Homework 9

Aleck Zhao

November 16, 2017

1 Graduation Requirements (34 points)

John Hopskins University[1] has n courses. In order to graduate, a student must satisfy several requirements of the form "you must take at least k courses from subset S". However, any given course cannot be used towards satisfying multiple requirements. For example, if one requirement says that you must take at least two courses from $\{A, B, C\}$, and a second requirement states that you must take at least two courses from $\{C, D, E\}$, then a student who has taken just $\{B, C, D\}$ would not yet be able to graduate as C can only be used towards one of the requirements.

Your job is to give an efficient algorithm for the following problem: given a list of requirements $r_1, r_2, \ldots, r_m$ (where each requirement r_i is of the form "you must take at least k_i courses from set S_i"), and given a list L of courses taken by some student, determine if that student can graduate.

(a) Given the m requirements and list of L courses taken (as above), design a flow network so that the maximum flow is $\sum_{i=1}^m k_i$ if and only if the student can graduatek.

Solution. Create an end node t. Then for each requirement r_i, create a node with an edge of capacity k_i to t. For each class $C \in L$, create a node and an edge to r_i if $C \in S_i$. Finally, create a start node s with an edge of capacity 1 to each class node.

The maximum possible flow is $\sum_{i=1}^m k_i$, so if this is attained, for each i, there must be k_i classes in S_i with flow 1 going to r_i to maintain flow conservation, which is exactly the requirement to graduate. If the student can graduate, then for each node r_i, there exists a set of k_i class nodes with flows values of 1 to r_i, which means the max flow is $\sum_{i=1}^m k_i$ by flow conservation. No class is applied to more than one requirement because the flow out of any class can only be 1 or 0 by integrality. □

(b) Using the previous part, design an algorithm for the problem which runs in $O(|L|^2 m)$ time. Prove correctness and running time.

Solution. The number of vertices is $2 + |L| + m = O(|L|)$ and the number of edges is at most $2 + |L| + m|L| = O(m|L|)$ if there is an edge from each class in L to each requirement r_i. Since all capacities are integers, using Ford-Fulkerson gives a running time of $O(F(m|L| + |L|) = O(Fm|L|$. Now, the maximum possible flow has a flow of 1 coming out of each vertex in L, so $F = O(|L|)$, so the running time is $O(|L|^2 m)$ as desired. This algorithm is correct by part (a). □

[1] `https://www.youtube.com/watch?v=JEH2ha1p0WA`

2 Removing Train Stations (33 points)

We discussed in class how one of the original motivations for studying max-flow and min-cut comes from WWII, and in particular the questions of "how much stuff can be moved from s to t along the rail network" and "what rail lines can we remove in order to destroy the ability to ship anything from s to t". Now suppose that instead of being able to remove rail lines, we have the ability to sabotage rail *stations*. Can we still find the best way to do this?

More formally, suppose that we are given a directed graph $G = (V, E)$, two vertices $s, t \in V$ with $(s, t) \notin E$, and a function $c : V \setminus \{s, t\} \to \mathbb{R}^{\geq 0}$. As usual, $|E| = m$ and $|V| = n$. We will call $c(v)$ the *cost* of v. Your goal is to find the subset $S \subseteq V \setminus \{s, t\}$ so that there is no path from s to t in $G - S$ (the graph obtained from G by removing all vertices in S and all edges incident on at least one vertex in S) which minimizes $\sum_{v \in S} c(v)$.

Design an algorithm for this problem which runs in $O(mn)$ time. You may assume that $m \geq n$ and that there is an algorithm for maximum flow which runs in $O(mn)$ time (as we discussed in class). Prove running time and correctness.

Solution. Construct the following graph: For each vertex $v \in V \setminus \{s, t\}$, create two vertices v_{in} and v_{out}. Then for every edge $(s, v) \in E$, construct the edge (s, v_{in}) and for every edge $(v, t) \in E$, construct the edge (v_{out}, t). Next, for every remaining edge (u, v) not incident on either s or t, construct the edge (u_{out}, v_{in}). At this point, the graph is disconnected because there are no edges leaving v_{in} and no edges into v_{out}. Finally, for every $v \in V$, construct the edge (v_{in}, v_{out}).

Set the capacity of the edge (v_{in}, v_{out}) as $c(v), \forall v \in V$, and set the capacities of all other edges to ∞. This new graph has $2n - 2 = O(n)$ vertices and $m + n = O(m)$ edges. We can compute a max-flow on this graph in $O(mn)$ time, and also derive the set of edges that comprise the min cut by looking at the residual graph.

In this graph, if we remove the edge (v_{in}, v_{out}), then this is equivalent to removing vertex v in the original graph because this is the only edge leaving v_{in} and the only edge into v_{out}, so after removing it, v_{out} is unreachable, and we cannot use any of the edges leaving v_{out}, which is the same as the edges leaving v, and we cannot use any edges into v_{in} since there are no edges out of v_{in}.

Thus, a feasible cut is removing all edges of the form (v_{in}, v_{out}), which has finite capacity $\sum_{v \in V} c(v)$, so the min cut must also be finite, and thus only remove these edges, and none of the edges with infinite capacity. The max flow is also finite, and will find this same min cut $(T, \overline{T})$, so $S = \left\{v : (v_{in}, v_{out}) \in (T, \overline{T})\right\}$. □

3 Linear Programming (33 points)

Suppose that we are given points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ in the plane, and we want to find a line $y = ax + b$ which is "close" to these points. Different definitions of "close" result in different problems.

(a) Let $h_1(a, b) = \max_{i=1}^n |y_i - ax_i - b|$. Note that $h_1(a, b) = 0$ if and only if $y_i = ax_i + b$ for all i, so we can think of $h_1(a, b)$ as a notion of the "error" of the line $y = ax + b$. Using linear programming, give an algorithm which finds the values (a, b) which minimize $h_1(a, b)$ in polynomial time. Prove correctness (you do not need to prove running time, but only polynomial-time algorithms will be accepted).

Solution. Suppose $m = h_1(a, b)$ is minimum value of the error. Then

$$|y_i - ax_i - b| \le m \implies \begin{cases} y_i - ax_i - b \le m \\ y_i - ax_i - b \ge -m \end{cases}$$

for all i. Thus, we have the linear program

$$\begin{aligned} \min \quad & 0 \cdot a + 0 \cdot b + 1 \cdot m \\ & -x_1 a - b - m \le -y_1 \\ & -x_1 a - b + m \ge -y_1 \\ & \vdots \\ & -x_n a - b - m \le -y_n \\ & -x_n a - b + m \ge -y_n \end{aligned}$$

From here, we may run any linear program solver that runs in polynomial time. At termination, we will also have the optimal vector $\begin{bmatrix} a & b & m \end{bmatrix}$ which gives the values of a and b. The algorithm is correct because this optimal vector satisfies all the constraints, which means m is indeed max of all $|y_i - ax_i - b|$, and it is the minimum value since the linear program minimized m. □

(b) Let $h_2(a, b) = \sum_{i=1}^n |y_i - ax_i - b|$. As in the previous part, note that $h_2(a, b) = 0$ if and only if $y_i = ax_i + b$ for all i, so we can think of $h_2(a, b)$ as a notion of the "error" of the line $y = ax + b$. Using linear programming, give an algorithm which finds the values (a, b) which minimize $h_2(a, b)$ in polynomial time. Prove correctness (you do not need to prove running time, but only polynomial-time algorithms will be accepted).

Solution. Set $u_i := y_i - ax_i - b$ and $v_i := |y_i - ax_i - b|$ for all i. Then $h_2(a, b) = \sum_{i=1}^n v_i$. Then we have $u_i \le v_i$ and $u_i \ge -v_i$, and the equality constraints $u_i = y_i - ax_i - b$ for all i. Thus, we can formulate this as a linear program in terms of $a, b, u_1, \cdots, u_n, v_1, \cdots, v_n$ as

$$\begin{aligned} \min \quad & \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 & \cdots & 1 \end{bmatrix}^T \begin{bmatrix} a & b & u_1 & \cdots & u_n & v_1 & \cdots & v_n \end{bmatrix} \\ & u_1 - v_1 \le 0 \\ & u_1 + v_1 \ge 0 \\ & \vdots \\ & u_n - v_n \le 0 \\ & u_n + v_n \ge 0 \\ & u_1 + x_1 a + b = y_1 \\ & \vdots \\ & u_n + x_n a + b = y_n \end{aligned}$$

From here, we may run any linear program solver that runs in polynomial time. At termination, we will also have the optimal vector that gives the values of a and b. The algorithm is correct because this vector satisfies all the constraints, and the program minimizes $\sum_{i=1}^{n} v_i = \sum_{i=1}^{n} |y_i - ax_i - b| = h_2(a, b)$. $\square$