CS 561, HW7

Stephen Harding

December 6, 2012

1. **Exercise 26.2-4**

   Solution:

   The minimum cut is: $(\{s, v_1, v_2, v_4\}, \{v_3, t\})$ with the value 23.

   The third augmenting path (c) is the one that cancels flow. (On the $v_3 \to v_2$ edge.)

2. **Exercise 26.2-11**

   Solution:

   Our task is to determine the edge connectivity of an undirected graph $G = (V, E)$ by running a max-flow algorithm on at most $|V|$ flow networks where each network has $O(V)$ vertices and $O(E)$ edges.

   Begin by converting the undirected graph to a directed graph where for every edge in G, we replace it with directed edges (one in each direction) each of which has weight = 1. The directed version of G has $2 \cdot |E| = O(E)$ edges and $|V|$ vertices. The connectivity of the graph is defined as the minimum number of edges that can be removed to disconnect the graph. We find the minimum set of edges to remove by finding the minimum cut of the graph for some source and sink in the graph. To find the min cut that corresponds to the edge connectivity, we need to find the minimum of all the min-cuts that are possible for various sources and sinks. Since we construct the directed graph by converting each undirected edge into two opposing directed edges, we can fix an arbitrary source and consider the $|V| - 1$ flows that can be computed by selecting a sink from one of the remaining vertices.

   Procedure: select a source $s \in V$. For each $v \in V - \{s\}$ set v as the sink t and compute the max-flow (Note that by the Max-Flow/Min-Cut theorem, this is also the min cut). From the computed max flows, the minimum value is the edge connectivity. $\square$

3. **A bipartite graph is a graph that contains no cycle with an odd number of edges. Recall from the wrestler problem that a graph, $G = (V, E)$ is bipartite iff V can be partitioned into two sets L, R such that all edges in E have one endpoint in L and one endpoint in R. A *matching* in a graph is a set of edges $E' \subseteq E$ such that each vertex in V is matched at most once, i.e. it is incident to a most one edge in E'. A *perfect matching* is a matching where every vertex is matched, i.e. is incident to exactly one edge in E'. For a set of vertices $S \subseteq V$, let $N(S)$ be the set of all neighbors of S, i.e. $\{y \in V : (x, y) \in E$ for some $x \in S\}$**

   Assume we are given a bipartite graph where $|L| = |R|$. Prove that there is a perfect matching in G iff $|N(S)| \geq |S|$ for all $S \subseteq L$.

   Solution:

   We must show both side of the conditional. i.e. we must show both:

   (a) *If there is a perfect matching in G, then $|N(S)| \geq |S|$, $\forall\ S \subseteq L$.*

(b) *If* $|N(s)| \geq |S|$, $\forall S \subseteq L$, *then there is a perfect matching in* G.

(a): This is true because we know that G is a bipartite graph and also that there is a perfect matching in G. This means that every vertex in L, there is *at least* one edge that connects it to a unique vertex in R. (Note that there may be other edges that connect it to other vertices in R as well.) Thus, $|N(S)| \geq |S|$ for any $S \subseteq L$. $\surd$

(b): To show this, we rely on **Corollary 26.11** in out textbook which states *"The cardinality of a maximum matching M in a bipartite graph G equals the value of a maximum flow f in its corresponding flow network G'"*

Thus, we begin by constructing a new graph $G' = (V', E')$ where $V' = V \cup \{s, t\}$. E' is constructed by converting all of the undirected edges in E to directed edges where the edges are directed from nodes in L to nodes in R. Finally, we also add directed edges from s to each node in L as well as edges from each node in R to t. All edges have capacity = 1. By Corollary 26.11, the number of matches is equal to the max flow = min-cut of G'. It is sufficient to show that $|(S, T)| = |L| = |R|$ if (S, T) is the min cut.

To see that this is the case, note that we already know that for every subset of L, the number of neighbors of the subset (all of which must be in R since this is a bipartite graph) is greater than the size of the subset. Since every edge's weight is 1, the min-cut will never be less that $|L|$. Furthermore, the min-cut also cannot be greater than $|L|$ since there are $|L|$ edges coming out of s (i.e. the max flow cannot be greater than $|L|$).

Therefore, the min-cut = Max-Flow = $|L|$. Thus the number of matches = $|L| = |R|$ and so there is perfect matching. $\square$

4. **Consider the following problem related to segmenting the pixels of an image between foreground and background. We have a picture that consists of n pixels. We represent this as an undirected graph $G = (V, E)$ where V is the set of pixels and there is an edge $(i, j) \in E$ iff pixel i and pixel j are neighbors in the image. We want to find a good segmentation, which is an assignment of each pixel to either the foreground or the background.**

   For each pixel i, we have a likelihood a_i that i belongs to the foreground and a likelihood b_i that i belongs to the background. These likelihood values are all non-negative. Additionally, for each edge $(i, j) \in E$, we have a non-negative separation penalty $p_{i,j}$ which is charged if one of i or j is assigned to the foreground and the other is assigned to the background.

   Our problem then is to find a partition of the set of pixels into sets A and B so as to maximize:

$$L(A, B) = \sum_{i \in A} a_i + \sum_{i \in B} b_i - \sum_{(i,j) \in E, |A \cap \{i,j\}| = 1} p_{i,j}$$

   Give an efficient algorithm to solve this problem.

   Solution:

   Begin by constructing a graph as follows:

   First, add a node s and connect it with directed edges pointing away from s to each of the pixel nodes i where the weight is a_i. Next, add a node t and connect it with directed edges pointing towards t with wight b_i to each of the pixels i. Pixel nodes i and j are connected with directed edges going in both directions with weight $p_{i,j}$.

Finally, compute the MIN-CUT of the graph where s is the source and t is the sink. The cut forms the partition we are trying to find. The s side of the cut represents the foreground pixels and the t side the background pixels. Note also that this $L(A, B)$ cut does in fact maximize the given sum because the cut will take the lightest possible cut that forms a partition. The edges that form the cut may either be one of the a_i's, b_i's, or the $p_{i,j}$'s. If it is a $p_{i,j}$ then min-cut will automatically select the best edges in order to minimize that part of the sum. Furthermore, min-cut will also automatically form the partition such that the highest values for the a_i's will fall on the A side and the b_i's on the B side.

5. **Exercise 29.2-2 (Linear Program for example in Figure 24.2(a))**

   Solution:

   Maximize d_y
   Subject to:

   $d_s = 0$
   $d_t \leq 3$
   $d_t \leq d_y + 1$
   $d_x \leq d_t + 6$
   $d_x \leq d_y + 4$
   $d_x \leq d_z + 7$
   $d_z \leq d_x + 2$
   $d_z \leq d_y + 6$
   $d_y \leq 5$
   $d_y \leq d_t + 2$

6. **Exercise 29.2-4 (Network Flow as an LP)**

   Solution:

   Maximize $\sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs}$

   Subject to:

   $f_{sv_1} \leq 16$
   $f_{sv_2} \leq 13$
   $f_{v_1 v_3} \leq 12$
   $f_{v_3 v_2} \leq 9$
   $f_{v_3 t} \leq 20$
   $f_{v_2 v_1} \leq 4$
   $f_{v_2 v_4} \leq 14$
   $f_{v_4 v_3} \leq 7$
   $f_{v_4 t} \leq 4$

7. **Rock, Paper, Scissors is a simple 2 person game. In a given round, both players simultaneously choose either Rock, Paper or Scissors. If they both choose the same object, it's a tie. Otherwise, Rock beats Scissors; Scissors beats Paper; and Paper beats Rock. Imagine you're playing the following betting variant of this game with a friend. When Scissors beats Paper, or Paper beats Rock, the loser gives the winner \$1. However, in the case when Rock beats Scissors, this is called a SMASH, and the loser must give the winner \$10.**

(a) **Say you know that your friend will choose Rock, Scissors or Paper, each with probability 1/3. Write a linear program to calculate the probabilities you should use of choosing each object in order to maximize your expected winnings. Let p_1, p_2, p_3 be variables associated with the best way of choosing Rock, Scissors and Paper respectively.**

Solution:

The payoff matrix is given as:

	R	S	P
R	0	10	-1
S	-10	0	1
P	1	-1	0

Thus, the solution is to maximize the following expression (Note: since the values on the diagonal are all zero, we don't need to include them in the expression):

$$p_1 \cdot 10 \cdot \frac{1}{3} + p_2 \cdot 1 \cdot \frac{1}{3} + p_3 \cdot 1 \cdot \frac{1}{3} - p_1 \cdot 1 \cdot \frac{1}{3} - p_2 \cdot 10 \cdot \frac{1}{3} - p_3 \cdot 1 \cdot \frac{1}{3} = \frac{1}{3}(p_1 \cdot 10 + p_2 + p_3 - p_1 - p_2 \cdot 10 - p_3)$$

Subject to:

$0 \leq p_1 \leq 1$

$0 \leq p_2 \leq 1$

$0 \leq p_3 \leq 1$

$p_1 + p_2 + p_3 = 1$ □

(b) **Now say that your friend is smart and, also, clairvoyant: she will magically know the exact probabilities you are using and will respond optimally. Write another linear program to calculate the probabilities you should now use in order to maximize your expected winnings.**

Solution:

Using the given hint, we need to maximize Q where Q is constrained by the three possible options for the opponent. (i.e. they always pick rock, or always pick paper, or always pick scissors.)

In addition to the constraints given in (a), we maximize Q where Q is subject to:

$Q \leq p_3 - 10 \cdot p_2$ – this is where she always chooses rock

$Q \leq 10 \cdot p_1 - p_3$ – this is where she always chooses scissors

$Q \leq p_2 - p_1$ – this is where she always chooses paper

8. **The problem INDEPENDENT-SET asks: "Does there exist a set of k vertices in a graph G with no edges between them?" Show that this problem is NP-Complete. (hint: Reduce from CLIQUE)**

Solution:

To show that INDEPENDENT-SET is NP-Complete, we first show that an NP-Hard problem (we use CLIQUE in this case) can be reduced to INDEPENDENT-SET. Then we show that it is also in NP by showing that a 'yes' answer can be verified in polynomial time.

(a) **CLIQUE can be reduced to INDEPENDENT-SET:** This is easy to show. First, note that the nodes that form a clique in a graph will also form an independent set in the graph's compliment (a new graph in which we place edges between nodes if and only if there is not an edge between those nodes in the original graph). This is because the clique nodes are fully connected in the original graph and are therefore disconnected in the compliment graph.

Thus, we can decide CLIQUE for some k by computing INDEPENDENT-SET for that same k on the compliment of the graph. Computing the compliment graph $G' = (V', E')$ of $G = (V, E)$

can be done in polynomial time in $|V|$ by examining each pair $u, v \in V$ and if there is no edge $(u, v) \in E$ we add it to E'. This can be done in $O(\binom{|v|}{2}) = O(|V|^2)$ time.

Hence, we can reduce CLIQUE to INDEPENDENT-SET in polynomial time. Therefore, INDEPENDENT-SET is NP-HARD.

(b) **A yes answer can be verified in polynomial time:** This is trivial. Simply examine each edge (u, v) that is incident to the set of vertices returned by INDEPENDENT-SET where u is in the set. We only need to verify that v is not in the set. Placing the vertices in a hash-table will allow us to perform each check in amortized constant time. Thus if n is the highest number of edges incident to a node, then we can do the verification in $O(nk) = O(k)$ time. $\square$