CS170 — Fall 2017— Homework 8 Solutions

Jonathan Sun, SID 25020651

0. Who Did You Work With?

Collaborators: Kevin Vo, Aleem Zaki, Jeremy Ou

1. iPhone X

(a) The main idea of this problem is that for all the demand to be met, there needs to be more knockoff iPhones being produced than the distributors demand. This can be represented as:

$$\sum_{i=1}^{m} a_i \geq \sum_{i=1}^{n} b_i.$$

(b) The main idea of this problem is that when a factory can distribute knockoff iPhones to a distributor, we want to prioritize the factory to deliver to distributors with few factories delivering to it. Storing into an array which factories, i, that can deliver to a distributor, j, so that $d_{i,j} \leq c_i$, allows us to know which factories can distribute to a distributor. Then, we check if the total number of knockoff iPhones being produced is greater than or equal to the demand. Now, if a factory can supply knockoff iPhones to two distributors, with one distributor with more factories than another, then we want for the factory to prioritize distributing knockoff iPhones to the distributor with fewer factories. This can be done by taking a_i of every factory in the array and subtracting the number of knockoff iPhones that are delivered while prioritizing the distributors with fewer factories. This is correct because we first check to see if every factory within the distance from the distributor(s) can create enough knockoff iPhones and then we subtract the total number of knockoff iPhones manufactored from that factory, and prioritizing distributors with fewer factories, so that if a factory is within the distance of more than one distributor.

(c) The main idea of this problem is that we first need to try to satisfy the demand domestically for the different countries so we can try to limit the amount of knockoff iPhones being imported or exported. If a country does not create enough knockoff iPhones for its citizens, then that country will need to import knockoff iPhones. If that country has excess knockoff iPhones, then that country can export knockoff iPhones. So, only if a country does not have enough knockoff iPhones or has excess knockoff iPhones will e_k or f_k be affected.

2. What happens in Vegas...

(a) max $\{z\}$

$10x_1 - 4x_2 - 6x_3 + z \leq 0$

$-3x_1 + 1x_2 + 9x_3 + z \leq 0$

$-3x_1 + 3x_2 - 2x_3 + z \leq 0$

$x_1 + x_2 + x_3 = 1$

$x_1, x_2, x_3 \geq 0$

Therefore, the optimal strategy is:

$x_1 = 0.335$

$x_2 = 0.563$

$X_3 = 0.102$

The expected payoff is -0.480.

(b) max $\{z\}$

$10x_1 - 3x_2 - 3x_3 + z \leq 0$

$-4x_1 + 1x_2 + 3x_3 + z \leq 0$

$-6x_1 + 9x_2 - 2x_3 + z \leq 0$

$x_1 + x_2 + x_3 = 1$

$x_1, x_2, x_3 \geq 0$

Therefore, the optimal strategy is:

$x_1 = 0.268$

$x_2 = 0.323$

$x_3 = 0.409$

The expected payoff is -0.480.

3. Minimum Spanning Trees

(a) The purpose of the objective function is to find the minimum spanning tree by generating all possible spanning trees and then finding the minimum spanning tree from those possible spanning trees. In the integer linear program, $w_{u,v}$ is the edge weight between vertices u, v and $x_{u,v}$ is a decision variable to see if the edge belongs in a spanning tree. One constraint is that all the nodes in a spanning tree needs to be connected to the spanning tree since a partition will cut a spanning tree into two parts. This means that the edge is needed to make it a spanning tree. Another constraint is that $x_{u,v}$ has to be either 0 or 1, which is symbolizes true or false and this is used to check whether or not an edge belongs in the spanning tree. The last constraint is that every edge has to be a valid edge.

(b) Creating the formulation is not a polynomial time algorithm because the algorithm finds all possible spanning trees first and then selects the minimum spanning tree. So, generating all possible spanning trees from n nodes gives $n^{(n-2)}$ spanning trees. This means that the runtime is about $O(n^n)$, which is much worse than polynomial time.

(c) If $x_{u,v} \geq 0$, then it could be less than 1 instead of being either 0 or 1. This means that $w_{u,v}x_{u,v}$ can be a smaller value which in turn means that you can scale the weights of the tree differently. This achieves a better objective value because the objective value can now be further minimized.

4. Decision vs. Search vs. Optimization

(a) **Main Idea:**
The main idea is that the algorithm will first call DECISION to check if it returns true or not. If it returns false then there is no solution. If it returns true, then the algorithm will go through all the vertices in the graph and check if removing that vertex and its outgoing edges will give vertex cover of size at most $b - 1$. This will be done repeatedly until we find a vertex cover of size at most b.

Proof of Correctness:
The algorithm works because a vertex cover will include all vertices where each edge is incident to at least a vertex in the graph. So, if we remove a vertex that is part of the vertex cover, then there should be at most $b - 1$ vertices left in the vertex cover. The algorithm will therefore go through all vertices and check if removing it will give $b - 1$ vertices in the vertex cover. If so, we continue to remove another vertex and its incident edges and repeat the process. If not, then we know that that vertex is not part of the vertex cover and so we test another vertex. □

Run Time:
$O(|V|^2 \times$ runtime of DECISION).

Justification:
The runtime is $O(|V|^2 \times$ runtime of DECISION) because the first time we check the vertices in the graph, we will call on DECISION up to $|V|$ times. However, as we remove vertices, we will call on DECISION up to $|V - 1|$ times, $|V - 2|$ times, etc. until there are no more vertices. So, the algorithm will call on DECISION $\sum_{i=0}^{V} V - i$ times. This can be approximated to calling DECISION $|V|^2$ times and so the total runtime is $O(|V|^2 \times$ runtime of DECISION).

(b) **Main Idea:**
The main idea is that the algorithm will keep on doing a binary search of b for SEARCH and checking if this gives a minimum vertex cover or not. So, the first time we call SEARCH, we call it with b and if this returns true then we know that there is a vertex cover of size at most b. Then, we call SEARCH on $\frac{1}{2}b$ and if it returns true, then we know that there is a vertex cover of size at most $\frac{1}{2}b$. Else, we know that there is a minimum vertex cover between $\frac{1}{2}b$ and b. So, we call SEARCH on $\frac{3}{4}b$ since $\frac{3}{4}b$ is halfway between $\frac{1}{2}b$ and b. This idea continues until we find a minimum vertex cover.

Proof of Correctness:
The algorithm works because we are essentially doing a binary search with our input b into SEARCH. Because SEARCH can take in any value and check if there exists a vertex cover of size at most of that value, every time SEARCH is called the input value becomes more and more accurate. □

Run Time:
$O(\log b \times$ runtime of SEARCH).

Justification:
The runtime is $O(\log b \times$ runtime of SEARCH) because a binary search has a runtime of $O(\log b)$. Since we are calling SEARCH $\log b$ times, the runtime will be $O(\log b \times$ runtime of SEARCH).

5. Max-Flow Variants

(a) The idea is that since every vertex has a capacity, I can modify graph G by adding another outgoing edge and vertex for every vertex. This outgoing edge will have capacity of c_v. Thus, we can treat this as a normal Max-Flow problem with extra edges that have capacities c_v and no capacities on the vertices. The flows will be the same size regardless if F is a flow in G or if F' is a flow in G' because the incoming flows will be the same as the outgoing flows.

(b) The idea is that since there can be multiple source nodes and we want to maximize the total flow coming out of all the sources, we need to make the capacities of the edges really big, preferably ∞, so that they can hold all of the flows from the sources. So, the capacities should be lower bounded by the sum of all the flow from the sources.

(c) The idea is that since there is both a demand that needs to be met and that every vertex has a capacity, I can modify the idea from part (a) to also include another source and also take the idea from part (b) to have an upperbound for all edges capacities to be at least greater than the demand, preferably ∞.