Homework #7
Introduction to Algorithms
601.433/633
Spring 2020

Due on: Wednesday, April 29th, 12pm

Problem 1 (15 points)

For a directed graph $G = (V, E)$ with capacities $c : E \to \mathbb{R}^+$ and a flow $f : E \to \mathbb{R}^+$, the *support* of the flow f on G is the set of edges $E_{\mathsf{supp}} := \{e \in E \mid f(e) > 0\}$, i.e. the edges on which the flow function is positive.

Show that for any directed graph $G = (V, E)$ with non-negative capacities $c : E \to \mathbb{R}^+$ there always exists a maximum flow $f^* : E \to \mathbb{R}^+$ whose *support* has no directed cycle.

Hint: Proof by contradiction?

Proof. First, given a flow network $G = (V, E)$, denote its maximum flow as f and assume that the support of f has a directed cycle. Repeat the following steps until there is at least one edge in the cycle whose flow is zero:

1. On the edges of this cycle, find the edge with minimum value of flow and denote the this value as m.
2. Reduce the flow of each edge in the cycle by m.

In this way, the new flow is still a max flow since the flow in the cycle will not go out of the cycle. Thus, the zero-flow edges does not belong to the support of max flow and then the cycle breaks. □

Problem 2 (20 points)

In a graph $G = (V, E)$, a matching is a subset of the edges $M \subseteq E$ such that no two edges in M share an end-point (i.e. incident on the same vertex).

Write a linear program that, given a bipartite graph $G = (V_1, V_2, E)$, solves the maximum-bipartite-matching problem. I.e. The LP, when solved, should find the largest possible matching on the graph G.

Clearly mention the variables, constraints and the objective function. Prove why the solution to your LP solves the maximum-bipartite-matching problem. You may assume that all variables in the solution to your LP has integer values.

Proof. For every $e \in E$, we use a variable a_e to indicate whether e is in the matching. And we use $v \in e$ to denote that v is a vertex of e. So the LP reads as the following,

$$\max \quad \sum_{e \in E} a_e \tag{1}$$

$$\text{s.t.} \quad \sum_{e : v \in e} a_e \leq 1, \forall v \in V \tag{2}$$

$$0 \leq a_e \leq 1, \forall e \in E \tag{3}$$

The objective requires to maximize the number of edges in the matching, the first constraint ensures that every vertex is used at most once, and the second constraint ensures that every edge is either not used or used at most once.

Correctness We will assume that there is a solution to the LP (maximizing the objective) that has integer solutions. The correctness follows easily by a proof of contradiction – assume there was a max-matching that was strictly greater than the solution to the LP, note that one can assign 1 to every edge picked in that matching and achieve a larger objective value while satisfying all the constraints hence contradicting the fact that the solution maximized the objective value.

Comment on integrality: This matrix corresponding to the constraints of the LP has the property that it is *unimodular*. Solutions to such types of LPs are known to

have integer solutions. See for e.g.,

`https://en.wikipedia.org/wiki/Unimodular_matrix`

□

Problem 3 (15 points)

In class you saw that VERTEX-COVER and INDEPENDENT-SET are related problems. Specifically, a graph $G = (V, E)$ has a vertex-cover of size k *if and only if* it has an independent-set of size $|V| - k$.

We also know that there is a polynomial-time 2-approximation algorithm for VERTEX-COVER. Does this relationship imply that there is a polynomial-time approximation algorithm with a constant approximation ratio for INDEPENDENT-SET? Justify your answer.

Proof. It does not imply the existence of an $O(1)$-approximation algorithm for the maximum size independent-set. Suppose that we had in a graph of size n, the size of the smallest vertex cover was k, and we found one that was size $2k$. This means that we found an independent set of size $n - 2k$, when the true size of the largest independent-set was $n - 2k$. So, to have it be a constant factor approximation, we need that there is some λ so that we always have $n - 2k \geq \lambda(n - k)$. However, if we had that k was close to $n/2$ for our graphs, say $n/2 - \epsilon$, then we would have to require $2\epsilon \geq \lambda n/2 + \epsilon$. Since we can make ϵ stay tiny, even as n grows large, this inequality cannot possibly hold. □

Problem 4 (10 points)

You are given a biased coin c but you dont know what the bias is. I.e. the coin c outputs *H* with probability $p \in (0, 1)$ (note that it is not inclusive of 0 and 1) and *T* with probability $1 - p$ every time you toss it but you don't know what p is.

Can you simulate a fair coin using by tossing c? I.e. Come up with a "rule" for tossing c (possible several times) and outputting *H* or *T* based on the output of

the coin tosses of c such that you will output *H* with probability $1/2$ and *T* with probability $1/2$? Please prove the correctness of your solution.

Hint: i) What if you had two coins c_1, c_2 with the same bias instead of one? Can you toss them together and output *H* or *T* based on the output? ii) You shouldn't be trying to "estimate" p.

Proof. Let 0 and 1 denote the two outputs of the coin and assume the probability of 0 is p.

We keep throwing the coin twice until the following event happens: the output of the two throws are different.

Let t_1, t_2 be the random variables denoting the two throws that were different. We output the value of t_2 as the toss of the fair coin.

By the Bayesian formula and the fact that the throws are independent, we know that,

$$\begin{aligned}\mathbb{P}[t_2 = 1 | t_1 \neq t_2] &= \frac{\mathbb{P}[t_2 = 1, t_1 \neq t_2]}{\mathbb{P}[t_1 \neq t_2]} \\ &= \frac{\mathbb{P}[t_2 = 1, t_1 = 0]}{\mathbb{P}[t_1 \neq t_2]} \\ &= \frac{p(1-p)}{2p(1-p)} = \frac{1}{2}\end{aligned}$$

□