CSE 317 Design and Analysis of Algorithms *Assigned: Sunday, 14 Apr*

Problem Set 4

Spring 2024 *Due: 11:59 pm Monday, Apr 29*

Alina Afghan 24491.

1. Let's design a graph for network flow. Let's first assign a source node s and a sink node t. Next, let's assign a node r_i to each of our requirements, and a node c_i to each of our courses. We'll draw an edge of capacity 1 going from r_i to every c_i that is contained within the requirements of that node. Then we'll draw an edge of capacity 1 from each course node til the sink node, of capacity 1, since each course can only be used to satisfy ONE requirement. Finally, we'll keep capacity k_i from the source node til each of the requirement nodes where k_i = the number of courses needed within that requirement, and try to push flow equal to the sum of each edge capacity leading out from the source. If the max flow equals that sum of k_i, then the student can graduate.

   We can prove this by showing that since we know that in order to graduate, a student needs to fulfill some number of requirements of the form 'you must take at least k_i courses from subset S_i', the number of courses the student needs overall is the sum of every k_i.

   The graph we formed has sum of k_i capacity leading out from the source node til the requirement, and each requirement has an edge with capacity one going to the courses within that requirement only, so we will have edges going til every course node within that requirement i. Since the course nodes going to the sink have capacity one, each course can only be used once. If we push flow of k_i, it will follow the outlined path and reach the sink node. We can apply Edmond-Karp algorithm on this graph construction to find the flow.

2. We can begin to construct a solution for this by first reducing the problem to a bipartite graph, with a source node s, a sink node t, n course nodes and m problem nodes. We'll draw directed edges from the source node to each of the problem nodes with a capacity equal to the value given by solving that given problem. Every concept node needed by problem p_j will be connected by a directed edge from the problem til the concept, with a capacity of infinity, and each concept node will then in turn be connected to the sink node with an edge of capacity C_i, the cost of understanding that concept. We can now use the s-t cut algorithm to find the min-cut of this graph, and output the problems and concepts on the left side of our cut as our solution.

   To understand why this works, we try to solve the problem of MINIMIZING the cost of the concepts we understand PLUS the sum value of the problems we did NOT solve, thus maximizing the value of problems we did solve. This will give us the same answer as MAXIMIZING the sum value of problems we DID solve, MINUS the cost of the concepts we understand, since the x that maximizes f(x) is the same x that minimizes -f(x). (If the max value for a function f(x) is a, then $f(x) \leq f(a)$ for all x, which implies $-f(x) \geq -f(a)$) for the negation function -f(x), showing that it is the min.)

   Let's call the total value given by the cost of the concepts we understand PLUS the sum value of the problems we did NOT solve as k. First of all, the min-cut will divide our graph into two disjoint subsets, one containing the source node s and the other containing the sink node t. It will not cut any of the infinite edges, meaning whichever subset contains P_j will also contain its required concepts, i.e we never solve a problem without understanding its concepts. The capacity of the s-t cut is the sum of all edge capacities between s and the problems on the t-side, plus all edge capacities between t and the concepts on the s-side.

3. **First Program:**

$$\begin{aligned}
\text{minimize} \quad & (2y_1 + 6y_2 + y_3) \quad \text{(Objective Function)} \\
\text{subject to} \quad & y_1 + 7y_2 + 2y_3 \geq 1 \quad \text{(Constraint 1)} \\
& y_1 + 2y_2 + y_3 \geq 3 \quad \text{(Constraint 2)} \\
& 2y_1 + 5y_2 - y_3 \leq -2 \quad \text{(Constraint 3)} \\
& y_1, y_2, y_3 \geq 0 \quad \text{(Non-negativity Constraint on the 3 variables)}
\end{aligned}$$

Second Program:

$$\begin{aligned}
\text{minimize} \quad & (5y_2 - 2y_1) \quad \text{(Objective Function)} \\
\text{subject to} \quad & -3y_1 \geq 1 \quad \text{(Constraint 1)} \\
& 2y_2 \leq -3 \quad \text{(Constraint 2)} \\
& -2y_1 - y_2 \geq 2 \quad \text{(Constraint 3)} \\
& y_1, y_2 \geq 0 \quad \text{(Non-negativity Constraint on the 2 variables)}
\end{aligned}$$

4. First we can define our constraints as follows:

$p_{00|00} + p_{10|00} = p_{00|10} + p_{10|10}$

$p_{01|00} + p_{11|00} = p_{01|10} + p_{11|10}$

$p_{00|01} + p_{10|01} = p_{00|11} + p_{10|11}$

$p_{01|01} + p_{11|01} = p_{01|11} + p_{11|11}$

$p_{00|10} + p_{10|10} = p_{00|00} + p_{10|00}$

$p_{01|10} + p_{11|10} = p_{01|00} + p_{11|00}$

$p_{00|11} + p_{10|11} = p_{00|01} + p_{10|01}$

$p_{01|11} + p_{11|11} = p_{01|01} + p_{11|01}$

$p_{00|00} + p_{01|00} = p_{00|01} + p_{01|01}$

$p_{10|00} + p_{11|00} = p_{10|01} + p_{11|01}$

$p_{00|10} + p_{01|10} = p_{00|11} + p_{01|11}$

$p_{10|10} + p_{11|10} = p_{10|11} + p_{11|11}$

$p_{00|01} + p_{01|01} = p_{00|00} + p_{01|00}$

$p_{10|01} + p_{11|01} = p_{10|00} + p_{11|00}$

$p_{00|11} + p_{01|11} = p_{00|10} + p_{01|10}$

$p_{10|11} + p_{11|11} = p_{10|10} + p_{11|10}$

$p_{00|00} \geq 0$

$p_{00|01} \geq 0$

$p_{00|10} \geq 0$

$p_{00|11} \geq 0$

$p_{01|00} \geq 0$

$p_{01|01} \geq 0$

$p_{01|10} \geq 0$

$p_{01|11} \geq 0$

$p_{10|00} \geq 0$

$p_{10|01} \geq 0$

$p_{10|10} \geq 0$

$p_{10|11} \geq 0$

$p_{11|00} \geq 0$

$p_{11|01} \geq 0$

$p_{11|10} \geq 0$

$p_{11|11} \geq 0$

$p_{00|00} + p_{01|00} + p_{10|00} + p_{11|00} = 1$

$p_{00|01} + p_{01|01} + p_{10|01} + p_{11|01} = 1$

$p_{00|10} + p_{01|10} + p_{10|10} + p_{11|10} = 1$

$p_{00|11} + p_{01|11} + p_{10|11} + p_{11|11} = 1$

After formulating these constraints, we'll convert the input into two matrices, one equality matrix and one inequality matrix, and enter both of those into cdd to convert to vertex form. Once the program outputs the vertices, we can depict them as follows, removing the first digit from each row since it merely signified that the vertex is an extreme point.

V-representation

begin

(1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0)

(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1)

(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)

(1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0)

(1/2, 0, 0, 1/2, 1/2, 0, 0, 1/2, 0, 1/2, 1/2, 0, 1/2, 0, 0, 1/2)

(1/2, 0, 0, 1/2, 1/2, 0, 0, 1/2, 1/2, 0, 0, 1/2, 0, 1/2, 1/2, 0)

(1/2, 0, 0, 1/2, 0, 1/2, 1/2, 0, 0, 1/2, 1/2, 0, 0, 1/2, 1/2, 0)

(1/2, 0, 0, 1/2, 0, 1/2, 1/2, 0, 1/2, 0, 0, 1/2, 1/2, 0, 0, 1/2)

(0, 1/2, 1/2, 0, 1/2, 0, 0, 1/2, 0, 1/2, 1/2, 0, 0, 1/2, 1/2, 0)

(0, 1/2, 1/2, 0, 1/2, 0, 0, 1/2, 1/2, 0, 0, 1/2, 1/2, 0, 0, 1/2)

(0, 1/2, 1/2, 0, 0, 1/2, 1/2, 0, 0, 1/2, 1/2, 0, 1/2, 0, 0, 1/2)

(0, 1/2, 1/2, 0, 0, 1/2, 1/2, 0, 1/2, 0, 0, 1/2, 0, 1/2, 1/2, 0)

(0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0)

(0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1)

(0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0)

(0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0)

(0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1)

(0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0)

(0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0)

(0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0)

(0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0)

(0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0)

(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0)

(0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1)

end