5. Network flow

- Network flow
- Maximum flow problem
- Ford-Fulkerson algorithm
- Min-cost flow

Network flow

- Network N is a set of
  - a directed graph $G = (V, E)$
  - a source $s \in V$ which has only outgoing edges
  - a sink (or a destination) $t \in V$ which has only incoming edges
  - a positive *integral* capacity c_{ij} on each directed edge (i, j)

  - we can define network on an undirected graph by making every edge bi-directional
  - the integral capacity assumption can be relaxed some times, but it is necessary other times

- a flow $f : E \mapsto \mathbb{R}^+$ is **feasible** if
  - edge capacity limit:

    $$0 \;\le\; f_{ij} \;\le\; c_{ij}$$

    for all $(i, j) \in E$
  - conservation of flow:

    $$\sum_j f_{ij} \;=\; \sum_k f_{ki}$$

    for all $i \in V \setminus \{s, t\}$

- **define.** the **value** of the flow is the amount of flow leaving the source
  - ★ **claim.** equivalently, the value of a feasible flow is the amount of flow entering the sink

$$v(f) = \sum_j f_{sj} = \sum_j f_{jt}$$

- **proof.**

$$\begin{aligned} v(f) &= \sum_j f_{sj} \\ &= \sum_j f_{sj} + \sum_{i\in V\setminus\{s,t\}} \left[\sum_a f_{ia} - \sum_b f_{bi}\right] \\ &= \sum_{i\in V}\sum_{j\in V} f_{ij} - \sum_{i\in V\setminus\{t\}}\sum_{j\in V} f_{ji} \\ &= \sum_{i\in V}\sum_{j\in V} f_{ij} - \sum_{i\in V}\sum_{j\in V} f_{ji} + \sum_j f_{jt} = \sum_j f_{jt} \end{aligned}$$

the last line follows from the fact that, in the summation, the flows coming from the source are counted only once with negative sign and the flows going to the sink are counted once with a positive sign.

- **max-flow problem** is to find a feasible flow with maximum value

- we denote a partition of nodes into two sets S and S^c, a s-t cut, by the notation (S, S^c), where $s \in S$ and $t \in S^c$
- the **value of the cut** $c(S, S^c)$ is the sum of all capacities leaving S and entering S^c

$$c(S, S^c) \;=\; \sum_{a \in S, b \in S^c} c_{ab}$$

- **claim.** for any cut (S, S^c) it can be shown that

$$v(f) \;=\; \sum_{a \in S, b \in S^c} f_{ab} - \sum_{a \in S, b \in S^c} f_{ba}$$

proof.

$$\begin{aligned} v(f) &= \sum_j f_{sj} = \sum_j f_{sj} + \sum_{i \in S \setminus \{s\}} \Big[\sum_a f_{ia} - \sum_b f_{bi} \Big] \\ &= \sum_{i \in S} \sum_{a \in S} f_{ia} + \sum_{i \in S} \sum_{a \in S^c} f_{ia} - \sum_{i \in S} \sum_{a \in S} f_{ai} - \sum_{i \in S} \sum_{a \in S^c} f_{ai} \\ &= \sum_{a \in S, b \in S^c} f_{ab} - \sum_{a \in S, b \in S^c} f_{ba} \end{aligned}$$

since all terms $f_{a,b}$ with both a and b in S appear twice, once with positive sign and once with the negative sign, canceling each other, except for those which cross over S and S^c

- **claim.** for any flow f and any cut (S, S^c), $v(f) \leq c(S, S^c)$
  proof.

$$v(f) = \sum_{a \in S, b \in S^c} f_{ab} - \sum_{a \in S, b \in S^c} f_{ba} \leq \sum_{a \in S, b \in S^c} f_{ab} \leq \sum_{a \in S, b \in S^c} c_{ab} = c(S, S^c)$$

- **claim.** max-flow $\leq$ min-cut

$$\max_{f \text{ feasible}} v(f) \quad \leq \quad \min_{S} c(S, S^c)$$

max-flow problem

- for general networks, there is a polynomial time algorithm that finds the max-flow
- the max-flow is always equal to the min-cut value

$$\max_{f \text{ feasible}} v(f) \quad = \quad \min_{(A,B)} c(A,B)$$

- Greedy max-flow algorithm (sub-optimal)

```
★ Initialize $f_{ij} = 0$ for all $i, j$
★ **repeat**
      - find a path $P$ between $s$ and $t$ such that
            $\min_{(i,j) \in P} (c_{ij} - f_{ij}) > 0$
      - call such a path unsaturated
      - let $df = \min_{(i,j) \in P} (c_{ij} - f_{ij})$
      - set $f_{ij} = f_{ij} + df$ for all $(i, j) \in P$
★ **until** no more unsaturated $s - t$ paths
```

- this algorithm is inefficient (number of paths can be exponential in the size of the graph)
- this algorithm does not find the max-flow (sub-optimal)

a **residual network** $R(N, f) = (V, E_r)$ is a vertex set V with weighted and directed edge set E_r constructed as follows:

  - ⋆ if $f_{ij} < c_{ij}$ place an edge (i, j) with capacity $c'_{ij} = c_{ij} - f_{ij}$
  - ⋆ if $f_{ij} > 0$ place an edge (j, i) with capacity $c'_{ji} = f_{ij}$ (note that this is in the opposite direction)

- idea: any path P in $R(N, f)$ gives a path which we can increase the flow, including the ones that reverse previously assigned ones

- Ford-Fulkerson algorithm, 1956

```
⋆ Initialize $f_{ij} = 0$ for all $i, j$
⋆ while there is a path $P$ from $s$ to $t$ in $R(N, f)$ **do**
      - send a flow of value $df = \min_{(i,j)\in P} c'_{ij}$ in $R(N, f)$ along $P$
      - update $f$ in $(N, f)$: set $f_{ij} = f_{ij} + df$ for all $(i, j) \in P$
      - rebuild the residual network $R(N, f)$
⋆ **end while**
```

- a **path** in a directed graph is a sequence of nodes $v_1 v_2 \cdots v_k$ such that there is a directed edge from v_i to v_{i+1}

- **theorem.** If Ford-Fulkerson algorithm terminates, it outputs a maximum flow

- **proof.**
  - ★ suppose the algorithm terminates at time T with flow f^*, then there is no path from s to t in $R(N, f)$
  - ★ let S be the set of nodes reachable from s
  - ★ **claim 1.** $v(f^*) = c(S, S^c)$
  - ★ this proves that f^* is the maximum flow, since we already know that the maximum flow is at most the minimum cut and we found a flow whose value is the same as a cut (which is also a minimum cut)

- **proof of claim 1.**
  - ★ if exists an edge (i, j) such that $i \in S$, $j \in S^c$, and $f^*_{ij} < c_{ij}$, then there must be an edge in $R(N, f^*)$ from S to T
  - ★ if exists an edge (i, j) such that $i \in S^c$, $j \in S$, and $f^*_{ij} > 0$, then there must be an edge in $R(N, f^*)$ from S to T
  - ★ therefore,

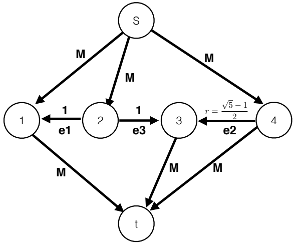
$$\begin{aligned} v(f) &= \sum_{i \in S, j \in S^c} \underbrace{f_{ij}}_{=c_{ij}} - \sum_{i \in S, j \in S^c} \underbrace{f_{ji}}_{=0} \\ &= c(S, S^c) \end{aligned}$$

corollary. the minimum cut value in a network is the same as the maximum flow value

corollary. if all edge capacities in a network are non-negative integers, then there exists an integral maximum flow

we are left to show that the algorithm terminates under mild assumptions:
claim. for a network where all weights are rational numbers, Ford-Fulkerson algorithm terminates in finite time

it should also be noted that there are cases where the algorithm does not terminate:
claim. there exists a network with irrational weights such that Ford-Fulkerson algorithm never terminates



step	augmenting path	added flow	residual flow e_1	e_2	e_3
0			1	r	1
1	$(s, 2, 3, t)$	1	1	r	0
2	$(s, 4, 3, 2, 1, t)$	r	r^2	0	r
3	$(s, 2, 3, 4, t)$	r	r^2	r	0
4	$(s, 4, 3, 2, 1, t)$	r^2	0	r^3	r^2
5	$(s, 1, 2, 3, t)$	r^2	r^2	r^3	0

- r is chosen such that $r^2 = 1 - r$
- since after step 1 we have the same residual capacity as after step 5 up to a scaling by r^2, the process never stops
- the value of flow converges to $v(f^\infty) = 1 + 2\sum_{i=1}^{\infty} r^i = 3 + 2r$, while the maximum flow is $v(f^*) = 2M + 1$
- the algorithm never terminates, and further the flow value does not converge to the maximum

Run-time

- **lemma.** if all edge capacities of N are integral, then Ford-Fulkerson algorithm terminates in $O(m\, v(f^*))$ time

- **proof.** the value of the flow increases at least by one at each iteration, and a directed path can be found in $O(m)$ time

- how many operations do we need to search for a path from s to t in a directed graph with number of edges at most m?
- how many iterations are required in the worst-case?

- Shortest Augmenting Path (SAP)
  - Choose the augmenting path in the residual network that has the fewest number of edges
  - **lemma.** If, at a certain iteration, the length of a shortest path from s to t in the residual network is ℓ, then at every subsequent iterations it is $\geq \ell$. Furthermore, after at most m iterations, the distance from s to t must become $\geq \ell + 1$.
  - **lemma.** the shortest augmenting path can be found in $O(m)$ time using breadth-first search.
  - **theorem.** SAP implementation of Ford-Fulkerson algorithm runs in time $O(m^2 n)$, where $m = |E|$ and $n = |V|$.

Application

- **Project selection** problem is defined by
  - a set of projects $\{T_1, \ldots, T_n\}$, each with a positive profit P_i
  - a set of equipments $\{E_1, \ldots, E_m\}$, each with a positive cost C_j
  - a project has a subset of equipments which are prerequisite
- find the optimal subset of projects that satisfy the prerequisite requirements and has maximum total profit.
  - construct a graph G where all the projects are connected to the source, and the weight of that edge is P_i
  - connect all the equipments to the sink, and the weight of that edge is C_j
  - if E_j is a prerequisite for T_i then add an edge (T_i, E_j) with weight ∞
  - **claim.** a cut (A, B) with a finite value $c(A, B) < \infty$ defines a feasible set of equipments and projects
  - **claim.** find the minimum cut (A^*, B^*). $B^* \setminus t$ is the set of projects and equipments that give maximum profit.

$$c(A,B) = P^+ - \underbrace{\Big(\sum_{T_i \in B} P_i - \sum_{E_j \in B} C_j\Big)}_{\text{net profit}} ,$$

where $P^+ = \sum_i P_i$ is the total profit achievable if one does all the projects.

- **Team elimination** problem
  - wins: (NYY,93), (BOS,89), (BAL,86), (TOR,88)
  - games to play: (NYY-BOS,1), (NYY-TOR,6), (NYY-BAL,1), (BOS-BAL,3), (TOR-BAL,1)
  - Question: Can Boston end up winning the division?
- Max-flow formulation
  - suppose BOS has won all the remaining games
  - consider a network with a source s and a sink t
  - each pair of teams NYY-TOR is a node, and connect it to the source with weight equal to games to play
  - each team BAL is a node, and connect matches to teams with infinite capacity
  - add edge between each team and the sink with weight $W - w_i$, where w_i is how many times the team has won, and W is the most wins BOS can have if it wins all the rest of the game. This ensures that each team does not accumulate more than W wins.
- if max-flow is less than or equal to the total number of remaining games, then BOS still has a chance (since there is a way to distribute the wins of the remaining games such that no team exceeds BOS wins)
- if max-flow is strictly less than the total number of remaining games, then BOS should be eliminated (since no matter how we distribute the wins of the remaining games, there is no way all teams can have less wins than BOS)

Minimum cost flow

motivating example:

- ⋆ There are n workers and n jobs. Assigning worker i to job j incurs a cost a_{ij}. Match workers to jobs such that the total cost is minimized (while maximizing the cardinality of the matching).

this weighted matching problem is a special case of **minimum cost flow problem**:

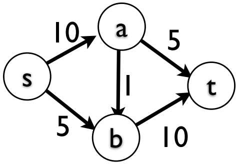
minimum cost flow problem:

- ⋆ given a network N, whose edges each have a cost per unit flow a_{ij} and an integer capacity c_{ij}, find a minimum-cost flow of value v

if we change this problem such that the cost is for using the edge independent of the flow, then the problem becomes NP-hard (i.e. no known algorithm can solve it in polynomial time)

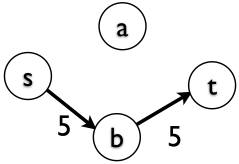
given N, we can always find the maximum flow value v^*, and we also know that for the value of $v > v^*$ it is impossible to find a flow of value v

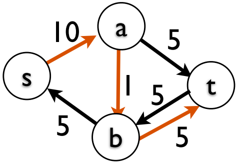
Review of Ford-Fulkerson algorithm for max flow problem

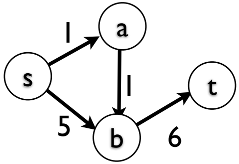


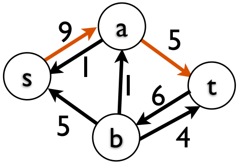
FLOW

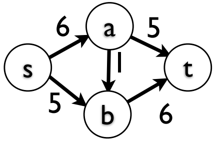
R(n,f)

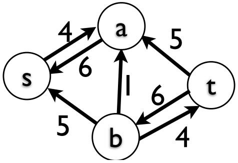






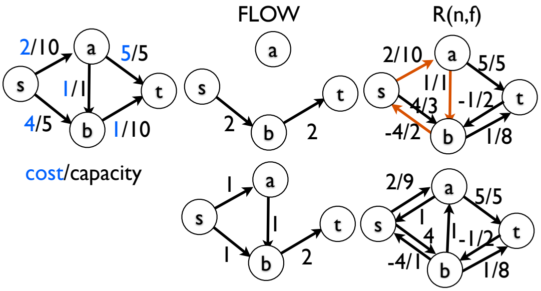






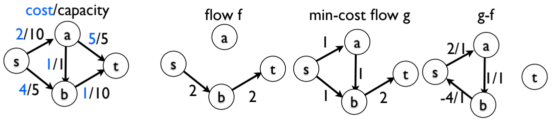
Minimum cost flow

- Idea: find the minimum cost augmenting path
- **Definition.** The *cost* of an augmenting path is the sum of the costs of its edges.
- When constructing a residual graph, put $-c_{ij}$ on the reverse edges.
- **Definition.** An *augmenting cycle* c is a directed cycle, whose edges all have positive capacity. The cost of c is the sum of the cost of directed edges in c.
- If we have a negative cost cycle, then we can augment the cycle to our flow to get another flow of same value but smaller cost.



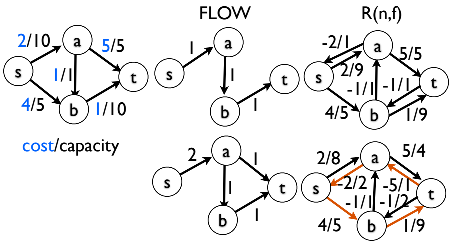
Minimum cost flow

- **Theorem.** A flow f of value v has a minimum cost among all flows that have value v if and only if there is no negative cost cycle in the residual network $R(N, f)$.
- **Proof.**
  - ($\Rightarrow$) Proof by transposition. Suppose there is a negative cost cycle, then we can add that cycle to our flow to get smaller cost flow of the same value. Then the original flow is not minimum cost flow.
  - ($\Leftarrow$) Proof by transposition. Suppose f is not minimum-cost flow. Then, there is a cheaper flow g of value v. The flow $g - f$ can be decomposed into a union of augmenting cycles. One of the cycles has a negative cost, since $g - f$ has negative cost.
- This gives a way for checking whether a flow has minimum cost or not. How do we find find the minimum cost flow?



Minimum cost flow

- **Theorem.** Let f be a minimum cost flow of value v. And let P be a minimum-cost augmenting path in the residual graph $R(N, f)$. Then, augmenting f by adding P gives a minimum-cost flow of value $v + 1$ (let's call it g).
- **Proof.** Proof by transposition.
  - Suppose g was not a minimum cost flow. Then, from the previous proof, we know that g admits a negative cost cycle C in the residual graph (which was not present in the residual graph $R(N, f)$). Since f admits no negative-cost cycles, there must be some edge $(i, j) \in P$ with $(j, i) \in C$. Then, P is not the minimum cost augmenting path in $R(N, f)$, since $P + C$ is an augmenting path with smaller cost.



Minimum cost flow

- Successive shortest paths algorithm.
  - Start with $f = 0$.
  - Repeat finding minimum cost augmenting paths in the residual graph.
  - Add that flow to current flow until the value is v.

Application: Community detection in social network

- Social network is a network of people connected to their ‘friends’
- Recommending friends is an important practical problem
- solution 1: recommend friends of friends
- solution 2: detect communities
  - idea1: use max-flow min-cut algorithms to find a minimum cut
  - it fails when there are outliers with small degree
  - idea2: find partition A and B that minimize **conductance**:

$$\min_{A,B} \frac{c(A,B)}{|A||B|}$$

Homework 5

Problem 1.

A local charity wishes to organize a series of blind dates. There are n male and n female students, and each male student is to be paired with a female student each evening. Based on forms filled in by the participants, the we know which pairs of students are compatible. Given this information, describe an algorithm to find the maximum number of rounds of blind dates that can be organized, subject to the following conditions: each round consists of exactly n dates, no student can participate in more than one date in the same round, no student encounters another student more than once, and no two incompatible people are ever matched.

Homework 5

Problem 2.

There are n students in a class. We want to choose a subset of k students as a committee. There has to be m_1 number of freshmen, m_2 number of sophomores, m_3 number of juniors, and m_4 number of seniors in the committee. Each student is from one of $k = m_1 + m_2 + m_3 + m_4$ departments. Exactly one student from each department has to be chosen for the committee. We are given a list of students, their home departments, and their class (freshman, sophomore, junior, senior).

Describe an efficient algorithm to select who should be on the committee such that the constraints are satisfied.

Homework 5

Problem 3. We have an $n \times n$ grid like below. A subset of the cells are 'point cells' (marked as black). A monotone path in the grid starts at the top-left cell and at each cell can only move to either right or down by one step (if there exists a cell to the right or below). Eventually, the path ends at the bottom-right cell. The goal is to cover as many 'point cells' as possible, with a single monotone path.

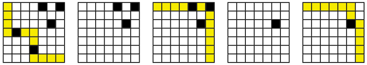


Figure : Greedily covering the point cells with three monotone paths.

(*a*) Describe an efficient algorithm for finding a monotone path that covers the maximum number of 'point cells'.

(*b*) Now consider the problem of covering all point cells using multiple monotone paths. The goal is to cover all point cells using as small number of paths as possible. A greedy heuristic is to iteratively apply the above algorithm. At each step, find the monotone path that covers maximum number of point cells that are not already covered, and repeat until all point cells are covered. Prove by counter example that this algorithm does not always give the optimal solution.

(*c*) Describe an efficient algorithm to compute the smallest set of monotone paths that covers every point cells.

Homework 5

Problem 4.

(a) Given a network $G = (V, E, s, t)$, give a polynomial time algorithm to determine whether G has a unique minimum $s - t$ cut.