Outline:

- Finish uncapacitated simplex method
- Negative cost cycle algorithm
- The max-flow problem
- Max-flow min-cut theorem

Uncapacitated Networks: Basic primal and dual solutions

- Flow conservation constraints $\mathbf{A}\mathbf{f} = \mathbf{b}$
  (rows $\leftrightarrow$ nodes; columns $\leftrightarrow$ arcs)
- delete last row: $\tilde{\mathbf{A}}\mathbf{f} = \tilde{\mathbf{b}}$
- basic (feasible) solution $\leftrightarrow$ (feasible) tree solution
  $n-1$ basic variables: flows that lie on the tree
  (easy to calculate given the tree)
- Calculation of dual basic solution $\mathbf{p}$ (one variable per node)

$$[p_1 \cdots p_{n-1}] \begin{bmatrix} | & & | \\ \tilde{\mathbf{A}}_{B(1)} & \cdots & \tilde{\mathbf{A}}_{B(n-1)} \\ | & & | \end{bmatrix} = [c_{B(1)} \cdots c_{B(n-1)}]$$

$$[p_1 \cdots p_{n-1}\ 0] \begin{bmatrix} | & & | \\ \mathbf{A}_{B(1)} & \cdots & \mathbf{A}_{B(n-1)} \\ | & & | \end{bmatrix} = [c_{B(1)} \cdots c_{B(n-1)}]$$

  i.e., use original columns (dimension n), but set $p_n = 0$

- if $(i,j) \in$tree, i.e., f_{ij} is basic, $p_i - p_j = c_{ij}$
  solve by starting at "root" node n, move down the tree
- $p_i - p_j =$ cost of path from i to j along the tree
- for (i,j) outside the tree:
  $\overline{c}_{ij} = c_{ij} - (p_i - p_j)$= cost of cycle created by arc (i,j).

Uncapacitated Network Simplex Algorithm

- **Algorithm:**
  Start with a tree T, and flows f_{ij}, $(i,j) \in T$
  - $p_n = 0$; solve $p_i - p_j = c_{ij}$, $(i,j) \in T$
  - For $(i,j) \notin T$, let $\overline{c}_{ij} = c_{ij} - (p_i - p_j)$
  - If all $\overline{c}_{ij} \geq 0$, then optimal,
    and the p_i are a dual optimal solution
  - Else pick (i,j) with $\overline{c}_{ij} < 0$
  - Consider cycle created by arc (i,j)
  - "Push" flow around that cycle, until some arc flow is zeroed
  - Zeroed arc exits the tree/basis
- If all b_i are integer, basic (or optimal) $\mathbf{f}$ is integer
- If all c_{ij} are integer, basic (or optimal) $\mathbf{p}$ is integer
- How to start the algorithm?
  - Assume single source, single sink
  - Auxiliary arc from source to sink, with high cost.
  - Let that arc be in the tree, all flow goes through it.

The capacitated case

- Tree solution:
  Pick a tree. For $(i, j) \in T$, set f_{ij} either to 0 or to u_{ij}
- Calculate p_i and $\overline{c}_{ij}$ as before.
- If $\overline{c}_{ij} < 0$ and $f_{ij} = 0$, push flow around the cycle, in the direction of (i, j).
- If $\overline{c}_{ij} > 0$ and $f_{ij} = u_{ij}$, push flow in the opposite direction.

Optimality conditions

- **Def: Pushing** flow around a cycle:
  $f_{ij} \to f_{ij} + \delta$ for forward arcs
  $f_{ij} \to f_{ij} - \delta$ for backward arcs
  (flow conservation equation is respected)
- **Def:** A cycle is **unsaturated** if we can push some flow around it.
  $f_{ij} < u_{ij}$ for forward arcs
  $f_{ij} > 0$ for backward arcs
- **Def: Cost** of a cycle:
  Sum of the c_{ij}, with minus sign for backward arcs.
- **Theorem:** Optimal flow iff there is no unsaturated cycle with negative cost.
- Easy direction:
  If $\exists$ negative cost unsaturated cycle,
  can push some flow along that cycle
  cost reduction
  flow is not optimal
- Converse direction: proof is more involved

Negative Cost Cycle Algorithm

- **Algorithm:**
  1. Start with a feasible flow $\mathbf{f}$.
  2. Search for an unsaturated cycle C with negative cost.
  3. If none, stop (optimal)
  4. Else, push as much flow as possible along C
  (if can push an infinite amount, optimal cost is $-\infty$)

- Assume b_i integer, and u_{ij} integer or infinite
  Assume integer initial flow

- Integrality maintained throughout

- If the optimal cost is finite,
  terminates with integer optimal solution

- In noninteger case, not guaranteed to terminate!
  - Number of iterations can be large

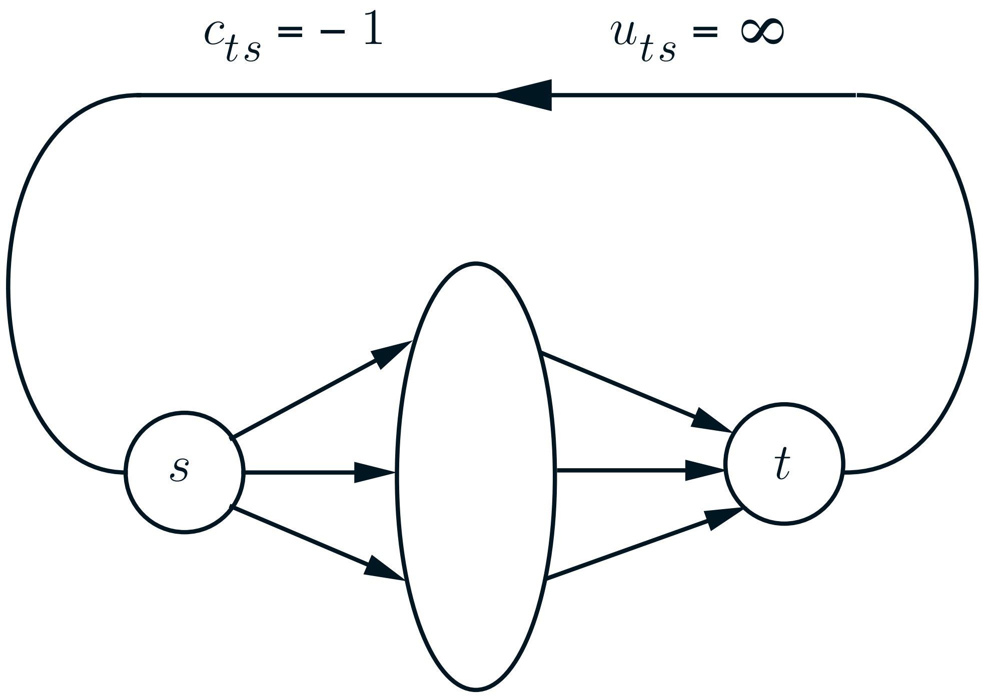
- Algorithm can be made efficient under special rules
  for choosing among negative cost cycles

- Searching for negative cost cycles can be done in $O(n^3)$ time

The Maximum Flow problem

- Given capacities u_{ij}; no costs
  maximize flow from source s to destination t

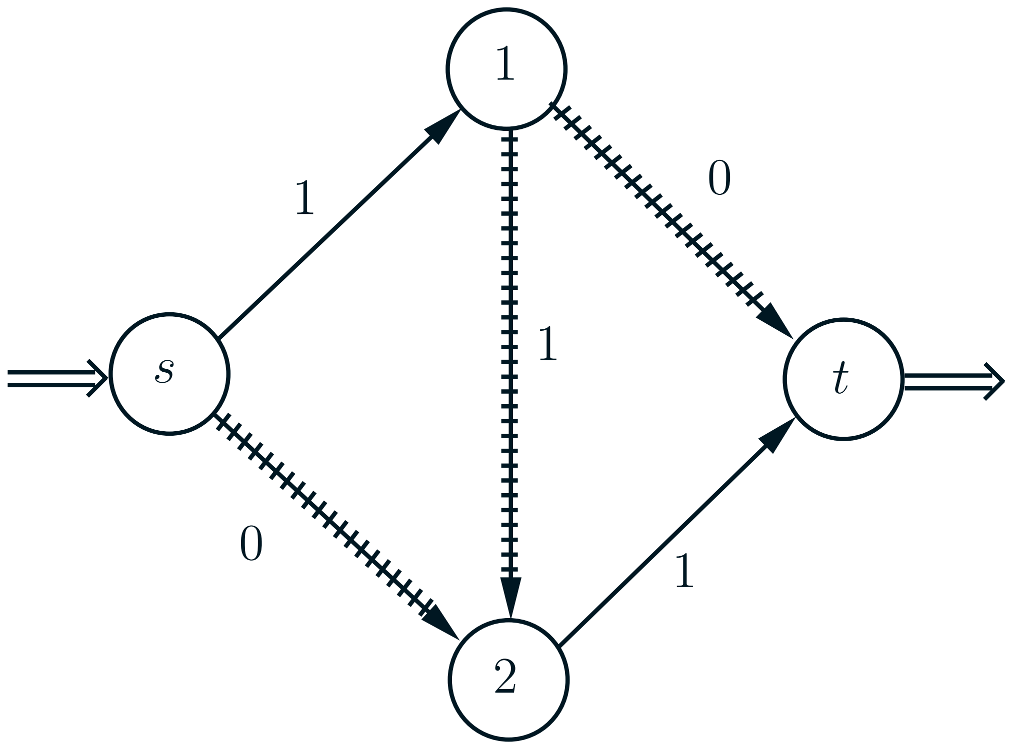
- Equivalent min-cost flow problem:



- Negative cost cycle:
  artificial arc and "unsaturated" path from s to t
  ("**augmenting path**")
  along which flow can be pushed

Augmenting paths

- Arcs that can be used:
  - can use arc (i, j) in forward direction if $f_{ij} < u_{ij}$
  - can use arc (i, j) in backward direction if $f_{ij} > 0$



(all capacities are 1)

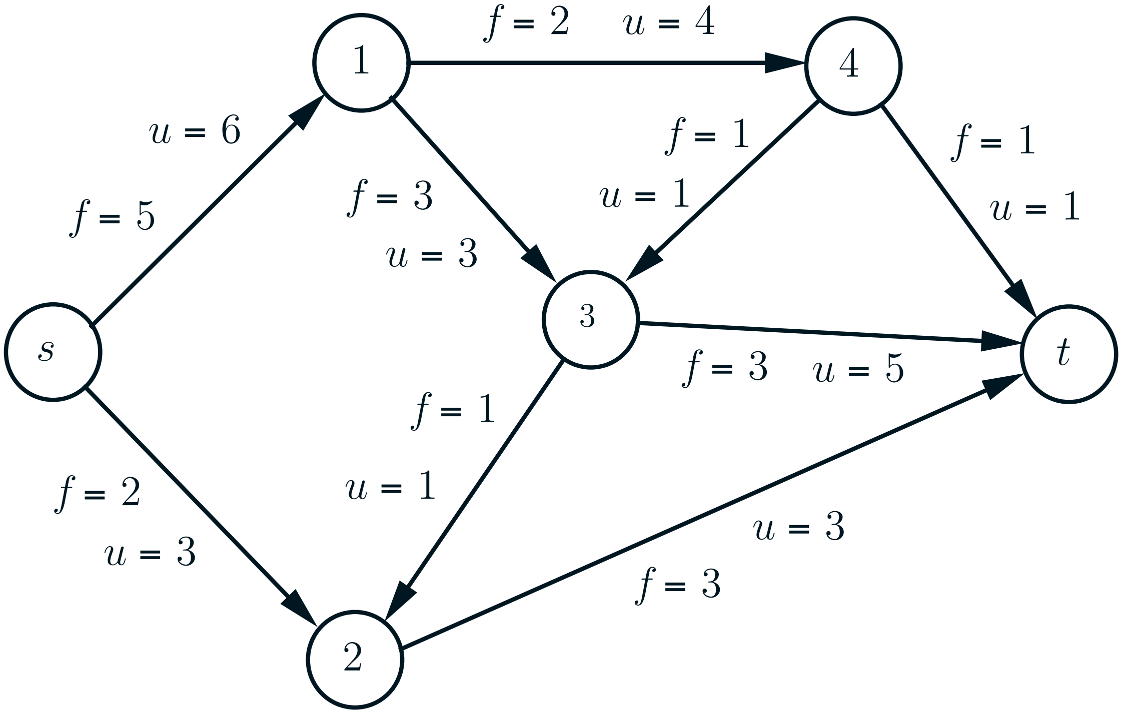
$$\text{flow pushed :} \quad \min\left\{\min_{(i,j)\in F}(u_{ij} - f_{ij}),\ \min_{(i,j)\in B} f_{ij}\right\}$$

- Ford-Fulkerson algorithm:
  search for augmenting path and push flow

Searching for an augmenting path

- **Labeled** node i: have determined that $\exists$path from s to i, with
  $f_{ij} < u_{ij}$ on forward arcs
  $f_{ij} > 0$ on backard arcs
- **Scanned** node i: have looked at all neighbors of i and attempted to label them
- Labeling algorithm:
  - Initialize: label s
  - select labeled but unscanned node
  - scan it, and label its neighbors, if possible
  - repeat
- If t labeled, have found augmenting path
- If stuck, with t unlabeled, no augmenting path exists.
- Work: $O(m)$

Labeling algorithm example



Comments on overall algorithm

- Not guaranteed to terminate!
- Works with primal feasible solutions
- Max-flow is infinite iff $\exists$ path from s to t with infinite capacities
  (check ahead of time)
- Guaranteed to terminate if:
  max-flow is finite and u_{ij} are all integer (or rational)
- Complexity (in integer case): [let $U = \max u_{ij}$]

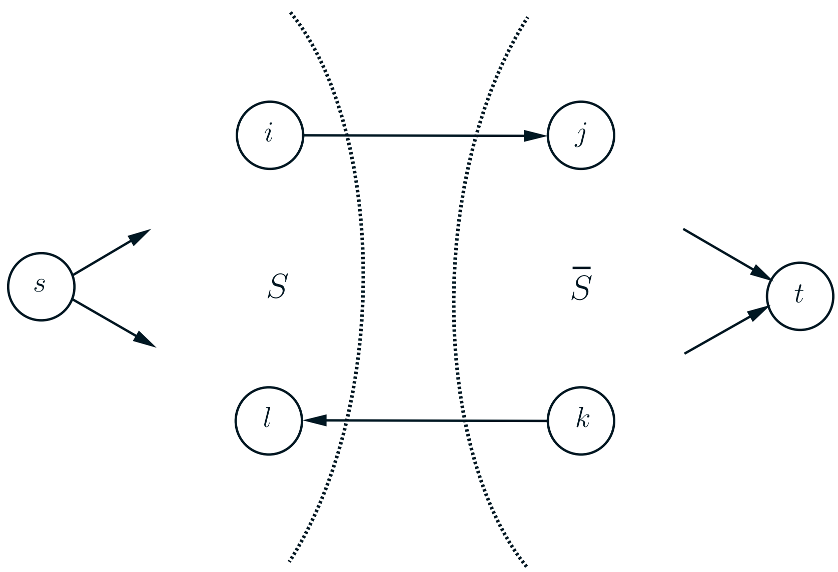
$$nU \cdot O(m)$$

Max-flow min-cut theorem

- Cut S: $s \in S$, $t \notin S$.

$$\text{cut capacity} = C(S) = \sum_{\{(i,j)\in\mathcal{A} \mid i\in S,\ j\notin S\}} u_{ij}$$

- max-flow $\leq \min_S C(S)$
- Start algorithm with optimal flow.
- Fails to find augmenting path, algorithm terminates
- Consider set S of labeled nodes



$$f_{ij} = u_{ij}, \qquad f_{kl} = 0$$

current flow= capacity $C(S)$ of this cut

- Therefore:
  current flow is optimal
  this cut is minimal
  max-flow value = min-cut capacity
- Smacks of duality

Comments

- Size of problem: $O(m \log U)$
- Ford-Fulkerson algorithm: $O(mnU)$: "exponential"
- Can be modified to polunomial($m, n, \log U$) (Exercise 7.25)
- Better algorithms:
  look for "shortest" augmenting path
  augment flow on many paths simultaneously
  etc. etc.
  can get complexity $O(mn \log n)$