Notes 9

- The network simplex algorithm is the main algorithm solvers like CPLEX use to solve network problems.
- Network simplex algorithm solves min-cost flow problems.
  - that means it also solves shortest path and max-flow.

- Generally, there are three types of optimization algorithms:

A
- Start with primal feasible, iteratively increase the objective function value while maintaining feasibility, until we can't increase further

B
- Start with an infeasible solution that over-shoots the optimal objective. Iteratively decrease the objective function value, making your solution more and more feasible. Once its feasible, we're done.

C
- Start with a primal solution (feasible) and a possibly infeasible dual solution, but they maintain complimentary slackness. Iteratively increase the feasibility of the dual solution while maintaining complimentary slackness. Once its feasible, we are done

A) Ford-Fulkerson B) Dijkstra C) Network Simplex.

Primal.

$$\min \quad \sum c_{ij} y_{ij}$$

$$\sum y_{ia} - \sum y_{aj} = b(a) \quad : \rho_a$$

$$y_{ij} \le u_{ij} \quad : \pi_{ij}$$

$$0 \le y_{ij}$$

Dual

$$\max \quad \sum_a b(a)\rho_a + \sum_{ij} u_{ij}\pi_{ij}$$

s.t.

$$\rho_j - \rho_i + \pi_{ij} \le c_{ij}$$

$$\rho_i \text{ free}$$

$$\pi_{ij} \le 0$$

- because of complimentary slackness, if
  $y_{ij} \neq u_{ij}$ then $\pi_{ij} = 0$.
- that means for those non-tight arcs,

$$\underbrace{\rho_j - \rho_i - c_{ij}}_{\nearrow} \le 0.$$

can be interpreted in terms of setting prices.

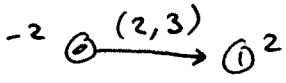
- buy at (i) $(-\rho_i)$
- transport from (i) to (j) $(-c_{ij})$
- sell at (j) (ρ_j)

- must make no money to be feasible.

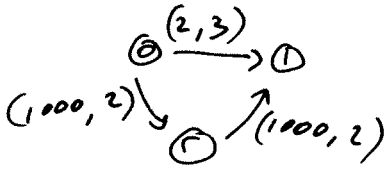
Network Simplex(G, costs, upper bounds etc)

- get rid of all lower bounds and negative costs through transformations to supply/demands
- add a fake "root" node, r.
- add arcs (a, r) for each node a with $b(a) \neq 0$ (supply
- add arcs (r, a) for each node a with $b(a) \geq 0$ (demand
- make those arcs have a huge cost $(n \cdot C_{max})$, capacity $|b(a)|$
- start with feasible solution shipping everything through r.
- maintain a tree of arcs, T ... like a basis for simplex.
- until all edges (i, j) are either 1) tight or 2) dont make money
  - compute dual variables y_a based on the tree T
  - identify the arc that makes the most money, e
  - add the arc to T, to create a cycle C
  - push as much flow on C in the direction of the new arc as possible, maintaining y feasible
  - at some point, we cant push any more because of some arc, e'
  - kick e' out of the tree T, replace it with e

e.g. (cost, capacity)

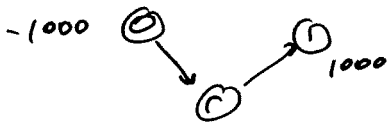


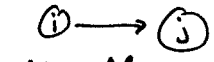
(original)



(with r)

T

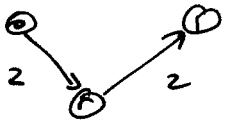




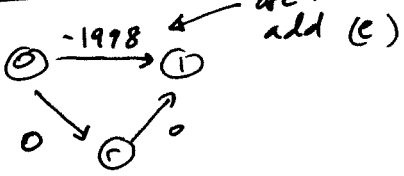
add add subtract = 0.

$s_r = 0$

y

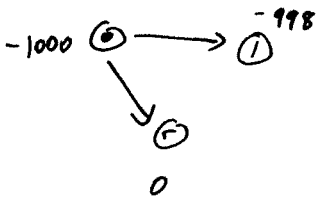


"profit"

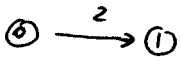


- can put 2 units in the direction of the arc.
- kick out (r, 1)

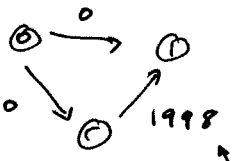
T



y



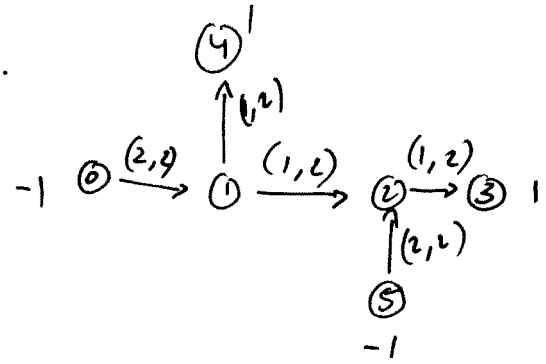
"profit"



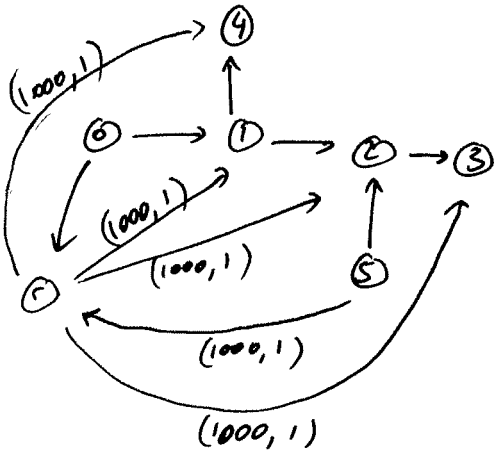
lose lots of $.

so, now we are at optimal because there is no arc that makes money.

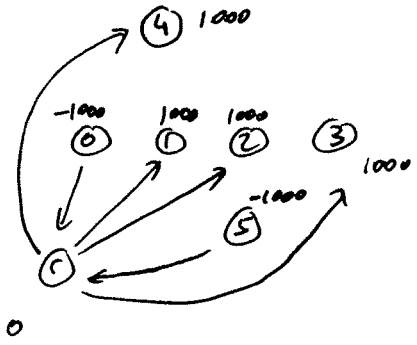
e.g.



(original)

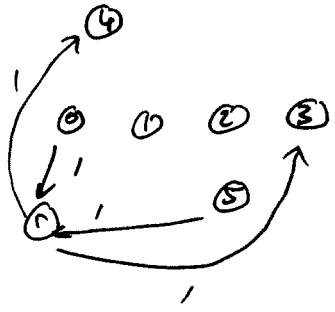


T

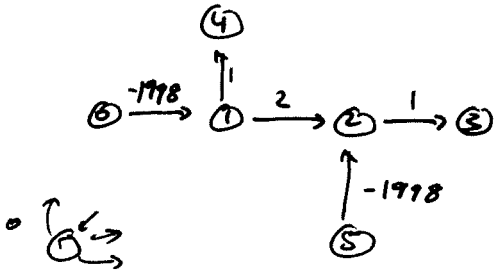


0

Y

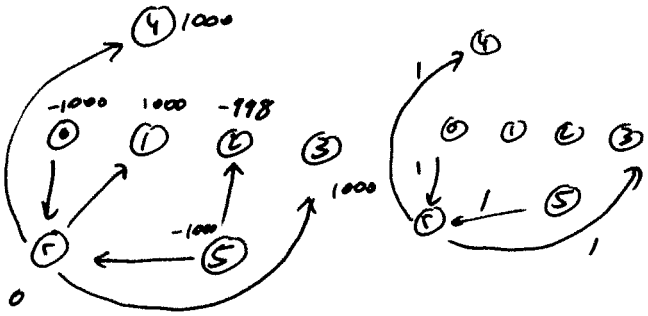


profit



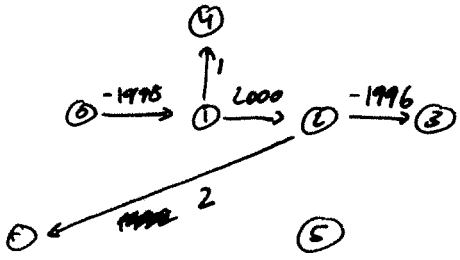
- add (5,2) to the tree.
- can add 0 flow, kick out (r,2).

T



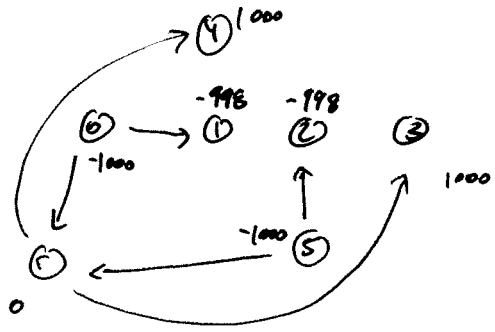
Y

profit

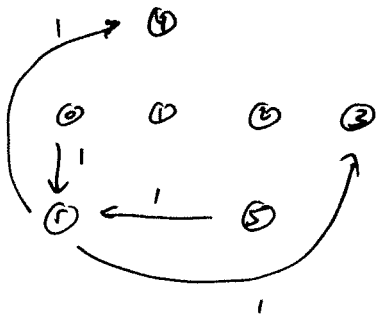


- add (0,1)
- can add 0 flow, kick out (r,1)

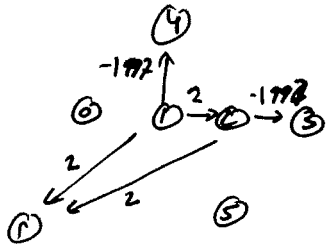
T



Y

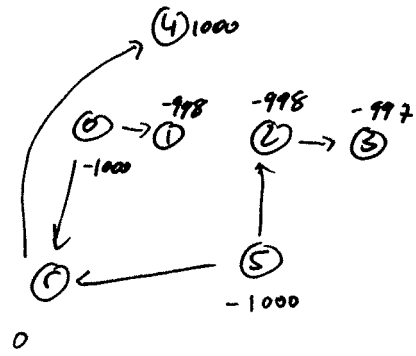


profit

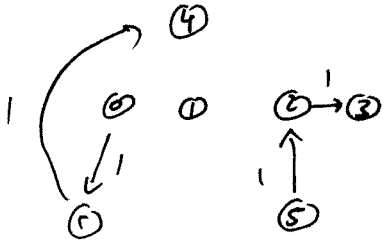


- add (2, 3)
- can put 1 flow.
- kick out arc (r, 3)

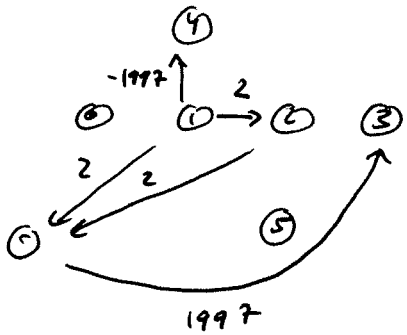
T



Y

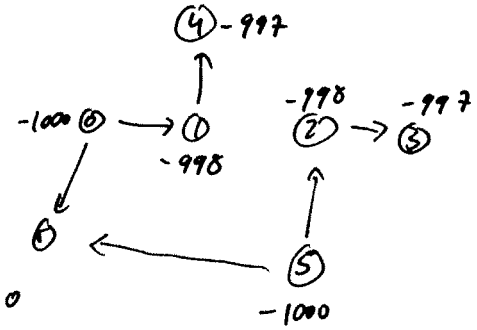


profit

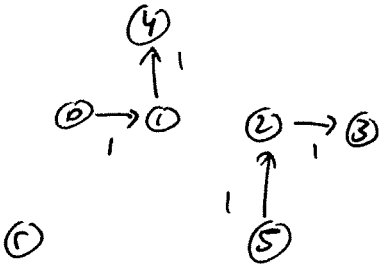


- add (1, 4)
- can put 1 flow on the cycle.
- kick out arc (r, 4)

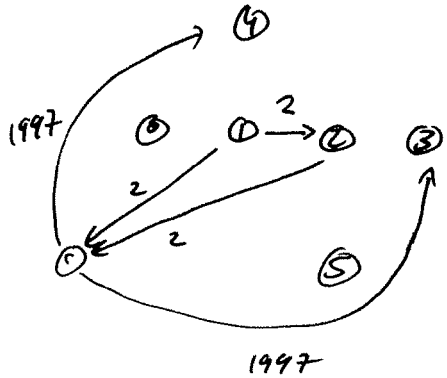
T



Y



profit



done.