2 LP Meets Linear Regression

Note that the smallest value of z that satisfies $z \geq x, z \geq x$ is $z = |x|$. Now, consider the following linear programming problem:

$$\begin{array}{l} \min \sum_{i=1}^{n} z_i \\ \text{subject to} \begin{cases} y_i - (a + bx_i) \leq z_i & \text{for } 1 \leq i \leq n \\ (a + bx_i) - y_i \leq z_i & \text{for } 1 \leq i \leq n \end{cases} \end{array}.$$

Since $\sum_{i=1}^{n} z_i$ is minimized, z_i will be set to the $\max(y_i - (a+bx_i), (a+bx_i) - y_i)$. Note that $\max(y_i - (a+bx_i), (a+bx_i) - y_i)$ is, in fact, $|y_i - (a + bx_i)|$.
If for some solution we have that $z_i > |y_i - (a + bx_i)|$, then by setting $z_i = |y_i - (a + bx_i)|$ we will get a solution with a smaller value of the objective function, therefore the initial solution was not optimal. Hence, the constraints requires that the optimal solution will set $z_i = |y_i - (a + bx_i)|$, so the new problem is indeed equivalent to the original problem. However, now it is a linear programming problem.

3 Flow vs LP

(a) **Algorithm**: We create a bipartite graph with $m + n + 2$ nodes. Label two of the nodes as a "source" and a "sink." Label m nodes as suppliers, and n nodes as purchasers. Now, we will create the following edges:

- Create an edge from the source to supplier i with capacity $s[i]$.
- For each pair (i, j) in L, create an edge from supplier i to purchaser j with infinite capacity.
- Create an edge from purchaser j to the sink with capacity $b[j]$. We then plug this graph into our network flow solver, and take the size of the max flow as the number of dollars we can make.

Proof of correctness: We claim that the value of the max flow is precisely the maximum amount transactions we can make. To show this, we can show that a strategy of selling products corresponds exactly to a flow in this graph and vice versa.
For any flow, let $x_{i,j}$ be the amount of flow that goes from the node of supplier i to the node of purchaser j. Then, we claim a product selling strategy will sell exactly $x_{i,j}$ products from supplier i to purchaser j. This is a feasible strategy, since the flow going out of the node of supplier i is already bounded by $s[i]$ by the capacity of the edge from the source to that node, and similarly for the purchasers. Similarly, for the other direction, one can observe that any feasible selling strategy leads to a feasible flow of the same value.

(b) We define a variable $x_{i,j}$, denoting the amount of products we take from supplier i and sell to purchaser j. Then, we have the following linear program

$$\begin{array}{l} \max \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j} \\ \sum_{i=1}^{m} x_{i,j'} \leq b\,[j'], \text{ for all } j' \in [1, n] \\ \sum_{j=1}^{n} x_{i',j} \leq s\,[i'], \text{ for all } i' \in [1, m] \\ x_{i,j} = 0, \text{ for all } (i, j) \notin L \\ x_{i,j} \geq 0, \text{ for all } (i, j) \in L \end{array}$$

The linear program has mn variables and $\mathcal{O}(mn)$ linear inequalities, so it can be solved in time polynomial in m and n.

(c) Network flow is better. Linear programming is not guaranteed to find an integer solution (not even if one exists), so the approach in part (b) might yield a solution that would involve selling fractional products. In contrast, since all the edge capacities in our graph in part (a) are integers, the Ford-Fulkerson algorithm for max flow will find an integer solution. Thus, max flow is the better choice, because there are algorithms for that formulation that will let us find an integer solution.

4 A Cohort of Secret Agents

Solution. *We can think of each secret agent i as a unit of flow that we want to move from s_i to any vertex $\in T$. To do so, we can model the graph as a flow network by setting the capacity of each edge to 1, adding a new vertex t and adding an edge (t', t) for each $t' \in T$. We can add a source s and edges of capacity 1 from s to s_i. By doing so, we restrict the maximum flow to be k.*
Lastly, we need to ensure that no more than c secret agents are in a city. We want to add vertex capacities of c to each vertex. To implement it, we do the following: for each vertex v that we want add constraint to, we create 2 vertices v_{in} and v_{out}. v_{in} has all the incoming edges of v and v_{out} has all the outcoming edges. We also put a directed edge from v_{in} to v_{out} with edge capacity constraint c.
If the max flow is indeed k, then as every capacity is an integer, the Ford-Fulkerson algorithm for computing max flow will output an integral flow (one with all flows being integers). Therefore, we can incrementally starting at each secret agent i follow a path in the flow from s_i to any vertex in T. We iterate through all of secret agents to reach a solution.

5 Applications of Max-Flow Min-Cut

(a) The proposition is known as Hall's theorem. On one direction, assume G has a perfect matching, and consider a subset $X \subseteq L$. Every vertex in X is matched to distinct vertices in R, so in particular the neighborhood of X is of size at least $|X|$, since it contains the vertices matched to vertices in X.
On the other direction, assume that every subset $X \subseteq L$ is connected to at least $|X|$ vertices in R. Add two vertices s and t, and connect s to every vertex in L, and t to every vertex in R. Let each edge have capacity one. We will lower bound the size of any cut separating s and t. Let C be any cut, and let $L = X \cup Y$, where X is on the same side of the cut as s, and Y is on the other side, namely L/X. There is an edge from s to each vertex in Y , contributing at least $|Y|$ to the value of the cut. Now there are at least $|X|$ vertices in R that are connected to vertices in X. Each of these vertices is also connected to t, so regardless of which side of the cut they fall on, each vertex contributes one edge cut (either the edge to t, or the edge to a vertex in X, which is on the same side as s). Thus the cut has value at least $|X| + |Y| = |L|$, and by the max-flow min-cut theorem, this implies that the max-flow has value at least $|L|$, which implies that there must be a perfect matching.

(b) The proposition is known as Menger's theorem. By max-flow min-cut theorem, we only need to show that the max flow value from s to t equals the maximum number of edge-disjoint $s - t$ paths.
If we give each edge capacity 1, then the maxflow from s to t assigns a flow of either 0 or 1 to every edge (using, say, Ford-Fulkerson). Let F be the set of saturated edges; each has flow value of 1. Then extracting the edge-disjoint $s - t$ paths from the flow can be done algorithmically. Follow any directed path in F from s to t (via DFS), remove that path from F, and recurse. Each iteration, we decrease the flow value by exactly 1 and find 1 edge-disjoint $s - t$ path.
Conversely, we can transform any collection of k edge-disjoint paths into a flow by pushing one unit of flow along each path from s to t; the value of the resulting flow is exactly k.