# Problem 1

$$\begin{aligned} w = \min \sum c_{ij} f_{ij} \\ \sum_j f_{ij} - f_{ji} = 0 \ (\forall i) \\ \sum f_{ij} = 1 \\ f_{ij} \geq 0 \end{aligned}$$

## Part (a)

I explain why the above linear program captures the minimum mean cycle problem.

$f_{ij}$ is a circulation, so it can be decomposed into cycles. For each cycle $k$, let $n_k$ be the number of edges involved in the cycle, $\lambda_k$ be the weight of that cycle in the decomposition, so that the contribution of the cycle to the circulation $f_{ij}$ is $\lambda_k c_{ij}^k$, where $c_{ij}^k = \frac{1}{n_k}$ if the cycle $k$ goes from edge $i$ to $j$ and $c_{ij}^k = 0$ otherwise. In this way, $f_{ij} = \sum_k \lambda_k c_{ij}^k$ with $\sum_k \lambda_k = 1$. Note that $\sum_{ij} c_{ij}^k = n_k \frac{1}{n_k} = 1 \forall k$, so that $\sum_{ij} f_{ij} = \sum_{ij} \sum_k \lambda_k c_{ij}^k = \sum_k \lambda_k \sum_{ij} c_{ij}^k = \sum_k \lambda_k = 1$.

I claim that all the cycles of $f_{ij}$ are minimum mean cycles when $f_{ij}$ is optimal. Indeed, if one of the cycles, say cycle $k$, was not a minimum mean cycle, then I could improve the optimum by setting $\lambda_k = 0$ and increasing the weights of the other (smaller) mean cycles accordingly.

## Part (b)

The dual of this linear program is:

$$\begin{aligned} w = \max \lambda \\ \lambda + y_i - y_j \leq c_{ij} \end{aligned}$$

If I set $p_i = -y_i$, I can transform this linear program into a form where costs $(c_{ij} - \lambda)$ and reduced costs $((c_{ij} - \lambda) + p_i - p_j)$ appear explicitly:

$$\begin{aligned} w = \max \lambda \\ (c_{ij} - \lambda) + p_i - p_j \geq 0 \end{aligned}$$

## Part (c)

For a $k$-edge cycle of 1 unit of flow, the cost is $\sum_{\text{edges} \in \text{cycle}} c_{ij} \geq k\lambda$ since the prices cancel out. Therefore, the ratio cost to length of any cycle is $\geq \lambda$, i.e. $\lambda$ is a lower bound on the ratio cost to length. By maximizing $\lambda$, the LP finds the greatest lower bound, which corresponds to the value of the smallest ratio cost to length.

## Part (d)

I suggest a combinatorial algorithm that uses binary search to find the right $\lambda$ to solve the dual problem.

My initial search space is between $\lambda_l = \min_{ij} c_{ij}$ and $\lambda_u = \max_{ij} c_{ij}$. I can lower bound the difference between the smallest and next smallest mean cost of a cycle by $\frac{1}{n^2}$. Indeed, let $\frac{a}{b}$ be the smallest mean cost of a cycle ($a$ being the total cost of a cycle, and $b$ its length) and $\frac{c}{d}$ be the next smallest mean cycle. Then $|\frac{a}{b} - \frac{c}{d}| = \frac{1}{bd}|ad - cb| \geq \frac{1}{n^2}$ because $b \leq n$ and $c \leq n$. This means that once my search space interval is $\leq \frac{1}{n^2}$, I can stop the search because I am closer to the right $\lambda$ than any possible other value.

For a given $\lambda$, I assign prices by adding a vertex $s'$ and edges of length 0 from $s'$ to any other vertex and running Bellman-Ford using the costs $c_{ij} - \lambda$ as the edge length and then setting $p_i = d(s', i)$. All edges must have non-negative reduced cost $((c_{ij} - \lambda) + p_i - p_j)$ to satisfy the LP constraints. If the constraints are not satisfied, then the given $\lambda$ is too large, so I make it the new upper bound $\lambda_u$. If the constraints are satisfied, then the given $\lambda$ becomes the new lowerbound $\lambda_l$.

Once the binary search stops (because $\lambda_u - \lambda_l < \frac{1}{n^2}$), I can find a minimum mean cycle by finding a cycle of edges with reduced cost of 0 (or, actually, between 0 and $\lambda_u - \lambda_l$, since $\lambda = \lambda_l$ is just very close to optimal).

# Problem 2

## Part (a)

I show that if Alice's mixed strategy is known, then Bob has a pure strategy serving as his best response.

Let $A_j$ represent the $j$th column of the matrix $A$. Let $j' = \arg\max_j xA_j$. Consider the pure strategy $y_{j'} = 1$ and $y_j = 0, \forall j \neq j'$. This strategy is optimal because $xAy = \sum_j xA_j y_j = xA_{j'} \geq xA\lambda$ for any other strategy $\lambda$ which will dilute the max $xA_{j'}$ by mixing in other components.

## Part (b)

I show how to convert Alice's and Bob's programs above into linear programs, and thus find an optimal strategy for both players in polynomial time.

Let $A_j$ be the $j$th column of $A$ and $a'_i$ be the $i$th row of $A$.

The program for Alice is:

$$\begin{aligned} w &= \min \lambda \\ \sum_i x_i &= 1 \\ \lambda - xA_j &\geq 0, \forall j \\ x_i &\geq 0, \forall i \end{aligned}$$

The program for Bob is:

$$\begin{aligned} z &= \max \lambda \\ \sum_j y_j &= 1 \\ \lambda - a'_i y &\leq 0, \forall i \\ y_j &\geq 0, \forall j \end{aligned}$$

## Part (c)

Consider Alice's linear program. It gives the optimum because it finds the minimum expected payoff in the space of possible mixed strategies for Alice while inferring Bob's best response to Alice, which by part (a) is a pure strategy. Because Bob will respond by playing the pure strategy that maximizes the expected payoff, the expected payoff for a given Alice's strategy $x$ is just $\lambda = \max xA_j$. The constraint $\lambda - xA_j \geq 0, \forall j$, ensures that $\lambda \geq \max xA_j$. Minimizing $\lambda$ ensures that this inequality is tight and that the best of Alice's strategies is chosen. The case for Bob's linear program is symmetric.

## Part (d)

The dual of Alice's linear program is Bob's linear program, so by strong duality, Alice's expected payoff using her best strategy is equal to Bob's expected payoff using his best strategy.

# Problem 3

Given a standard form LP $\min\{cx|Ax = b, x \geq 0\}$, I define a different LP with an obvious basic feasible point ($x = 0$), whose optima are precisely the feasible points of the original LP.

$a_i'$ is the $i$th row of $A$.

$$\min\{-\sum_{\forall i \text{ s.t. } b_i \geq 0} a_i'x + \sum_{\forall i \text{ s.t. } b_i < 0} a_i'x\}$$

$$\text{subject to the constraints}$$

$$0 \leq a_i'x \leq b_i, \forall i \text{ s.t. } b_i \geq 0,$$

$$b_i \leq a_i'x \leq 0, \forall i \text{ s.t. } b_i < 0,$$

$$x \geq 0.$$

The minimum of this LP is $\sum_i -|b_i|$ and is achieved exactly on feasible points of the original LP, since, then, $Ax = b, x \geq 0$.

I can therefore use this different LP (modified to standard form) to start the simplex algorithm. I call the simplex algorithm on this different LP with the obvious basic feasible solution $x = 0$, which will return a feasible point of the original LP. I transform the returned feasible point into a basic feasible point, by repeatedly making the solution more basic by sliding it in the direction of a constraint that's not yet tight. I can now use this basic feasible point to start the simplex algorithm on the original LP.

# Problem 4

## Part (a)

I prove that if there is an independent set of size $k$ in $G$, then there is an independent set of size $k^2$ in the product graph.

Let the independent set of vertices in $G$ of size $k$ be $S$. I construct an independent set of vertices in the product graph of size $k^2$, $S'$, as follows. For each $v \in S$, I add the $k$ vertices of $G_v$ corresponding to $S$ to $S'$. Since I add $k$ vertices for each of the $k$ vertices of $S$, $S'$ will contain $k^2$ vertices at the end of the process. These $k^2$ vertices are independent. Indeed, take any two vertices $v', u' \in S'$. Say $v' \in G_v$ and $u' \in G_u$. If $u = v$, then $v'$ and $u'$ are independent because $G_v$ is a copy of $G$ and the corresponding vertices are independent in $G$. If $u \neq v$, then $v'$ and $u'$ are independent because $v$ and $u$ are independent in $G$ and so any vertices between $G_v$ and $G_u$ are independent since there are no edges between vertices in $G_v$ and vertices in $G_u$.

## Part (b)

I prove that given an independent set of size $s$ in the product graph, one can find an independent set of size $\sqrt{n}$ in G.

Partition the independent set $S$ of size $s$ in the product graph into sets $S_v$ of size $s_v$, where $S_v$ is the set of vertices of $S$ belonging to $G_v$. Let $k$ be the number of non-empty such sets $S_v$. If $k \leq \sqrt{n}$, then there exists at least one $S_v$ with $s_v \geq \frac{n}{k} \geq \sqrt{n}$, and, since the vertices in $S_v$ are independent in $G_v$, the corresponding $\geq \sqrt{n}$ vertices in $G$ are independent too. If $k \geq \sqrt{n}$, then there $\geq \sqrt{n}$ vertices in distinct $G_v$'s that are independent in the product graph, which means that the vertices corresponding to these $G_v$'s are independent in the graph $G$ as there are no edges between them.

## Part (c)

I prove that if there is an $\alpha$-approximation for MIS for some *fixed* $\alpha$, then there is a polynomial approximation scheme for MIS.

For a fixed $\alpha$, I can find a $k$ such that $1 + \epsilon \geq \alpha^{\frac{1}{2^k}}$. I construct a product graph $G_1$ from $G$, then a product graph $G_2$ from $G_1$, and so on, until the product graph $G_k$ from $G_{k-1}$.

Let OPT be the size of the MIS in $G$. Let OPT$'$ be the size of the MIS in $G_k$. Note that OPT' = OPT$^{2^k}$. I find an $\alpha$-approximation of a MIS in $G_k$, with size $s'$, s.t. $\alpha s' \geq$ OPT'. By part (b), I can find a independent set in $G$ of size $s$, s.t. $s = s'^{\frac{1}{2^k}} \geq \frac{\text{OPT}'^{\frac{1}{2^k}}}{\alpha^{\frac{1}{2^k}}} \geq \frac{\text{OPT}^{\frac{1}{2^k}}}{\alpha^{\frac{1}{2^k}}}$. Hence, since $(1 + \epsilon)s \geq$ OPT, there is a approximation scheme for MIS.

The algorithm runs in polynomial time for a fixed $\epsilon$ because the size of the graph $G_k$ increases polynomially for a fixed $k$.

# Problem 5

## Part (a)

Supposing the optimum diameter $d$ known, I devise a greedy 2-approximation algorithm.

I repeat until all points are in a cluster: For any point not yet in a cluster, I create a cluster of all the points not yet in a cluster within distance $d$ of it.

This algorithm ensures that the maximum diameter is $\leq 2d$. In addition, it will not create more than $k$ clusters. Indeed, for any point acting as the cluster center (center point), all points in the same cluster as the center point in the optimal solution will have been considered for inclusion. So all center points are in different clusters in the optiomal solution. Thus the number of clusters in the optimal solution is greater than or equal to the number of center points, which is equal to the number of clusters created by this algorithm.

## Part (b)

The new algorithm ensures that once there are $k$ center points, all points will be within distance $d$ of a center point. Indeed, as long as there is a point that is at a distance $> d$ from all current center points, it belongs to a different cluster in the optimal solution and so, as long as that's true, the center points belong to different clusters in the optiomal solution. Since each center point considers for inclusion in its cluster only points that are within distance $d$ of it, the new algorithm is a 2-approximation like the algorithm in part (a).