1

1.1 Proof

Suppose we have such a linear programming problem P:

$$\begin{aligned} &\text{minimize } c \cdot x \\ &\text{subject to } Ax \geq 0 \\ &x \geq 0 \\ &x_1 = 0 \\ &\text{where } c = [-1, 0, 0, ..., 0]^T \end{aligned} \tag{P}$$

P's dual LP D is thus:

$$\begin{aligned} &\text{maximize } 0 \cdot y \\ &\text{subject to } A^T y \leq c^T \\ &y \geq 0 \\ &\text{where } c = [-1, 0, 0, ..., 0]^T \end{aligned} \tag{D}$$

We assume P is feasible. The objective function $c \cdot x$ of P is bounded to 0 since $c = [-1, 0, ..., 0]^T$ and $x_1 = 0$. By strong duality theorem, we know that D is feasible and bounded as well. Therefore, there must exist y^* satisfying constraints of D. Mathematically,

$$\begin{aligned} &\exists y^* : A^T y^* \leq c^T, y^* \geq 0, c = [-1, 0, ..., 0]^T \\ \Rightarrow\ &\exists y^* : A^T y^* \leq c^T \leq 0, A_1 y^* = y^* \cdot A_1 \leq -1 < 0, y^* \geq 0 \end{aligned}$$

In other words, there exists $y \in \mathbb{R}^m$ such that $A^T y \leq 0, y \geq 0$ and $y \cdot A_1 < 0$.

1.2 Proof

The dual D:

$$\begin{aligned} &\text{maximize } b \cdot y \\ &\text{subject to } A^T y \leq c^T \\ &y \text{ free} \end{aligned} \tag{D}$$

x^* is an optimal basic feasible solution thus the primal is feasible and bounded. Thus we could apply strong duality here, i.e. the dual D is feasible and bounded. Particularly, $c \cdot x^* = b \cdot y^*$. y^* is feasible and optimal. Now we apply one of the complementary slack properties. That is,

$$\forall j : x_j^* = 0 \vee y^{*T} A_j = c_j$$

At least one of the conditions above must be satisfied. Furthermore, we know that every $x^* = A_B^{-1} b > 0$ since B is non-degenerate. Therefore, the second condition must be satiesfied

$$\forall j : y^{*T} A_j = c_j \tag{1}$$

must be satisfied. Finally, assume that y^* is not unique, there must be at least two $y_1^* \neq y_2^* \Rightarrow \exists j : c_j = y_1^{*T} A_j \neq y_2^{*T} A_j = c_j$ which leads to a contradiction. Therefore, the optimal solution of D, y^*, must be unique.

1.3 Proof

$$\begin{aligned}
&P \cap Q = \emptyset \\
\Rightarrow &\forall p \in P : p \notin Q \\
\Rightarrow &\exists i \in [n], \forall p \in P : \hat{a}_i^T p > \hat{b}_i \text{ where } \hat{a}_i^T \text{ is } i\text{th row of } \hat{A}
\end{aligned}$$

We also know that $\exists i \in [n], \forall q \in Q : \hat{a}_i^T q \leq \hat{b}_i$

$$\begin{aligned}
\text{Therefore, } &\exists i \in [n], \forall p \in P, \forall q \in Q : \hat{a}_i^T p > \hat{a}_i^T q \\
\Rightarrow &\hat{a}_i^T (p - q) = (p - q) \cdot \hat{a}_i > 0
\end{aligned}$$

Therefore, I have proven there exists $c := \hat{a}_i$ such that $(p - q) \cdot c > 0$ for every $p \in P$ and $q \in Q$.

2

2.1 Proof

- Assume that there are two pure equilibria. Then there are two different optimal elements (P_{ij} and $P_{i'j'}$ where $i' \neq i \vee j' \neq j$) in the payoff matrix. Since they are optimal, $P_{ij} = P_{i'j'}$. Otherwise either one of them is not optimal. However $P_{ij} = P_{i'j'}$ contradicts that no two outcomes in the payoff matrix have the same value.
- For multiple number of equilibria ($\# > 2$), we can choose 2 of them and the problem is reduced to the 2 equilibria problem.
- 0 equilibrium: an instance(paper scissor rock) was given in the lecture.
- 1 equilibrium: an instance of such a payoff matrix $[0]$

In summary, there could be either 0 or 1 equilibria.

2.2 Mixed Equilibria and probability

Suppose we have such a labeled 2×3 grid.

1	2	3
4	5	6

Then we have such a payoff matrix P where 1 stands for I win the game while -1 stands for you win the game.

					You			
		12	23	45	56	14	25	36
	1	1	-1	-1	-1	1	-1	-1
	2	1	1	-1	-1	-1	1	-1
I	3	-1	1	-1	-1	-1	-1	1
	4	-1	-1	1	-1	1	-1	-1
	5	-1	-1	1	1	-1	1	-1
	6	-1	-1	-1	1	-1	-1	1

Now let's notate my mixed strategy as $\sum_{i=1}^{6} x_i = 1$ and yours as $\sum_{j=1}^{7} y_j = 1$
The value of the mixed equilibria must satisfy

$$\sum_i \sum_j \hat{x}_i y_j p_{ij} | \forall \hat{x} \leq \sum_i \sum_j x_i^* y_j^* p_{ij} \leq \sum_i \sum_j x_i \hat{y}_j p_{ij} | \forall \hat{y}$$

It is not hard to see that both the set of column vectors of the payoff matrix is linearly independent. Thus, x^* must be uniformly distributed since $x_i^* \geq 0$. Otherwise, $\exists \hat{x}$: $\sum_i \sum_j \hat{x}_i y_j p_{ij} > \sum_i \sum_j x_i^* y_j^* p_{ij} \Rightarrow x^*$ is not optimal. Similarly, y^* must be uniformly distributed as well. The value of mixed equilibria is $\frac{1}{6}\frac{1}{7} \sum_i \sum_j p_{ij} = \frac{1}{42}(-14) = -\frac{1}{3}$. $2p - 1 = -1/3$ where p is the probability that I win. So $p = 1/3$

2.3 Proof

Let's say the modified payoff matrix is P'. The game value for P is V and game value for P' is V'.
For the original game,

$$\max_i \sum_j y_j^* p_{ij} \leq V = \sum_i \sum_j x_i^* y_j^* p_{ij} \leq \min_j \sum_i x_i^* p_{ij}$$

Since k column is the min column, the modification does not affect LHS. Player x doesn't change the mixed strategy. For the RHS, the min for P' can only be larger or equal

$$\sum_i \sum_j x_i^* y_j^* p_{ij} \leq \min_j \sum_i x_i^* p_{ij} \leq \min_j \sum_i x_i'^* p'_{ij}$$

Thus the player y still cannot profit from deviating. As a consequence $\sum_i \sum_j x_i'^* y_j'^* p'_{ij}$ remains the same. In other words, the value of the new game is the same.

3

3.1 ILP

Define variables

- x_{ij}: the student s_i is assigned to the project p_j
- y_{il}: the student s_i is assigned to lth ranked project out of k picked projects.

Objective: minimize average unhappiness

$$\text{minimize } \frac{1}{n} \sum_{i=1}^{n} \sum_{l=1}^{k} l y_{il}$$

Constraints:

- either assigned or not:

$$\forall i \in [n], j \in [m], l \in [k] : x_{ij}, y_{il} \in \{0, 1\}$$

- Every student must be assigned to exactly one project.

$$\forall i \in [n] : \sum_{j=1}^{m} x_{ij} = 1$$

$$\forall i \in [n] : \sum_{l=1}^{k} y_{il} = 1$$

- We may assign multiple students to the same project, but no project may have more than 3 students assigned to it.

$$\forall j \in [m] : \sum_{i=1}^{n} x_{ij} \leq 3$$

- students may not be assigned to projects that are not in their k preferred projects.

$$\forall i \in [n], j \in [m], l \in [k] : x_{ij} \leq y_{il}$$

3.2 Relaxation

$$\begin{aligned}
&\text{minimize } \frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{k} l y_{il} \\
&\text{subject to } \forall i \in [n], j \in [m], l \in [k] : x_{ij}, y_{il} \in [0,1] \subset \mathbb{R} \\
&\qquad \forall i \in [n] : \sum_{j=1}^{m} x_{ij} = 1 \\
&\qquad \forall i \in [n] : \sum_{l=1}^{k} y_{il} = 1 \\
&\qquad \forall j \in [m] : \sum_{i=1}^{n} x_{ij} \leq 3 \\
&\qquad \forall i \in [n], j \in [m], l \in [k] : x_{ij} \leq y_{il}
\end{aligned} \tag{LP}$$

3.3 Proof

The strategy is to show that if a solution is feasible w.r.t. the constraints and not integral, then the solution is not a basic solution as it is not an extreme point. Let's firstly prove that x_{ij} is integral.
Assumption: Suppose that $\exists i \in [n], \exists j \in [m] : x_{ij} \in (0,1)$ for a feasible solution.
For such i, $\exists j' \neq j \in [m] : x_{ij'} \in (0,1)$. Otherwise, $x_{ij'} = 0 \vee 1$ where $j' \neq j$, which is infeasible for the constraint $\forall i \in [n] : \sum_{j=1}^{m} x_{ij} = 1$ contradicting to the assumption. In addition, $x_{ij} \neq 1$. Therefore, there are at least 2 $x_{ij} \in (0,1)$. We denote these two as x_{il} and $x_{il'}$.
Let's choose an ϵ satisfying

$$0 < \epsilon < \min_{x_{ij}>0} x_{ij}$$

Now we construct x'_{ij} in such a way

$$x'_{ij} = x_{ij} - \epsilon \text{ if } j = l$$
$$x'_{ij} = x_{ij} + \epsilon \text{ if } j = l'$$
$$x'_{ij} = x_{ij} \text{ otherwise}$$

. and x''_{ij}

$$x''_{ij} = x_{ij} + \epsilon \text{ if } j = l$$
$$x''_{ij} = x_{ij} - \epsilon \text{ if } j = l'$$
$$x''_{ij} = x_{ij} \text{ otherwise}$$

Now we know that $x \neq x'$, $x \neq x''$ and $x = x' + x''$. Therefore, x is not an extreme point.
Similarly, the proof above applies to y as well since $\forall i \in [n] : \sum_{l=1}^{k} y_{il} = 1$. Therefore, we have proved that non-integral feasible solutions are not basic feasible solutions. In other words, the linear relaxation will always have an integral optimal solution.

3.4 Poly time solution

$OPTILP \leq OPTLP$ and we have shown that $OPTLP$ is always integral $\Rightarrow OPTILP = OPTLP$. Now we only need to solve LP to get the optimal value for ILP. Simply use an ellipsoid algorithm to solve LP which has been shown to be in poly-time.(Or we can reduce[Cook reduction] this LP to calling an "oracle", which is a well known P algorithm to solve max bipartite perfect matching by polynomial times)Therefore, we can solve the original ILP by solving the relaxed version in poly-time.

4

4.1 ILP

Traveling Salesman Problem:
Given a set of cities $S = \{1, 2, ..., n\}, n > 2$

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{n} \sum_{j=i+1}^{n} d_{ij} x_{ij} \\ \text{subject to } & \forall i \in [n] : \sum_{j=1:j\neq i}^{n} x_{ij} = 2 \\ & \forall \emptyset \subset S \subset E : \sum_{(i,j)\in\delta(S)} x_{ij} \geq 2 \\ & \forall (i,j) \in E : x_{ij} \in \{0,1\} \end{aligned} \tag{TSP}$$

Square Traveling Salesman Problem:
Given a set of coordinate-pairs $X_n \subset \mathbb{R}^2$ of size $n > 2$

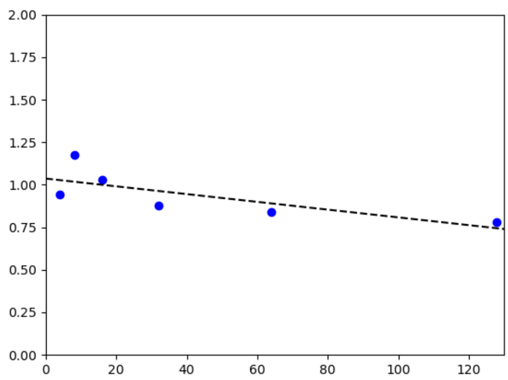
$$\begin{aligned} \text{minimize} \quad & \sum_{p \in X_n} \sum_{q \in X_n : q \neq p} \frac{1}{2} ||p - q||_2^2 x_{pq} \\ \text{subject to } \forall p \in X_n : \quad & \sum_{q \in X_n : q \neq p} x_{pq} = 2 \\ \forall \emptyset \subset S \subset E : \quad & \sum_{(p,q) \in \delta(S)} x_{pq} \geq 2 \\ \forall (p,q) \in E : \quad & x_{pq} \in \{0,1\} \end{aligned} \tag{STSP}$$

It doesn't matter that $||p?q||_2^2$ is not a linear function as p and q are not variables.

4.2 average objective for sampling 10 times

size	4	8	16	32	64	128
optimal objective	0.94	1.17	1.03	0.88	0.84	0.77

4.3 plot



4.4 asymptotic hypothesis

$h(n) = 1$.

5 Appendix

```
import sys
import math
import random
from itertools import combinations
import gurobipy as gp
from gurobipy import GRB
import matplotlib.pyplot as plt
import math
```

```
from scipy.stats import linregress
#the main frame is from the official tsp.py example from gurobi website
##some parts are modified to adapt the requirements for question 4

#prepare n_size
n_values = []
for i in range(2,8):
    n_values.append(int(2**i))
n_averages = []
# Create n random points

for n in n_values:
    # sample 10 times
    total = 0
    count = 0
    #calculate 10 rep average
    for j in range(10):
        count += 1
        points = [(random.random(), random.random()) for i in range(n)]

        dist = {(i, j):
                (sum((points[i][k]-points[j][k])**2 for k in range(2)))
                for i in range(n) for j in range(i)}

        m = gp.Model()

        # Create variables
        vars = m.addVars(dist.keys(), obj=dist, vtype=GRB.BINARY, name='e')
        for i, j in vars.keys():
            vars[j, i] = vars[i, j]

        m.addConstrs(vars.sum(i, '*') == 2 for i in range(n))

        m._vars = vars
        m.Params.LazyConstraints = 1
        m.optimize(subtourelim)

        vals = m.getAttr('X', vars)
        tour = subtour(vals)
        assert len(tour) == n

        total += m.ObjVal
    average = total / count
    n_averages.append(average)

x = n_values
y = n_averages
```

```
reg = linregress(x, y)
plt.plot(x, y,'bo')
plt.axis([0, 130, 0, 2])
plt.axline(xy1=(0, reg.intercept), slope=reg.slope, linestyle="--", color="k")
plt.show()


# Callback - use lazy constraints to eliminate sub-tours
def subtourelim(model, where):
    if where == GRB.Callback.MIPSOL:
        vals = model.cbGetSolution(model._vars)
        # find the shortest cycle in the selected edge list
        tour = subtour(vals)
        if len(tour) < n:
            # add subtour elimination constr. for every pair of cities in tour
            model.cbLazy(gp.quicksum(model._vars[i, j]
                                     for i, j in combinations(tour, 2))
                         <= len(tour)-1)

def subtour(vals):
    # make a list of edges selected in the solution
    edges = gp.tuplelist((i, j) for i, j in vals.keys()
                         if vals[i, j] > 0.5)
    unvisited = list(range(n))
    cycle = range(n+1)  # initial length has 1 more city
    while unvisited:  # true if list is non-empty
        thiscycle = []
        neighbors = unvisited
        while neighbors:
            current = neighbors[0]
            thiscycle.append(current)
            unvisited.remove(current)
            neighbors = [j for i, j in edges.select(current, '*')
                         if j in unvisited]
        if len(cycle) > len(thiscycle):
            cycle = thiscycle
    return cycle
```