Max-Flow Applications

1 Edge-Disjoint Paths

Last time, we proved that we can decompose a flow f into edge disjoint paths if f is acyclic; now, we will prove that we can always find an acyclic f.

Lemma 1. *If f is any (s,t) flow, then in polytime, we can find an acyclic flow f' with $v(f') = v(f)$ and $f'_e \leq f_e$ $\forall e$. If f is integral, we can make f' integral.*

Proof. While $\{e | f_e > 0\}$ has a cycle, we use BFS, DFS, or topological sort to find a cycle C. Then, let $\mathcal{E} := \min_{e \in C} f_e$. Set $f'_e := f_e - \mathcal{E}$ $\forall e \in C$. Then, f' is still a flow because it satisfies conservation and capacity. Furthermore, $v(f') = v(f)$ because s is never in a cycle by definition, so we never decrease $f^{out}(s)$, which is the definition of $v(f)$. This algorithm finishes in $\leq m$ iterations.

Algorithm 1 Edge-Disjoint Paths Algorithm

1: Give each edge $e \in E$ capacity $c_e = 1$.
2: Find an integral max (s,t) flow f.
3: Find a path decomposition of f. Each path carries one unit of flow, so the edges must be disjoint because no edge has $c_e > 1$.
4: Return the paths.

□

Corollary 2. *Manger's Theorem (Edges): The maximum number of edge-disjoint paths is the minimum number of edges whose removal disconnects s from t.*

Corollary 3. *Manger's Theorem (Vertices): The maximum number of vertex-disjoint paths is the minimum number of vertices whose removal disconnects s from t.*

2 Bi-Segmentation

We can use min-cut to segment graphs into two segments A and B; more is **NP**-hard with this technique (but k-nearest neighbors can do it!).

Givens: A graph $\mathcal{G} = (V, E)$. For each $v \in V$, scores $a_v, b_v \geq 0$ that describe the probability that v belongs to A or B (e.g., based on a text analysis, image analysis, or similar). For each $e = (u, v) \in E$ (e.g., representing copurchases, adjacent pixels, or friends), a separation penalty p_e if u and v are assigned to different segments.

Goal: Assign each $v \in V$ to exactly one of $\{A, B\}$ while maximizing:

$$Q(A,B) := \sum_{v \in A} a_v + \sum_{v \in B} b_v - \sum_{u \in A, v \in B, e=(u,v)} p(u,v)$$

Reduction to min-cut: We rewrite $Q(A,B)$ as:

$$Q(A,B) := \sum_{v\in V} a_v + \sum_{v\in V} b_v - (\sum_{v\in B} a_v + \sum_{v\in A} b_v + \sum_{u\in A, v\in B, e=(u,v)} p(u,v))$$

$$Q(A,B) := C - Q'(A,B)$$

We notice that maximizing Q is equivalent to minimizing Q'. We add a new source s with edges to all v with capacity a_v and a new sink t with edges from all v with capacity b_v. Call this new graph $\mathcal{G}'$.

$$Q'(A,B) = c(A \cup \{s\}, B \cup \{t\}) \text{ in } \mathcal{G}'$$

Algorithm 2 Bi-Segmentation Algorithm

1: Build $\mathcal{G}'$.
2: Find the minimum (s,t) cut $(S, \bar{S})$.
3: Return $(S \setminus \{s\}, \bar{S} \setminus t\})$. {

3 Project Selection

Given n projects with values $p_i \in \mathbb{R}$ (i.e., can be negative) and dependencies (i,j) which mean that in order to complete i, we must first complete j.

Goal: Select a feasible set S of projects maximizing $\sum_{i\in S} p_i$.

Let the projects be nodes in a graph $\mathcal{G}$ and the dependencies (i,j) correspond to edges with capacity $c_e = \infty$ (i.e., they can't be cut). Then, observe:

$$\sum_{i\in S} p_i = \sum_i p_i - \sum_{i\in \bar{S}} p_i$$

$$= C - \sum_{i\in \bar{S}} p_i$$

Thus, we seek to minimize $\sum_{i\in \bar{S} p_i}$. Initially, we might do this with an edge of capacity p_i from s to each node i. However, this cannot handle negative capacities. Instead, we utilize t and say that if $p_i \geq 0$, create an edge from s to i of capacity p_i; otherwise, create an edge from i to t of capacity $-p_i \geq 0$.

Consider the minimum (s,t) cut $(A, \bar{A})$ in $\mathcal{G}$. We know this does not cut any dependency edges because of their infinite capacity, so one either completes a dependency sequence or does not start it. Thus, $A \setminus \{s\}$ is a feasible set.

$$c(A, \bar{A}) = \sum_{i\in A, p_i<0} (-p_i) + \sum_{i\in \bar{A}, p_i\geq 0} p_i$$

$$= \sum_{i\in A, p_i<0} (-p_i) + \sum_{i\in \bar{A}} p_i - \sum_{i\in \bar{A}, p_i<0} p_i$$

$$= \sum_{i, p_i<0} (-p_i) + \sum_{i\in \bar{A}} p_i$$

$$= C + \sum_{i\in \bar{A}} p_i$$

Thus, the minimum cut in $\mathcal{G}$ minimizes $\sum_{i\in \bar{A}} p_i$.

4 Sports Elimination

Given n sports teams, current numbers of wins w_i, and the number of remaining games $r_{i,j} \geq 0$ between teams i and j, is there a possible outcome of all remaining games such that the USC Trojans "win" (i.e., $w_{USC} \geq w_i \ \forall i$ with $r_{i,j} = 0 \ \forall i, j$)? We can assume without loss of generality that $r_{USC,j} = 0 \ \forall j$ because USC should without loss of generality win all of their remaining games.

For USC to win, each team i can win at most $x_i = w_{USC} - w_i$ of their remaining games. For each pair (i, j), $r_{i,j}$ games will be played, producing one winner each (no ties).

We generalize MBCM: Create a bipartite graph $\mathcal{G} = (A, B)$ with one node $u_{i,j}$ representing all games between i and j in A and one node v for each team in B. Let there be edges $\{(u_{i,j}, v_i), (u_{i,j}, v_j)\}$ with capacity $c_e = \infty$. Then, we create a source node s with edges $(s, u_{i,j})$ and $c_e = r_{i,j}$. Finally, we create a sink node t with edges (v_i, t) and $c_e = x_i$.

This problem is almost exactly like MBCM, but the edges out of s and into t do not necessarily have capacity one. The solution still works the same way, though.

Algorithm 3 Sports Elimination Algorithm

```
1: Build 𝒢.
2: Find an integral max-flow f.
3: if v(f) = Σ_{i,j} r_{i,j} (i.e., all games assigned a winner without violating residual win capacity) then
4:     The USC Trojans can win.
5: else
6:     The USC Trojans cannot possibly win.
7: end if
```