Problem Set 6 Solutions

This problem set is due **at 10:00pm** on **Wednesday, April 15, 2020**.

EXERCISES (NOT TO BE TURNED IN)

Max Flow

- Do Exercises 26.1-6 in CLRS (pg. 714)
- Do Exercises 26.2-10 in CLRS (pg. 731)

Linear Programming

- Do Exercise 29.1-9 in CLRS (pg. 858)
- Do Exercise 29.2-6 in CLRS (pg. 864)
- Do Exercise 29.5-9 in CLRS (pg. 893)

Problem 6-1. Intercepting the Smugglers [40 points]

You are commanding a small border patrol outpost and you are tasked with stopping the smuggling of contraband from the infamous Sorkis system to the Tarconis system. To do that, you need to secure some of the intermediate systems (unfortunately, both Sorkis and Tarconis are off limits). Once a system is secured, any smuggler traveling through that system will be intercepted.

The number of security details at your disposal is quite limited, though. Can you find a deployment that is guaranteed to intercept any smuggler, no matter which route he/she chooses, while using the minimal possible number of security details (one per system)? Your algorithm should identify at which systems security details should be placed.

Assume you are given a hyperspace map represented as a graph G with each vertex corresponding to a system and each directed edge representing a hyperspace lane. (Assume also that there is no hyperspace lane that directly connects Sorkis to Tarconis.)

(a) [20 points] Devise an efficient (i.e., polynomial-time) algorithm that finds a minimum-size subset of systems such that once each one of these systems is secured, any smuggler trying to get from Sorkis to Tarconis will be intercepted. State the running time of your algorithm and briefly justify its correctness.

Solution:

Approach: At first glance, it looks like we need to find the minimum number of systems along a path from Sorkis (our source, s), and Tarconis (our sink, t). We want to find a minimum cut of *vertices* that will disconnect s from t. We know a lot about minimum cuts of *edges*. We need to find a way to model minimum cuts of vertices. An idea is to transform the graph G so that there are edges in the new graph corresponding to vertices from G. With the right approach, we can force any edge min-cut to contain *only* these new edges. That way a min-cut in the new graph will correspond naturally to a set of vertices in G. A second idea is to use infinite capacities on all the original edges and finite capacities for the new edges. That way, any min-cut, if finite, must contain only the special edges.

Solution: Given our original graph $G = (V, E)$, we build a new graph $G' = (V', E')$ as follows. We add $s = s_{out}$ and $t = t_{in}$ to V'. For every $v \in V - \{s, t\}$ we include two vertices v_{in} and v_{out} in V'. We also add an edge pointing from v_{in} to v_{out} with capacity 1. We call these edges v-edges (as they correspond to original vertices of G). For every edge $e = (u, v) \in E$, we add an edge (u_{out}, v_{in}) to E' with capacity ∞.

The idea behind the construction is that any $s - t$ path in G corresponds naturally to a path in G' and yet a finite s-t cut of G' includes only v-edges which are in 1-1 correspondance with vertices in $V - \{s, t\}$. Given that there is no

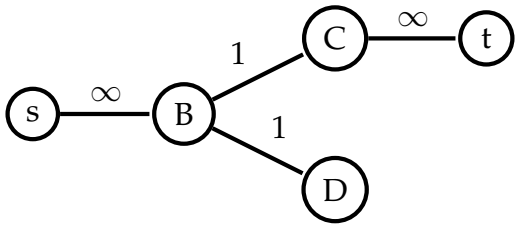
direct connection between s and t in G, any $s-t$ path in G goes through an internal system and, therefore, its associated path goes through a v-edge. Note that, in particular, this means that the minimum cut of G' is finite. This follows from the flow decomposition theorem. Indeed, any $s-t$ path in G' goes through a v-edge and, therefore, can carry flow of at most 1.

We now argue that a finite minimum s-t cut in G' defines a minimum set of vertices to guard in G. We first show that any finite cut C of G' defines a subset $U \subseteq V$ of size at most $|C|$ that touches every $s-t$ path in G. Otherwise, there would be an associated $s-t$ path in G' which wouldn't cross an edge of C. Conversely, for any subset $U \subseteq V$ that touches every path in G, its associated set of v-edges form a cut C in G'. Indeed, if there is a path in G' that does not cross C, then its associated path in G wouldn't touch U.

To find the size of the min-cut, we run a max-flow algorithm on our graph G'. By strong duality, the size of the max flow equals the size of the minimum cut. The transformed graph has $O(|V|)$ vertices and $O(|E| + |V|) \in O(|E|)$ edges, because for each $v \in V - \{s, t\}$ we added a single edge and created two vertices v_{in} and v_{out}. Our max-flow is bounded by the min-cut, which can be at most $|V|$ because we can cut through at most the V unit-capacity edges we created. Thus, using Ford-Fulkerson, we can find the max-flow in $O(|E||V|)$ time.

How do we actually recover the set of edges in the minimum cut? We present a general algorithm to find the edges in a minimum cut. We start with the last residual graph G_f with a cut with capacity 0 and we look at the saturated edges. Not all saturated edges belong to the min-cut. Only those edges that actually constitute an $s-t$ cut. We can find these edges by running a BFS search from s, avoiding any saturated edge, to form a set S of s-reachable nodes. Then S versus $V \backslash S$ is a cut and only saturated edges cross it. This process, starting from G_f, takes time $O(|V| + |E|)$.

Common mistakes: Most students attempted to find a min-cut on the original graph G. Some erroneously put unit capacities on all edges in G, which may find an incorrect min-cut (consider a graph from $s \to a \to t$ where all edges have capacity 1– a valid min-cut might slice through the edge from s to a, but this doesn't work– we would then secure s, and that's not valid). Others assigned capacities correctly and then selected the vertices on one endpoint of the edges that formed the cut. Note that this is not guaranteed to select the optimal number of vertices. Consider the following graph G:



The min-cut clearly crosses through the edges connecting $B \to C$ and $B \to D$ and has a value of 2, so we might have selected to secure C and D, but this is suboptimal– we only need to secure B. Some students attempted to mitigate this problem by finding the sets of vertices on both sides of the cut and securing the smaller set, but the optimal set of bases to secure may have vertices on either side of the min-cut, which is why a graph transformation is necessary.

A word about using ∞ capacities: Obviously, computers cannot handle infinite numbers. Many programming languages have 'Infinity' built iin a way which is consistent with other arithmetic operations. In our situation, we could also model an infinite capacity with a number large enough to prevent such an edge to be in any minimum cut. What number to use? We know that there is a minimum cut consisting of all the v-edges. This cut has size $|V|$. Therefore, using the value $M = |V| + 1$ suffices.

(b) [20 points] It turns out that smugglers' ships have a one-time, hyperboosting capability, which allows a smuggler to evade interception by a security detail *once*. So, as long as a smuggling route passes through *at most one* secured system, smugglers still will succeed. Adjust your algorithm from the previous part to accommodate this modification and still intercept all smugglers. No justification is required. (Assume that there is no pair of hyperspace lanes that connect Sorkis to Tarconis while passing through just one intermediate system.)

Solution: We build from ideas of part (a). Whereas part (a) looked for a smallest set of vertices that block every $s - t$ path, we are now looking for a smallest set of vertices $U \subseteq V$ such that U intersects every $s - t$ path in two vertices.

First, note that if the distance between s and t in G is at most 1, then the task is impossible. The smugglers can always take this path and use their hyperboosting capability if the single node is guarded. Luckly, we can test whether this is the case in $O(|E|)$ time by looping through all the vertices and testing whether a given node has an edge from s and an edge to t. So, for now on, we assume that the minimal distance between s and t is 2.

We use ideas similar to those in problem 4-2(b). We build a new graph G'' with two copies (G'_1 and G'_2) of the graph G' from part (a). We add a new node s and direct edges with ∞ capacity to the two s_{out} nodes. Similarly, we add a new node t and connect edges with ∞ capacity from the two t_{in} nodes to t. Further, for every node $v \in V$, we connect the v_{in} node from G'_1 with the v_{out} node from G'_2 with an edge with capacity ∞. We call these edges h-edges.

Intuitively, we view the graph G'' as a map for the smugglers. An $s-t$ path in G'' has to take an h-edge and this is where the smugglers use their hyper-boosting capability. Note that all h-edges point from G'_1 to G'_2 so they only get to use the capability once.

We claim that the vertices associated with a smallest $s-t$ cut in G'' give us a smallest set of vertices that touch every $s-t$ path of G in two places. Note that no justification of this fact was needed for full credit. For completeness, we prove the claim bellow. We can find the edges of the minimum $s-t$ cut with the algortihm presented in part (a). The running time is the same as before as $|V''| = O(|V|)$ and $|E''| = O(|V| + |E|) = O(|E|)$.

Addendum to the solution: We now prove the claim that the construction works. An important fact to keep in mind is that any path p'' in G'' is associated with a path in G comprised of the nodes associated with the v-edges and the h-edge of p'' in the order they are traversed. Further, any path in G'' utilizes exactly one h-edge.

For starters, we note that the minimum $s-t$ cut in G'' is finite. To see this, we argue that the maximum flow is finite. We use the fact that any flow can be decomposed into a finite sum of paths and cycles (cycles do not contribute to the size of the flow). Take any path with positive flow. We argue that the flow is bounded by 1. This is a consequence of the fact that every $s-t$ path in G has at least two internal vertices. As a result, every $s-t$ path has to go through a v-edge in either G'_1 or in G'_2 so the flow along any path is limited by the capacity of a v-edge which is 1. One h-edge can help bypass at most one node.

To prove the claim, we prove that (i) any subset $U \subseteq V$ that touches every path of G in two nodes provides a cut of G'' of size at most $|U|$ and (ii) any cut C of G'' provides a subset $U \subseteq V$ of size $|C|$ which touches every path in two nodes.

We start with (i). Fix a minimum subset $U \subseteq V$ that touches every $s-t$ path in G in at least two places. We would like to pick v-edges associated with these vertices. But, for every vertex v, we have two v-edges in G'', one in G'_1 and one in G'_2. Which one to pick? Intuitively, we want to pick edges from G'_1 when the node from U is the first to touch a given path, and from G'_2 for other nodes. To this end, consider the subset $U_1 \subseteq U$ of nodes which are the first node on

some path in G and the subset $U_2 \subseteq U$ of nodes that are the last node of some path. We claim that $U_1 \cap U_2 = \emptyset$ as, otherwise, there would be a path touched by at most one node of U. Define the set C of v-edges as follows. For a node in U_1 (U_2), place the corresponding v-edge from G'_1 (G'_2) in C. Clearly, the size of C is at most the size of U. We claim that C is a cut of G''. Assume that there is a path p'' in G'' that does not go through edges in C and consider its associated path p in G. Let u be the node associated with the unique h-edge in the path. Clearly, u cannot be both the first and last node from U in p. Assume, wlog. that u is not the last node from U that touches p. Then, the last node of p is in U_2 and, therefore, p'' crosses the corresponding v-edge in C.

Now we prove (ii). This is a bit simpler. Fix a minimum cut C of G''. Let U be the set of nodes associated with v-edges from the cut. Note that the same node of G can be associated with two v-edges. This does not affect the argument. We claim that the set U touches every $s - t$ path in G in at least two nodes. To see this, assume, for contradiction, that there is an $s - t$ path in G which is touched by U at most once. We show that this path is associated with a path in G'' that does not cross C, a contradiction. Indeed, let u be the unique node of the path touched by U. Consider the path that goes from s to u_{in} in G'_1, takes the h-edge to jump to $u_{out} \in G'_2$, and then follows a path to t. By construction, given that the path does not intersect with U (other than in u), this path does not go through any v-edges in C.

Common mistakes: Some students started from a minimum cut of G (where edges have capacity 1) and argued that one can choose both endpoints of all edges in the cut. Obvioulsy, there is no guarantee that this approach will yield an optimal answer.

Other students also started with a minimum cut of G. This time, they increased the capacity of these edges in the cut and looked for a second cut. Again, there is no guarantee that this approach yields an optimal solution.

Problem 6-2. A little LP [10 points]

Show that

$$\max 3x + 4y + 5z$$

$$\begin{aligned} x + y &\leq 2 \\ y + z &\leq 3 \\ x + z &\leq 4 \\ x, y, z &\geq 0 \end{aligned}$$

has a value of no more than 20.

Solution: The plan of attack is to write the dual, find a solution with value 20, and invoke weak duality. This will certainly prove that the optimal value of the LP is ≤ 20.

Luckily, the LP is already in standard form. The dual is:

$$\min 2a + 3b + 4c$$

$$\begin{aligned} a + c &\geq 3 \\ a + b &\geq 4 \\ b + c &\geq 5 \\ a, b, c &\geq 0 \end{aligned}$$

To find a good solution, one idea is to make a and b as small as possible, given that they contribute the least to the objective function. The solution $a = 3$, $b = 5$, $c = 0$ is feasible but has a value of 21.

Let's try to make $c = 1$. We must have $a = 2$ and $b = 4$ to get a feasible solution. Bingo! this gives us the desired value of 20.

Alternative Solution: You can also work directly with the primal. You can group the objective function to use the constraints:

$$\begin{aligned} 3x + 4y + 5z &\leq 3x + 6y + 5z \\ &= 2(x + y) + 4(y + z) + 1(x + z) \\ &\leq 2 \cdot 2 + 4 \cdot 3 + 4 \\ &= 20 \end{aligned}$$

As you can see, the coefficients on each grouping match with the variables in the dual. Finding the groupings can be generalized to multiplying each of the constraints by variable, and constructing the dual.

Problem 6-3. Routing Tours [40 points]

Tourists spawn in Kendall Square, represented as a node s, at a fixed rate of p. We would like to send them along MIT's network of hallways, which we model as a connected directed graph $G = (V, E)$ with no duplicate edges, to the student center, represented as a node t. Each hallway $(i, j) \in E$ has a maximum rate of tourists it can sustain, u_{ij}, as well as a disturbance ratio, c_{ij}, such that the disturbance caused by sending tourists at a rate of k down this hallway is $k \cdot c_{ij}$. Our goal is to distribute the tourists–that is, to pick a rate of tourists for each hallway–such as to minimize the total (additive) disturbance across all hallways. That is, if we let k_{ij} denote the rate of tourists on hallway (i, j), we want to minimize $\sum_{(i,j)\in E} k_{ij} \cdot c_{ij}$.

(a) [10 points] Formulate this problem as an LP (not necessarily in standard form).

Solution: Denote by x_{ij} the rate of tourists sent down hallway $e = (i, j)$. (Default to 0 if the hallway does not exist.) We want to minimize the total cost, subject to the capacity constraints and flow conservation.

$$\begin{aligned}
\min \quad & \sum_{i,j} x_{ij} c_{ij} \\
\text{subject to} \quad & x_{ij} \le u_{ij} && \forall e = (i,j) \in E \\
& \sum_{j \neq i} [x_{ij} - x_{ji}] = 0 && \forall i \in V \setminus \{s, t\} \\
& \sum_{j \neq s} [x_{sj} - x_{js}] = p \\
& \sum_{j \neq t} [x_{jt} - x_{tj}] = p \\
& x_{ij} \ge 0
\end{aligned}$$

Note that the flow conservation constraints are not independent. That is, one could remove either the source or sink flow constraint with no effect on the feasible region.

(b) [10 points] Consider a Linear Program P with an equality constraint $\mathbf{a}^T \cdot \mathbf{x} = b$. Describe a how you would deal with this equality constraint when writing the dual to P. In particular, show that such an equality constraint maps to a variable in the dual that does not have to be non-negative.

Solution: First, write the constraint in standard form using a pair of inequalities:

$$\mathbf{a}^T \cdot \mathbf{x} \geq b$$
$$-\mathbf{a}^T \cdot \mathbf{x} \geq -b$$

Let the corresponding variables in the dual program be d^+, d^-. The objective function of the dual will contain terms of the form bd^+ and $-bd^-$, where $d^+, d^- \geq 0$. Letting $d = d^+ - d^-$, the objective function will then contain a term of the form bd, where d can be any real.

Now consider the constraint in the dual corresponding to the variable x_j. The expression in terms of the dual variables will have a term of the form $a_j d^+$ and a term of the form $-a_j d^-$, and again by letting $d = d^+ - d^-$ the dual constraint will have a term of the form $a_j d$.

Therefore equality constraints in the primal map to dual variables which don't have to be positive or negative.

(c) [10 points] Using the recipe from part **(b)**, write the dual of the program you gave in part **(a)**.

Solution: We can apply our recipe from the previous part. We organize the different constraints into sections and give them appropriate names. Let the first $|E|$ dual variables be w_e, the next $|V| - 2$ be d_i, and the last two be $-d_s, d_t$. Since equality constraints in the primal map to unconstrained variables in the dual, we can express all the d_i variables in the dual without nonnegativity constraints. Note that the primal is not currently in standard form; the primal objective in standard form would actually be $\max \sum_{i,j} -c_{ij}x_{ij}$. This gives us the following for the dual:

$$\begin{aligned} \min \quad & \sum_e w_e u(e) + pd_s - pd_t & \\ \text{subject to} \quad & w_e + d_i - d_j \geq -c_{ij} & \forall e = (i,j) \in E \\ & w_e \geq 0 & \end{aligned}$$

which can then be re-written as:

$$\begin{aligned} \max \quad & pd_t - pd_s - \sum_e w_e u(e) & \\ \text{subject to} \quad & -w_e + d_j - d_i \leq c_{ij} & \forall e = (i,j) \in E \\ & w_e \geq 0 & \end{aligned}$$

(d) [10 points] Restrict your attention to the case where $p = 1$ and $u_{ij} = \infty$ for each edge (i, j). To what common problem X does the tourist routing problem reduce to? Explain why your LP formulation from either part a) or part b) corresponds to X.

Solution: With $u(e) = \infty$, we no longer have the variables w_e. This is because the penalty of assigning a value greater than 0 to one of them is overwhelming.

$$\begin{aligned} &\max \ d_t - d_s \\ &\text{subject to } \ d_j - d_i \leq c_{ij} \qquad\qquad \forall e = (i,j) \in E \end{aligned}$$

This is the shortest path problem from s to t, with c_{ij}'s representing the length of edges. d_v represents the shortest distance from s to a vertex v.

It is not surprising, because if we just want to send 1 unit of flow, we should send it along the shortest/minimum cost path.

Problem 6-4. Feedback Form [10 points] Please fill out a feedback form about this problem set at

`https://forms.gle/fXDMDjt5Ka1FCCGV7`.

It should not take more than a few minutes and will greatly help us improve teaching and material for future semesters!