Problem 1, CS38 Set 5, Matt Lim

(a) Note that for $1(a)$ and $1(b)$, we will let the flow and capacity be 0 for each pair of vertices u, v with $(u, v) \notin E$. Also note that f_{uv} denotes the flow going out from u to v and f_{vu} denotes the flow going into u from v. We will format the vertex-limited flow problem as a linear program in the following way. Our original input will consist of the following.

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

such that

$$f_{uv} \leq c(u, v) \text{ for all } u, v \in V$$

$$\sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv} \text{ for each } u \in V - \{s, t\}$$

$$\sum_{v \in V} f_{vu} + \sum_{v \in V} f_{uv} \leq g(v) \text{ for each } u \in V$$

$$f_{uv} \geq 0 \text{ for each } u, v \in V$$

Turning this into standard form then gives us the following.

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

such that

$$f_{uv} + S_C = c(u, v) \text{ for all } u, v \in V$$

$$\sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv} \text{ for each } u \in V - \{s, t\}$$

$$\sum_{v \in V} f_{vu} + \sum_{v \in V} f_{uv} + S_G = g(v) \text{ for each } u \in V$$

$$f_{uv} \geq 0 \text{ for each } u, v \in V$$

$$S_C, S_G \geq 0$$

Now for a short explanation. The first two constraints ensure that the flow satisfies the capacity constraint and flow conservation. The next constraint satisfies the constraint given in the problem, and the last two simply ensure that certain values are non-negative. Basically, all the constraints except for the third one ensure we get a valid flow, and the third one gives us what the problem wants; that is, that the flow entering any vertex v plus the flow exiting v doesn't exceed $g(v)$. Then, with all these constraints, we want to maximize the flow from s to t.

(b) Here is how we will format this problem as a linear program. Our original input will consist of the following.

$$\textbf{maximize} \sum_{e \text{ exiting } s} 2 \cdot f(e) - f(e^*)$$

such that

$$f_{uv} \leq c(u, v) \text{ for all } u, v \in V$$

$$\sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv} \text{ for each } u \in V - \{s, t\}$$

$$f_{uv} \geq 0 \text{ for each } u, v \in V$$

Turning this into standard form then gives us the following.

$$\textbf{maximize} \sum_{e \text{ exiting } s} 2 \cdot f(e) - f(e^*)$$

such that

$$f_{uv} + S_C = c(u, v) \text{ for all } u, v \in V$$

$$\sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv} \text{ for each } u \in V - \{s, t\}$$

$$f_{uv} \geq 0 \text{ for each } u, v \in V$$

$$S_C \geq 0$$

Now for a short explanation. The constraints ensure that the flow satisfies the capacity constraint and flow conservation, and that all flows are positive. In other words they ensure we get a valid flow. Now we must show that maximizing what we maximize gives us a maximum flow with the minimum flow across e^*. That is, we want to show that maximizing our function minimizes $f(e^*)$ while still giving us max flow. We will first show that maximizing our function gives us a max flow. Let the value of the flow we get by maximizing our function be f, and let the value of the max flow be f_{max}. $f > f_{max}$ clearly cannot happen because then f_{max} wouldn't be the max flow. Now we will show that $f < f_{max}$ cannot happen. So, assume to the contrary that $f < f_{max}$. Let $FUNC = \sum_{e \text{ exiting } s} 2 \cdot f(e) - f(e^*) = 2f - B$. Now consider $FUNC$ for a flow of value f_{max}. Let $f_{max} - f = k$. This means we can augment our flow by some value $k > 0$. Then we have that the new value of our function is $FUNC = 2(f + k) - (B + c)$, c a constant. If the augmented flow goes through e^*, then we have that $FUNC \geq 2(f + k) - (B + k) = 2f - B + k > 2f - B$. But this is a contradiction since we supposedly maximized $FUNC$. Now, if the augmented flow doesn't go through e^*, then the new value of our function is $FUNC = 2(f + k) - B > 2f - B$. So we again obtain a contradiction. Thus we have that $f = f_{max}$. Now we will show that maximizing our function minimizes $f(e^*)$. This is easy; we already have shown that maximizing our function gives us the max flow, so we have that the summation part of our maximized function is just $2 \cdot f_{max}$. So then to maximize the function, we simply must choose the minimum $f(e^*)$ value for all max flows, since we subtract it away from the summation. This clearly gives us the max flow with the smallest flow across e^*, as desired. So overall, we know maximizing our function gives us a max flow, and since we subtract $f(e^*)$ we know it gives us a max flow with the minimum flow across e^*. Thus we can conclude that our linear program will give us the maximum flow with the smallest flow across e^*.

Problem 2, CS38 Set 5, Matt Lim

(a) We will first formulate the Assignment Problem as a problem of finding the maximum weight perfect matching in a bipartite graph with every vertex on one side connected to every vertex on the other side, where $G = (L \cup R = V, E)$. We will assume that $|L| = |R| = n$. Let i index L and j index R, with $v_i \in L$ and $v'_j \in R$. For all $e \in E$, let $X_e = 1$ if e is selected, and $X_e = 0$ otherwise. Let W_e be the weight of the edge e. We will format this problem as a linear program as follows.

$$\textbf{maximize} \sum_{e \in E} X_e W_e$$

$$\textbf{such that}$$

$$\sum_{e \text{ including } v} X_e = 1 \text{ for all } v \in V$$

$$0 \leq X_e \leq 1 \text{ for all } e \in E$$

Basically what we are doing is maximizing the total weight of all the edges we pick. The first constraint says that each vertex has exactly one edge attached to it selected, which means that the edges we select form a perfect matching. This is akin to saying each "agent" is assigned to exactly one "task." So overall, we are maximizing the weight of the assignment.

Now we will show that the constraint matrix is totally unimodular. So, let us consider the constraint matrix A. Our constraint matrix will have $2n$ rows and n^2 columns. The first n rows will consist of $v_1, v_2, \ldots, v_n$ (the vertices in L) and the next n rows will consist of $v'_1, v'_2, \ldots, v'_n$ (the vertices in R). The columns will represent every possible edge, ordered $(v_1, v'_1), (v_1, v'_2), \ldots, (v_2, v'_1), (v_2, v'_2), \ldots, (v_n, v'_n)$. A cell will have the value of one if the vertex represented by its row is in the edge represented by its column, and zero if it is not in that edge. We have that each column has at exactly two ones in it, the two ones being the vertices in each edge. We also have that swapping rows does not affect if A is unimodular. This is because it is a linear algebra theorem that says that swapping the rows of a square matrix only changes the sign of the determinant. It then clearly follows that swapping rows doesn't affect unimodularity. Note that we use this matrix for the following reasons. We have that $Ax = b$ should give us our constraints. So given A, we can let x be an n^2 column vector that tells us if each edge is selected or not, and b be a $2n$ column vector that is all 1s, since each vertex is used. Then we can see that $Ax = b$ gives us our constraints. So we have explained why we use this constraint matrix A.

Now we will prove by induction that A is unimodular. To do this we will prove that every square submatrix of A has determinant 0, 1, or -1. First we will consider the base case. So, consider any submatrix of A that has dimensions 1×1. We showed above that every cell of A is either zero or one. And since the determinant of a 1×1 matrix is just the value of the cell, we have that every 1×1 submatrix of A has determinant zero or one. Now we will do our inductive assumption. So, assume that every $k \times k$ submatrix of A, where $1 \leq k \leq i - 1$ has determinant 0, 1, or -1. Now we must show that every $i \times i$ submatrix of A has such a determinant. So consider an arbitrary $i \times i$ submatrix B of A. Now we will consider three cases. The first case is when a column c in B is made of all zeroes. Then we have that the determinant of B is zero (expand by cofactors along c). The second case is when a column c in B is made of all zeroes and one one. Here we have that the determinant of B is either 0, 1, or -1. We can see this is true if we expand by cofactors along c. This is because by calculating the determinant this way, it is the same as if we calculate the determinant of the $(i-1) \times (i-1)$ matrix we get by crossing out column c and the row with the one one in column c and multiplying it by 1 or -1. And by our inductive assumption, the determinant of such a $(i-1) \times (i-1)$ matrix is 0, 1, or -1. So clearly the determinant of B is 0, 1, or -1. The third and final case is when every column of B is has two ones. Here we will do the following. We will reorder the rows of B so that they go $v_i, v'_j, v_m, v'_n, \ldots, v_k, v'_l$, etc. In other words, we alternate rows going by which side of the bipartite graph they are on. We can do this because having two ones in each column tells us that the number of rows from one side and the other side are equal in our submatrix, since otherwise there won't be enough combinations of rows/vertices to make unique edges from one side to another that result in fitting two ones in each column. Observe that this means that in each column, the ones in that column are in different parity rows. This is because we formulated this problem as a bipartite graph, and there are no edges that connect

vertices on the same side. Note that switching the rows like this doesn't change the unimodularity, as we showed above. So now, we will multiply this switched row matrix by a vector z. This vector z will contain alternating ones and negative ones. For example, $z = (-1, 1, -1, 1)$. Then we have that the matrix multiplication of $z \cdot B = 0$. This is because when we multiply each column of B by z, we will get one one multiplied by 1, and one one multiplied by -1, which adds up to 0 (since all the other cells in each column that aren't one are zero). From linear algebra, the determinant of B is zero because we multiplied it by a nonzero vector z and got the result of the 0 vector, which means B is singular and thus that its determinant is zero. So we have that all the cases are covered. So we can conclude by induction that every square submatrix of A has determinant 0, 1, or -1, and thus that A is totally unimodular.

(b) Let us represent the problem as a bipartite graph $G = (L \cup R = V, E)$ where every vertex on one side is connected to every vertex on the other side, and where L represents the lots and R represents the showrooms. So $|L| = m$ and $|R| = n$. Let i index L and j index R, with $v_i \in L$ and $v'_j \in R$. Let $X_{ij} = 1$ if we ship along edge (v_i, v'_j) and $X_{ij} = 0$ if we don't ship along that edge. Then let Q_{ij} be how many units we ship along edge (v_i, v'_j) (so if $X_{ij} = 0$ then for that same i, j, $Q_{ij} = 0$). Let C_{ij} be the cost of shipping one unit on the edge (v_i, v'_j). We will format this problem as a linear program as follows.

$$\textbf{maximize } -\sum_{\forall i,j} X_{ij} Q_{ij} C_{ij}$$

$$\textbf{such that}$$

$$\sum_j Q_{ij} \leq a_i \text{ for all } i$$

$$\sum_i Q_{ij} \geq b_j \text{ for all } j$$

$$0 \leq X_{ij} \leq 1 \text{ for all } i, j$$

In standard form, this is

$$\textbf{maximize } -\sum_{\forall i,j} X_{ij} Q_{ij} C_{ij}$$

$$\textbf{such that}$$

$$\sum_j Q_{ij} + S_A = a_i \text{ for all } i$$

$$\sum_i Q_{ij} - S_B = b_j \text{ for all } j$$

$$0 \leq X_{ij} \leq 1 \text{ for all } i, j$$

$$S_A, S_B \geq 0$$

This works because we are maximizing the negative of the total cost, which minimizes the total cost. And we do this under the constraints that we never exceed the supply or undercut the demand. So solving this linear program gives us an optimal shipment scheme subject to the supply and demand constraints.

Now we will show that the constraint matrix is totally unimodular. So, let us consider the constraint matrix A. Our constraint matrix will have $m + n$ rows and mn columns. The first m rows will consist of $v_1, v_2, \ldots, v_m$ (the vertices in L) and the next n rows will consist of $v'_1, v'_2, \ldots, v'_n$ (the vertices in R). The columns will represent every possible edge, ordered
$(v_1, v'_1), (v_1, v'_2), \ldots, (v_2, v'_1), (v_2, v'_2), \ldots, (v_m, v'_n)$. A cell will have the value of one if the vertex represented by its row is in the edge represented by its column, and zero if it is not in that edge. We have that each column has at exactly two ones in it, the two ones being the vertices in each edge. We also have that swapping rows does not affect if A is unimodular. This is because it is a linear algebra theorem that says that swapping the rows of a square matrix only changes the sign of the determinant. It then clearly follows that swapping rows doesn't affect unimodularity. Note that we use this matrix for the following reasons. We have that $Ax = b$ should give us our constraints. So given A, we can let x be an mn column vector that tells us how much is shipped

over each edge, and b be a $m+n$ column vector where the top part (m cells) is $a_i - S_A$ (unique slack variable for each supply) and the bottom part (n cells) is $b_j + S_B$ (unique slack variable for each demand). Then we can see that $Ax = b$ gives us our constraints. So we have explained why we use this constraint matrix A.

Now we will prove by induction that A is unimodular. To do this we will prove that every square submatrix of A has determinant 0, 1, or -1. First we will consider the base case. So, consider any submatrix of A that has dimensions 1×1. We showed above that every cell of A is either zero or one. And since the determinant of a 1×1 matrix is just the value of the cell, we have that every 1×1 submatrix of A has determinant zero or one. Now we will do our inductive assumption. So, assume that every $k \times k$ submatrix of A, where $1 \leq k \leq i-1$ has determinant 0, 1, or -1. Now we must show that every $i \times i$ submatrix of A has such a determinant. So consider an arbitrary $i \times i$ submatrix B of A. Now we will consider three cases. The first case is when a column c in B is made of all zeroes. Then we have that the determinant of B is zero (expand by cofactors along c). The second case is when a column c in B is made of all zeroes and one one. Here we have that the determinant of B is either 0, 1, or -1. We can see this is true if we expand by cofactors along c. This is because by calculating the determinant this way, it is the same as if we calculate the determinant of the $(i-1) \times (i-1)$ matrix we get by crossing out column c and the row with the one one in column c and multiplying it by 1 or -1. And by our inductive assumption, the determinant of such a $(i-1) \times (i-1)$ matrix is 0, 1, or -1. So clearly the determinant of B is 0, 1, or -1. The third and final case is when every column of B is has two ones. Here we will do the following. We will construct a row vector z as follows. Going down the rows, if the row represents a vertex from the left side, we will add a one to z. If the row represents a vertex from the right side, we will add a negative one to z. So if the rows are in the order l, l, l, l, r, where l is a vertex from the left side and r is a vertex from the right side, then $z = (1, 1, 1, 1, -1)$. Then we have that the matrix multiplication $z \cdot B = 0$. This is because when we multiply each column of B by z, we will get one one multiplied by 1, and one one multiplied by -1, which adds up to 0, since each column will have a 1 in a row representing a left side vertex and a 1 in a row representing a right side vertex. From linear algebra, the determinant of B is zero because we multiplied it by a nonzero vector z and got the result of the 0 vector, which means B is singular and thus that its determinant is zero. So we have that all the cases are covered. So we can conclude by induction that every square submatrix of A has determinant 0, 1, or -1, and thus that A is totally unimodular.

Problem 3, CS38 Set 5, Matt Lim

(a) We will format this problem as a linear program in the following way. Our original input will consist of the following. Note that we are not maximizing anything, just finding if there is a feasible solution. Also note that P_i is the expected payoff for the row player choosing row i and Q_j is the expected payoff for the column player choosing column j, and P_{max} and Q_{max} are the max payoffs for each player.

Find a feasible solution subject to the following contraints:

$$P_i = \sum_j q_j M[i,j] = P_{max} \text{ for } i \in S$$

$$Q_j = -\sum_i p_i M[i,j] = Q_{max} \text{ for } j \in T$$

$$P_i = \sum_j q_j M[i,j] \leq P_{max} \text{ for } i \notin S$$

$$Q_j = -\sum_i p_i M[i,j] \leq Q_{max} \text{ for } j \notin T$$

$$\sum_i p_i = 1 \text{ and } \sum_j q_j = 1$$

We have that the first two constraints must be true for a Nash equilibrium for the following reasons. If we have a Nash equilibrium, no player has any incentive to unilaterally deviate from their current mixed strategy. Thus, all the payoffs for all the moves given by S and T must be equal and maximum if S and T give us a Nash equilibrium, since otherwise there would be an incentive to deviate. This is because if they were not all equal, you would have an incentive to not do certain moves and to do other moves instead. And they must all be maximum because otherwise there would clearly be incentive to deviate by making some better move. The second two constraints must be true for a Nash equilibrium for the following reasons. If we have a Nash equilibrium, no player has any incentive to unilaterally deviate from their current mixed strategy. Thus, all the payoffs for all the moves not given by S and T must be less than or equal to the max payoffs if S and T give us a Nash equilibrium, since otherwise there would be an incentive to deviate by taking a better move. Then the last constraint just makes sure that the probabilities sum to one. So then we can see that if we can find a feasible solution for our constraints, we have a Nash equilibrium given by S and T, and that if we cannot find a feasible solution for our constraints, we do not have a Nash equilibrium given by S and T. This is because as we showed above, if the constraints are met then there are no incentives to deviate, and if they are not there are. Note that we are using linear programming so we can say that our solution is polynomial.

(b) We have that, given n possible moves to choose from for the row player and column player, there are 2^n ways to choose a subset S of those moves and 2^n ways to choose a subset T of those moves (since given a set of size n there are 2^n possible subsets), where S and T are described as in part (a). Then we have that there are $2^n \times 2^n$ ways to choose combinations of S and T. So we can just run part (a) on each of these combinations and see if any of them give us a Nash equilibrium. This is clearly $2^{O(2n)} \cdot \text{poly}(n)$, since (a) runs in polynomial time and there are 2^{2n} possible ways to choose S and T.