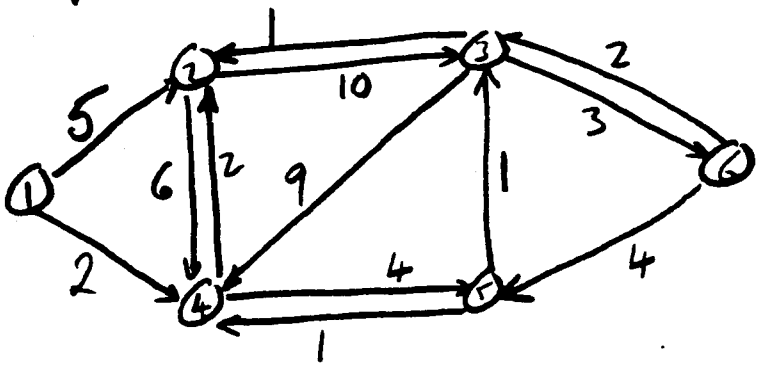
Go over handout.

Example of Dijkstra's Algorithm

67.617 COMB OPT

[redacted]

Find shortest (1,6) path



Init:

$X = \{1\}$

$v_1 = 0$ $l_2 = 5$, $l_3 = \infty$, $l_4 = 2$, $l_5 = \infty$, $l_6 = \infty$

Loop

$X = \{1, 4\}$

$v_4 = 2$ $l_2 = \min\{l_2, v_4 + c_{42}\} = \min\{5, 2+2\} = 4$

predecessor(4) = 1 $l_3 = \min\{\infty, 2+\infty\} = \infty$ $l_5 = \min\{\infty, 2+4\} = 6$

$l_6 = \min\{\infty, 2+\infty\} = \infty$

$X = \{1, 4, 2\}$

$v_2 = 4$ $l_3 = \min\{\infty, 4+10\} = 14$ $l_5 = \min\{6, 4+\infty\} = 6$

predecessor(2) = 4 $l_6 = \min\{\infty, 4+\infty\} = \infty$

$X = \{1, 4, 2, 5\}$

$v_5 = 6$ $l_3 = \min\{14, 6+1\} = 7$ $l_6 = \min\{\infty, 6+\infty\} = \infty$

predecessor(5) = 4

$X = \{1, 4, 2, 5, 3\}$

$v_3 = 7$ $l_6 = \min\{\infty, 7+3\} = 10$ predecessor(3) = 5

$X = \{1, 4, 2, 5, 3, 6\}$

$v_6 = 10$ $\bar{X} = \phi$ STOP

predecessor(6) = 3

Best path (trace backwards): 1, 4, 5, 3, 6

Theorem Dijkstra's algorithm is correct.

Proof By induction.

Inductive hypothesis:

After t passes through Step 3, $g(j)$ is correct for all $j \in U$, and $h(j)$ is the weight of a minimum weight 1-j path restricted to having intermediate nodes in U.

True initially when $U = \{1\}$.

From induction hypothesis, $g(j) \leq h(j) \quad \forall j \in V \setminus U$.

Suppose $g(i) \leq h(i)$ (i as in Step 2).

Then the min-wt 1-i path contains some node in $V \setminus U$

Let k be first such node

Then by the Proposition, the subpath from 1-k must be a min weight 1-k path so its weight is $g(k)$. But this 1-k path contains only nodes in U. $\therefore h(k) = g(k) \leq g(i) < h(i)$, ✗ to choice of i.

Thus $g(i) = h(i)$

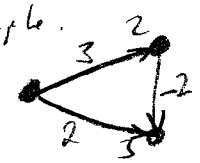
For $j \notin U \cup \{i\}$: $h(j)$ now represents the weight of a minimum weight 1-j path with intermediate nodes in $U \cup \{i\}$, since any such path either remains as before or contains i as its last node, in which case

$h(j) = g(i) + w_{ij}$.

//

Now drop assumption that all edge weights are nonnegative.

Use Bellman-Ford Algorithm: (Don't set $g(j)$ until final; motivate this by an example.



Bellman-Ford Shortest Path Algorithm

Step 1 (Initialization):

$h^0(1) = 0$, $h^0(j) = \infty$ for $j \in V \setminus \{1\}$, $k = 1$

Step 2: (Update h):

For all $j \in V$,

$$h^k(j) = \min \left\{ \min_{i:(i,j)\in A} \left(w_{ij} + h^{k-1}(i) \right), h^{k-1}(j) \right\}$$

Step 3: (Iterate)

If $h^k(j) = h^{k-1}(j)$ for all $j \in V$, then $g(j) = h^k(j)$ for all $j \in V$.

Otherwise, if $k < m$, $k \leftarrow k+1$ and return to step 2

if $k = m$, STOP: graph contains a negative length cycle.

(Problem 18.9)

At kth stage, the algorithm calculates shortest ~~path~~ walk using at most k edges.

m steps. At each step: an addition for each of n arcs, for each node take minimum over n numbers.

$O(m(n+m))$ $O(n^3)$ in complete case.

(n: arcs, m: vertices)

Define this.

Dijkstra: $O(n^2)$.

The shortest path problem can be expressed as an LP

Let $x_{ij} = \begin{cases} 1 & \text{if use edge } (i,j) \\ 0 & \text{o/w} \end{cases}$

Let w_{ij} = weight of arc (i,j)

For all nodes other than 1 and n, if path enters vertex it must leave vertex.

For vertex 1, path only leaves vertex
n enters

So get constraints

$$A x = \begin{pmatrix} -1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \leftarrow \text{vertex } 1 \quad \leftarrow \text{vertex } n$$

(NB: these constraints only constrain x to be a walk.)

$a_{ij} = \begin{cases} -1 & \text{if arc } j \text{ leaves vertex } i \\ +1 & \text{if arc } j \text{ enters vertex } i \\ 0 & \text{otherwise} \end{cases}$ ← Vertex-arc incidence matrix

Also have constraints x_{ij} binary

Objective function $\min \sum w_{ij} x_{ij}$

In fact, because of the structure of A and the RHS,

optimal solution to

$$\min \sum w_{ij} x_{ij}$$
$$\text{s.t. } Ax = \begin{pmatrix} -1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \quad (P)$$
$$x \geq 0$$

is optimal soln to $\min \sum w_{ij} x_{ij}$, $Ax = \begin{pmatrix} -1 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$, $x = 0$ or 1

What is dual to (P)?

Associate variables π_i with vertices. (potentials).

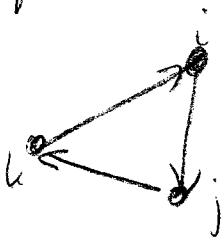
$$\max \quad \pi_n - \pi_1$$

$$\text{s.t.} \quad \pi_j - \pi_i \leq w_{ij} \qquad \forall \text{ arcs } (i,j).$$

For any optimal solution, $\pi_j - \pi_1 =$ length of shortest path from node 1 to node j.
($\Rightarrow$ by complementary slackness)

What happens if we have a negative length cycle?

Eg



$$w_{ij} + w_{jk} + w_{ki} < 0$$

Consider adding the dual constraints for these three arcs:

$$\begin{aligned} \pi_j - \pi_i &\leq w_{ij} \\ \pi_k - \pi_j &\leq w_{jk} \\ \pi_i - \pi_k &\leq w_{ki} \end{aligned}$$

$$0 \leq w_{ij} + w_{jk} + w_{ki} < 0$$

So (D) is infeasible.

For optimal solution

Maximum flow in a network.

Given a digraph $D = (V, A)$, have two distinguished nodes s and t — source & sink respectively. Capacities b_{ij} on edge (i,j)

Liquid enters the network only at s and leaves only at t.

Objective is to maximize flow ~~at~~ in the network.

Let x_{ij} = flow from i to j on arc (i,j).

Let v = flow in network (i.e. liquid entering s).

Flow conservation constraints:

$$Ax = \begin{pmatrix} -v \\ 0 \\ \vdots \\ 0 \\ v \end{pmatrix} \begin{matrix} \leftarrow \text{node } s \\ \\ \\ \\ \leftarrow \text{node } t \end{matrix}$$

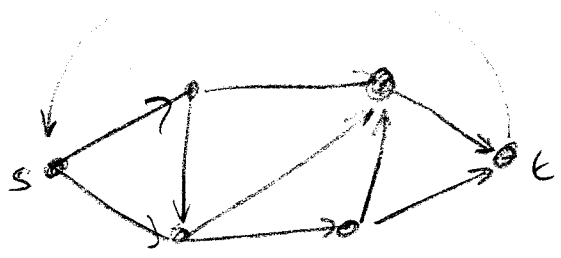
Let $d = \begin{pmatrix} +1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix}$

So max flow problem can be written:

$$\begin{aligned} \max \quad & v \\ Ax + dv &= 0 \\ x &\leq b \quad \leftarrow \text{capacities of edges} \\ x &\geq 0. \end{aligned} \qquad (P)$$

Replace b by $+\infty$ when b_{ij} is ∞ in LP course.

(Could be regarded as adding an extra arc from t to s and maximizing flow on that arc:

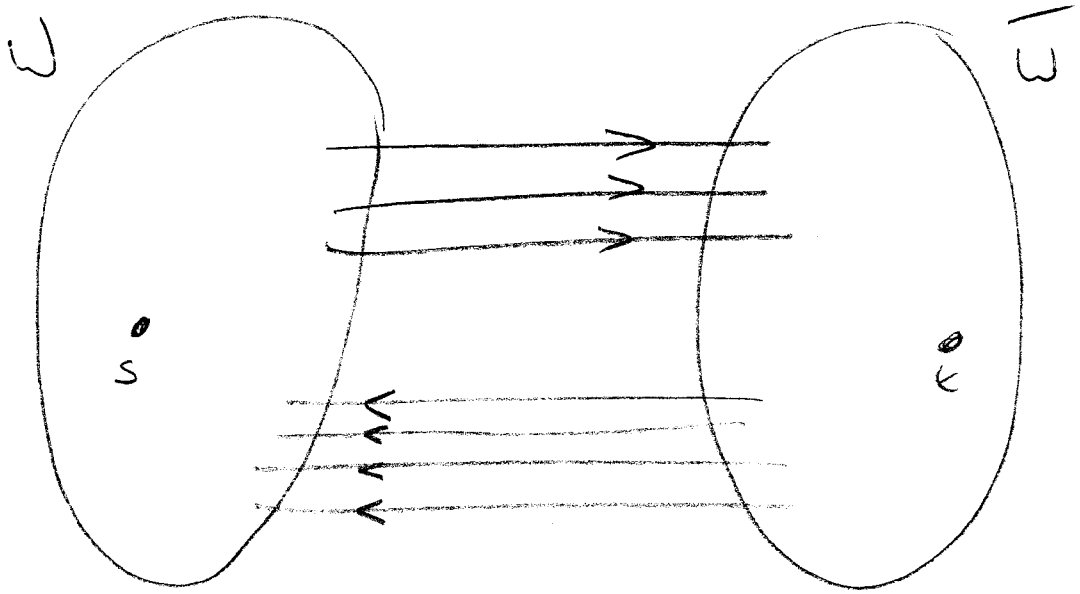


)

Cuts

An s-t cut is a partition $(W, \bar{W})$ of the nodes of V into sets W and $\bar{W}$ such that $s \in W$ and $t \in \bar{W}$. The capacity of an s-t cut is

$$c(W, \bar{W}) := \sum_{\substack{(i,j) \in E: \\ i \in W, j \in \bar{W}}} b_{ij} =: \delta^+(W)$$



Clearly, all flow from s to t must pass through the edges $\delta^+(W)$.

$\therefore$ max flow $\leq c(W, \bar{W})$ $\forall$ s-t cuts $(W, \bar{W})$.

Thm The maximum flow from s to t is bounded above by the capacity of any (s,t)-cut.

Dual to maximum flow problem is

$$\min \sum_{(i,j)\in A} b_{ij} y_{ij}$$

$$\pi_j - \pi_i + y_{ij} \geq 0 \quad \forall (i,j) \in A \qquad (D)$$

$$\pi_s - \pi_t = 1$$

$$y_{ij} \geq 0$$

Theorem Every (s,t)-cut determines a feasible solution with cost $C(W,\bar{W})$ to the dual of the max-flow as follows:

$$y_{ij} = \begin{cases} 1 & \text{if } i \in W,\ j \in \bar{W} \\ 0 & \text{o/w} \end{cases}$$

$$\pi_i = \begin{cases} 1 & \text{if } i \in W \\ 0 & \text{o/w} \end{cases}.$$

Proof Check this is feasible: Clearly $\pi_s = 1$, $\pi_t = 0$, so $\pi_s - \pi_t = 1$. Also, $y_{ij} \geq 0$.

Four cases: i) $i \in W$, $j \in \bar{W}$:

$$\pi_j - \pi_i + y_{ij} = 0 - 1 + 1 = 0 \quad \checkmark$$

ii) $i \in W$, $j \in W$:

$$\pi_j - \pi_i + y_{ij} = 1 - 1 + 0 = 0 \quad \checkmark$$

iii) $i \in \bar{W}$, $j \in W$:

$$\pi_j - \pi_i + y_{ij} = 1 - 0 + 0 = 1 \quad \checkmark$$

iv) $i \in \bar{W}$, $j \in \bar{W}$:

$$\pi_j - \pi_i + y_{ij} = 0 - 0 + 0 = 0 \quad \checkmark$$

Objective function:

$$\sum_{(i,j)\in A} b_{ij}\, y_{ij} = \sum_{\substack{(i,j)\in A: \\ i\in W,\, j\in \bar{W}}} b_{ij} = C(W,\bar{W}) \qquad //.$$

Theorem The value of the maximum flow equals the capacity of the minimum cut, and a flow x and cut $(W,\bar{W})$ are jointly optimal if and only if

$$(*) \qquad x_{ij} = 0 \quad \forall (i,j)\in A \text{ with } i\in \bar{W},\ j\in W$$
$$x_{ij} = b_{ij} \quad \forall (i,j)\in A \text{ with } i\in W,\ j\in \bar{W}$$

(Can not prove using LP $\Rightarrow$ don't know optimal dual solution corresponds to a cut.)

Proof Already shown

value of every flow $\leq$ value of every cut

(max) (min)

We'll give an algorithm to show that a cut of value v can be constructed from any max flow of value v.

Conditions $(*)$ are the complementary slackness conditions:

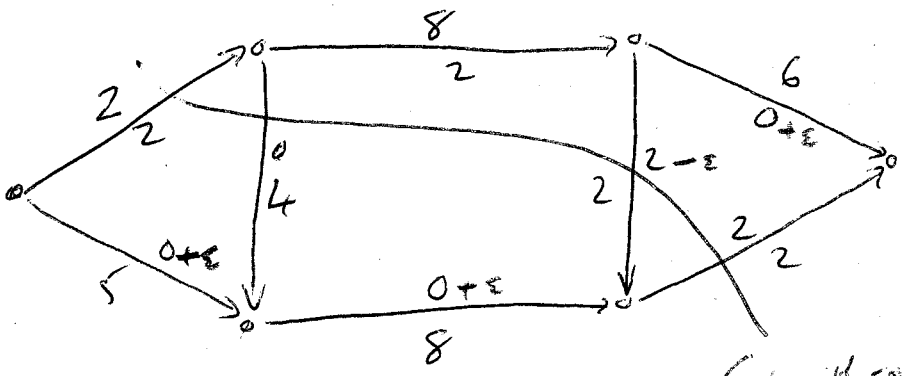
Only case when have strict inequality in dual constraints is when $i\in\bar{W}, j\in W$

$$\Rightarrow x_{ij} = 0 \text{ if } i\in\bar{W},\ j\in W$$

$$y_{ij} > 0 \text{ if } i\in W,\ j\in\bar{W}$$

$$\Rightarrow x_{ij} = b_{ij} \text{ if } i\in W,\ j\in\bar{W}. \qquad //.$$

An example of augmenting path algorithm.



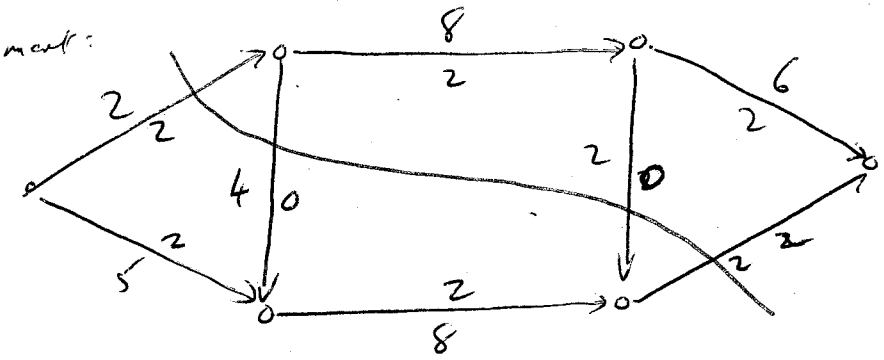
Blue numbers: capacities c_{ij}

Red numbers: flows x_{ij}

Pencil: augmenting path

Cut with capacity 4.
Max. flow of 2 before augmentation,
flow of 4 after augmentation.

Augment:



Flow of size 6.
Cut of capacity 4.

Defn Given a digraph $D = (V, A)$, ~~s≠t~~, source s, sink t, feasible s-t flow x: an **augmenting path** P is a path from s to t in the undirected graph G given by ignoring arc directions with the following properties:

i) For every arc $(i,j) \in E$ that is traversed by P in the forward direction (a forward arc), we have $x_{ij} < b_{ij}$.

ii) For every arc $(j,i) \in E$ that is traversed by P in the reverse direction (a backward arc), we have $x_{ij} > 0$.

If P is an augmenting path then we can increase flow from s to t while maintaining flow conservation at every node by increasing the flow on every forward arc of P and decreasing it along every backward arc.

Can increase until reach capacity of some forward arc, or empty a backward arc.

So max amount of flow augmentation possible along P is

$$\delta := \min_{\text{arcs of } P} \begin{cases} b_{ij} - x_{ij} & \text{along a forward arc} \\ x_{ji} & \text{along a backward arc.} \end{cases}$$

Theorem A feasible flow x is not maximum if there is an augmenting path with respect to it.

Ford-Fulkerson algorithm for solving max-flow problem

Call an arc (i,j) saturated if $x_{ij} = b_{ij}$

Call a vertex i reachable if $\exists$ augmenting path from s to i wrt x.

Have a set W of vertices we've already decided are reachable; each vertex in W has a label: the previous vertex in the path, max possible of flow augmentation possible to i along the path. $\equiv (L1(i), L2(i))$=last

Have a subset LIST of W containing those vertices in W not yet scanned.

For a vertex $i \in$ LIST, the operation of scanning i consists of finding which neighbours $\notin W$ of i ~~are reach can be reached along it~~ the ~~path~~ augmenting path to P can be extended to.

Complete alg.

Step 1: Initialize: $x = 0$

Step 2: Search for augmenting path:

Initialize: $W = \{s\}$, LIST $= \{s\}$, label$(s) = (\blacksquare, 0, \infty)$

While LIST $\neq \phi$:

Pick any vertex $i \in$ LIST. Set LIST $\leftarrow$ LIST $\setminus \{i\}$.

Scan i: For all j such that $(i,j) \in A$, $x_{ij} < b_{ij}$ and $j \notin W$,

$W \leftarrow W \cup \{j\}$, LIST $\leftarrow$ LIST $\cup \{j\}$, label$(j) = (i, \min\{L2(i), b_{ij} - x_{ij}\})$

For all j such that $(j,i) \in A$, $x_{ji} > 0$ and $j \notin W$,

$W \leftarrow W \cup \{j\}$, LIST $\leftarrow$ LIST $\cup \{j\}$, label$(j) = (-i, \min\{L2(i), x_{ji}\})$

Check to see if we have an augmenting path: If $t \in W$, go to step 3.

Step 3: Update flow

Augment along updating path:

Use first labels to trace path back from t

Increase flow by $L2(t)$ on forward arcs of path.

Decrease flow by $L2(t)$ on backward arcs of path.

Erase labels.

Return to step 2.

Theorem When the Ford-Fulkerson algorithm terminates, it does so at optimal flow.

Proof Consider the (s,t)-cut $(W, \bar{W})$.

All arcs (i,j) from W to $\bar{W}$ must be saturated, otherwise j would have been added to W when i was scanned.

Similarly all arcs (j,i) from $\bar{W}$ to W must be empty; otherwise j would have been added to W when i was scanned.

Therefore, $C(W, \bar{W}) =$ value of flow.

Therefore, flow must be optimal, and also $(W, \bar{W})$ must be the (s,t)-cut of minimum capacity //

~~This proves the above theorem that~~

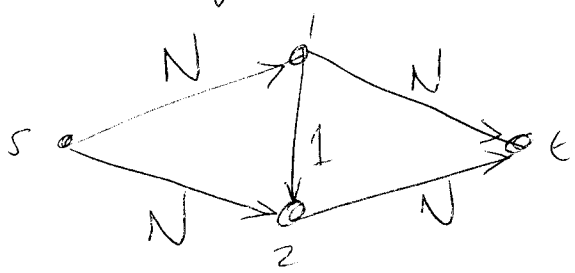
This proves max flow = min cut.

Also proves that a feasible flow x is maximum if there does not exist an augmenting path wrt it.

Theorem If all edge capacities are integer, then there exists an optimal flow where all edge flows are integer.

Proof Always increase flow by integer amount on augmenting path. //

Consider the graph:



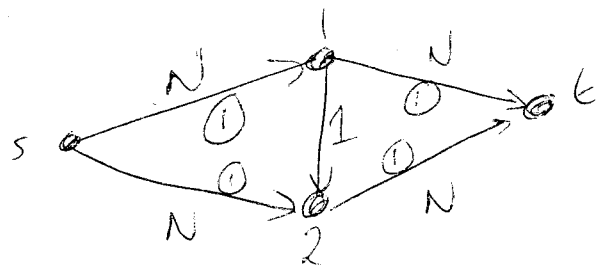
Max flow is $2N$

Initially have flow 0.

~~Could find~~ $s, 1, 2, t$ is an augmenting path, maximum possible flow on this path has value 1.

Then $s, 2, 1, t$

So have network

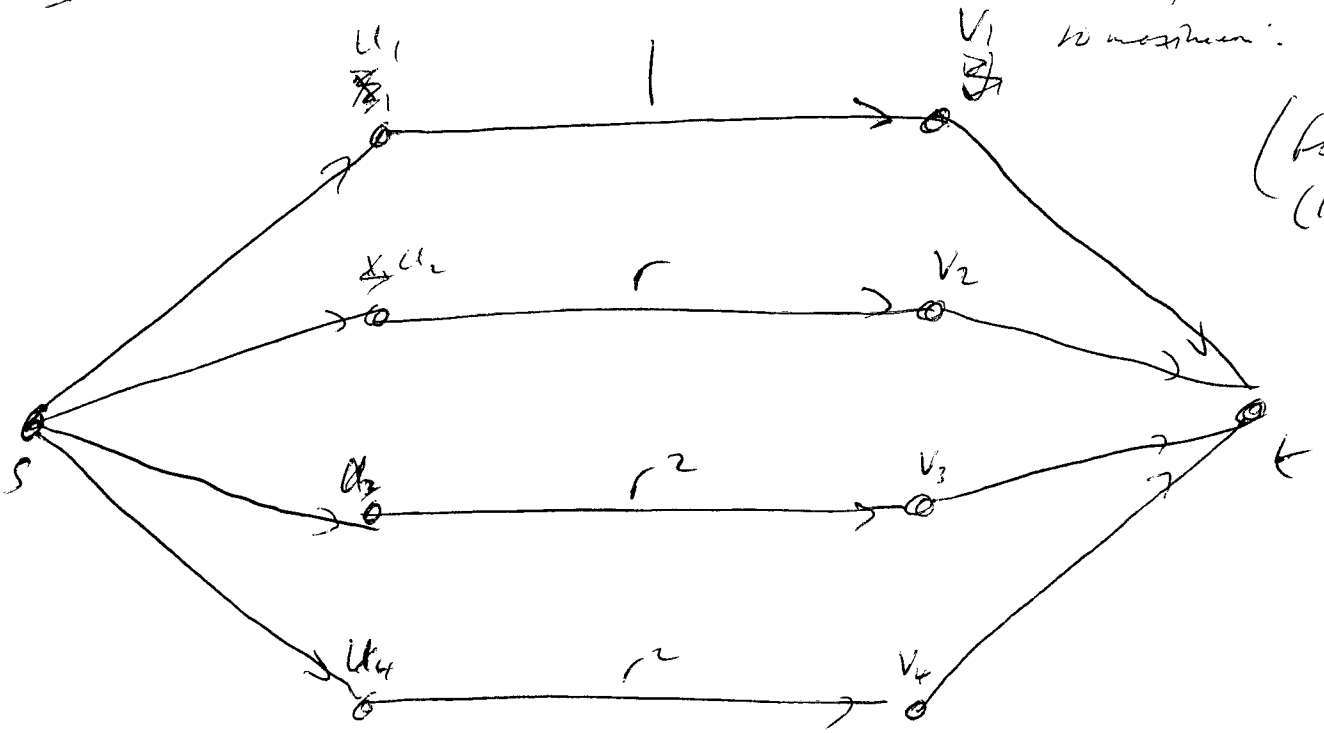


Circles are flows.

Keep proceeding in this way, so ~~need~~ take $2N$ augmenting flows to get optimal flow.

BAD: because function of edge capacities (not even $\log N$, but straight N).

Irrational example flow does not converge



to maximum.

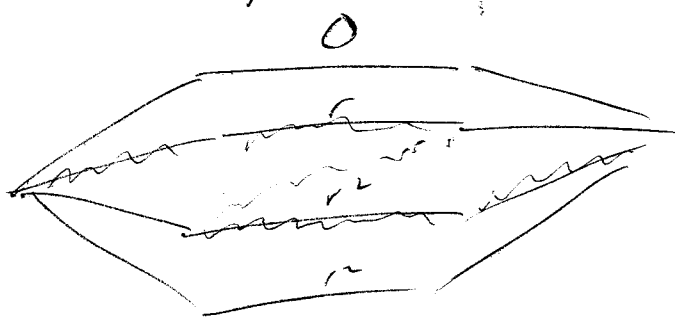
(Ford & Fulkerson (1962), p. 21)

$$r = \frac{-1+\sqrt{5}}{2} < 1. \qquad r^n = r^{n+1} + r^{n+2}.$$

Digraph is complete, apart from s and t, with capacity on every edge (apart from those indicated) being 4. Also missing edge (v_i, u_i) $i = 1, \ldots, 4$

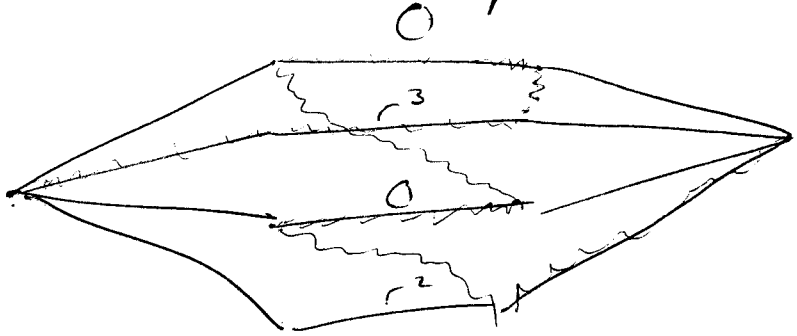
Put flow 1 along path s, u_1, v_1, t.

Residual capacity:



Now put flow r^2 along path s, u_2, v_2, u_3, v_3, t

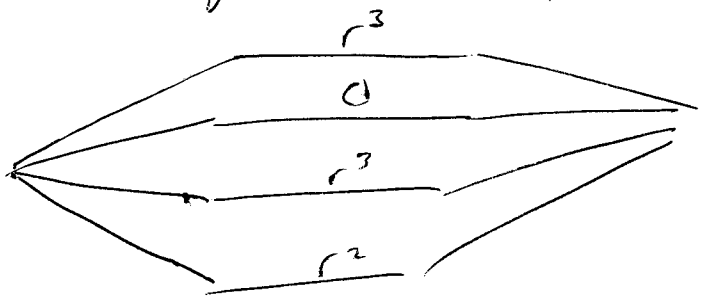
Now residual capacities:



Augmenting path: $s, u_2, v_2, v_1, u_1, v_3, u_3, v_4, t$

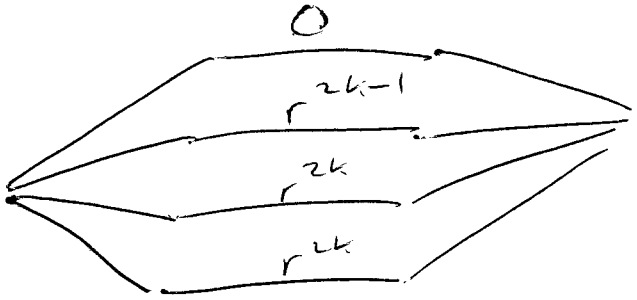
Max flow r^3.

Give new residual capacities:

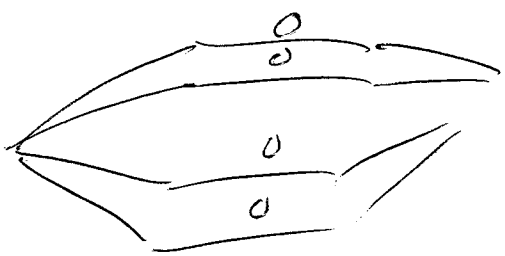


Looks like two steps ago, but with powers of r increased.

Can keep going like this, getting



Tends to



Never gets here.

Flow is only $1 + r + r^2 + r^3 < 4$

However, we have the following theorem:

Thm If at each step of the augmenting path algorithm, a shortest length augmenting path is found, then the number of augmentations is bounded by mn.

Shortest path can be found by breadth-first-search &
Scan all vertices at distance i before scanning any vertex at distance $i+1$.

Work to find an augmenting path is $O(n)$,
since each arc is considered no more than once.

~~Need at most~~

Best max-flow algorithms in terms of worst-case performance are not based on augmenting paths.
Involve looking for a collection of flows, and on each of these flows, ~~the flow~~ it is not possible to increase the total flow without decreasing flow on some arc.
(blocking flow.)

Goldberg / Tarjan / $O(n \log n)$.

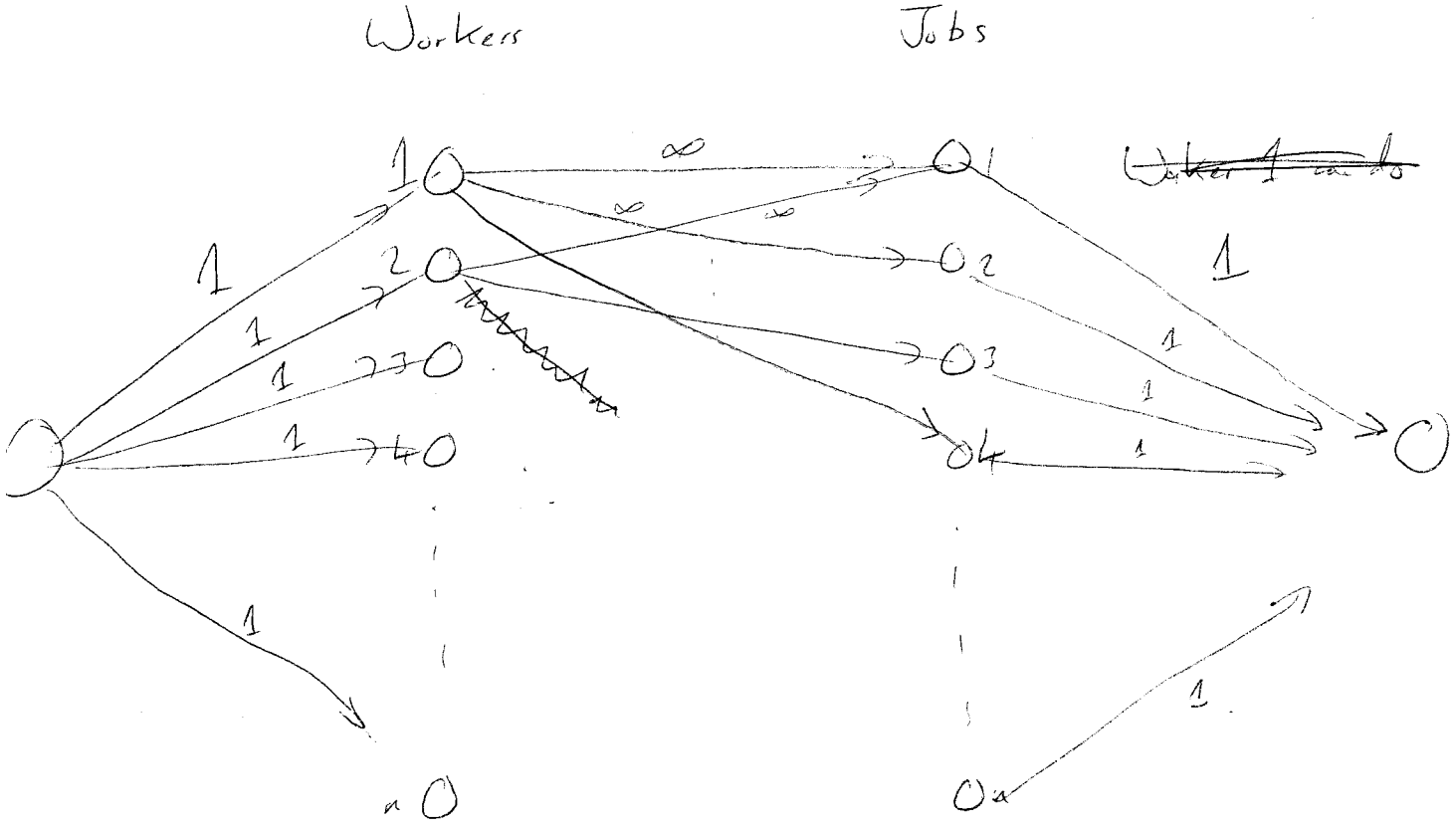
An application of max flow:

Consider an assignment problem:

Have n workers, n jobs, each job requires exactly one worker.

Each worker i is trained to do some subset J_i of the jobs.

Can express this as max flow as follows:



Worker 1 can do jobs 1, 2, 4

2 1, 3

⋮