Homework 6

Kyle Hadley

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
import cvxpy as cp
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
```

In [2]:
```
import warnings
warnings.simplefilter('ignore')
```

1. Graph Constraints

(a)

Given the graph, the incidence matrix E and source-sink vector b are written as,

$$E = \begin{bmatrix} -1 & 0 & 0 & -1 & 0 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & -1 \end{bmatrix}$$

$$b = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

(b)

Given the graph, the route indicator matrix R can be written as,

$$R = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Solving for ER we find that,

$$ER = \begin{bmatrix} -1 & 0 & 0 & -1 & 0 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$ER = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

From this, we can see that ER is equivalent to $\begin{bmatrix} b & b & b \end{bmatrix}$ (i.e. the column vector b repeated in each column of ER).

(c)

To show that E is not full row-rank, we first compute $1^T E$

$$1^T E = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 0 & -1 & 0 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & -1 \end{bmatrix} = 0$$

We can also show that $1^T b = 0$ for our problem,

$$1^T b = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

(d)

Given that $U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$, solving for U^{-1} we find that (using numpy solver),

$$U^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{bmatrix}$$

In [3]:

```
U = np.matrix([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [1, 1, 1, 1]])
print('U^-1 =', np.linalg.inv(U))
```

```
U^-1 = [[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
```

```
[ 0.  0.  1.  0.]
[-1. -1. -1.  1.]]
```

(e)

Re-writing our component of the Lagrangian $v^T(Ex - b)$ as $v'^T = (E'x - b')$ where $v'^T = v^T U - 1$, we can solve by substituting v^T in our Lagrangian component such that,

$$v^T(Ex - b) = v'^T U(Ex - b) = v'^T(UEx - Ub)$$

We can see from the derived relationship that $E' = UE$ and $b' = Ub$. Computing E' and b' for out graph given,

$$E' = UE = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 0 & -1 & 0 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & -1 \end{bmatrix}$$

$$E' = \begin{bmatrix} -1 & 0 & 0 & -1 & 0 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$b' = Ub = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Intuitively, it seems that v' represents the value function as it relates to routes from the origin and to our destination. The addition of the U matrix eliminates all edges leaving our destination node, which is 4 in this situation.

The useless row in the equation $E'x = b'$ is the 4th row. When we remove this row then the equation is full row rank.

2. Shortest Path: Explicit Path Enumeration

Consider the shortest path linear program $\min_{z \in \mathbb{R}^3} \ell^T z = c^T R z$ s.t. $1^T z = 1, z \geq 0$.

(a)

z represents the mass distribution vector through the graph. The elements of ℓ represents the cost of taking a given route, where ℓ_i is the costing of taking route R_i.

(b)

The dual of this optimization problem using λ and u is

$$\max_{\lambda, u \geq 0} \lambda$$

$$s.t. \, \lambda 1^T + u^T = c^T R, u \geq 0$$

In this case, λ represents the reward or cost of a given route. So in this case our problem will find a λ that is equivalent to the cost of the optimal route. u represents the inefficient edges in our optimization problem.

(c)

Given that z represents the routes in our graph, for an optimal route $z_i > 0$. Based on the slackness condition $z_i u_i = 0$, $z_i > 0$ implies that $u_i = 0$ - and we previously noted that u represents inefficienies in our problem. Thus, when $u_i = 0$ there are no inefficienies and that means that the mass distribution vector puts all the mass on the optimal route.

3. Shortest Path: Edge Formulation

Consider the shortest path linear program such that $\min_{x \in \mathbb{R}^6} c^T x$ s.t. $Ex = b, x \geq 0$ for $c \in \mathbb{R}^6$.

(a)

The Lagrangian of this optimization problem using v and μ is as follows,

$$L(x, v, \mu) = c^T x + v^T (Ex - b) - \mu^T x$$

(b)

Solving for the KKT conditions, we find that,

Stationarity : $\frac{\partial L}{\partial x} = c^T + v^T E - \mu^T = 0$

Feasibility : $\frac{\partial L}{\partial v} = Ex - b = 0 \Rightarrow Ex = b$

Positivity : $x \geq 0, \mu \geq 0$

Slackness : $\mu^T x = 0$ (i.e. $x_e \mu_e = 0$)

(c)

From the stationarity condition, we know that $c^T + v^T E - \mu^T = 0$ which can be written as $c_e + v_{s'} - v_s - \mu_e = 0$ for $\forall e \in E$. Taking the summation of this relationship yields us,

$$\sum_{e \in r} (c_e + v_{s'} - v_s - \mu_e) = 0$$

$$\sum_{e \in r} c_e + \sum_{e \in r} (v_{s'} - v_s) - \sum_{e \in r} \mu_e = 0$$

Our second term can be re-written as $v_d - v_o$ which is the difference in cost to go from the origin to the destination. Thus,

$$\sum_{e \in r} c_e + v_d - v_o - \sum_{e \in r} \mu_e = 0$$

$$\sum_{e \in r} c_e = v_o - v_d + \sum_{e \in r} \mu_e$$

Our left-most term is the total travel cost $\left(\sum_{e \in r} c_e\right)$, our middle term is the cost to go from the origin to the destination, and the right-most term is the sum of inefficiences along a given route.

If we are in an optimal route, $x_e > 0$; thus from our slackness relationship $(x_e \mu_e = 0)$, $\mu_e = 0$, thus we can see that $\sum_{e \in r} c_e = v_d - v_o$.

If we are in a suboptimal route, $x_e = 0$; thus from our slackness relationship $(x_e \mu_e = 0)$, $\mu_e >= 0$, thus we can see that $\sum_{e \in r} c_e = v_o - v_d + \sum_{e \in r}$.

Thus, we can see that for any route from the origin to the destination (optimal or suboptimal), the total travel cost is greater than or equal to the minimum travel cost.

(d)

If x_i represents our mass edge flow, we know that at optimum $x_i > 0$ as mass will be flowing. Given our slackness constraint $x_i \mu_i = 0$, $x_i > 0$ implies that $\mu_i = 0$ - where μ represents the inefficienies of a given edge. Thus, when $u_i = 0$ there are no inefficienies along the edges for a given x_i and mass only chooses optimal routes.

(e)

Given $\min_{x \in \mathbb{R}^6} c^T x$ s.t. $Ex = b, x \geq 0$ for $c \in \mathbb{R}^6$ and the Lagrangian from part (a)
$L(x, v, \mu) = c^T x + v^T (Ex - b) - \mu^T x$, we can re-write our initial problem as,

$$\min_x \left(\max_{v, \mu \geq 0} \left(c^T x + v^T (Ex - b) - \mu^T x \right) \right)$$

We know that,

$$\min_x \left(\max_{v, \mu \geq 0} \left(c^T x + v^T (Ex - b) - \mu^T x \right) \right) \geq \max_{v, \mu \geq 0} \left(\min_x \left(c^T x + v^T (Ex - b) - \mu^T x \right) \right)$$

Taking the RHS, we can solve for where x is minimized by taking the derivative with respect to x and find that

$$c^T + v^T E - \mu^T = 0$$

when x is minimized. Re-writing the RHS, we find that

$$c^T x + v^T(Ex - b) - \mu^T x = (c^T + v^T E - \mu^T) - v^T b$$

Given that we know that $c^T + v^T E - \mu^T = 0$, we can write the dual problem as

$$\min_{v,\mu \geq 0} -v^T b$$

$$\text{s.t. } c^T + v^T E - \mu^T = 0, \mu \geq 0$$

(f)

Re-writing in terms of v', we know $U^{-1}E' = E$ and $U^{-1}b' = b$ from problem (1), thus we can re-write the dual problem as,

$$\min_{v',\mu \geq 0} -v'^T U^{-1} b'$$

$$\text{s.t. } c^T + v'^T U^{-1} E' - \mu^T = 0, \mu \geq 0$$

It seems that reformulating the linear program in this form aggregates the masses to the optimal route, lowers the λ and eliminates the μ from the optimal route as we saw when we looked at the illustrations of the dual problem from last time.

4. Numerical optimization

(a)

Using, cvxpy we can solve problem (1) for the given cost vectors.

For both cost vectors, the resulting optimal primal and dual variables are outputted in console below. The solution represents that optimal path (or paths) for the given graph structure.

For $c^T = \begin{bmatrix} 1 & 3 & 1 & 3 & 1 & 1 \end{bmatrix}$, the optimal solution path is along the edges $1 \Rightarrow 3 \Rightarrow 5$ to go from the origin to the destination.

For $c^T = \begin{bmatrix} 1 & 2 & 1 & 3 & 1 & 1 \end{bmatrix}$, there are two paths that have the same cost (i.e. two optimal solution paths); the path edges are defined as: $1 \Rightarrow 3 \Rightarrow 5$ and $1 \Rightarrow 2$. The output demonstrates this by outputting a value of 0.5 almost like a probability of paths that can be accessed. The decision at node 2 to go down edges 2 or 3 lead to equivalent final costs, thus are represented by a 0.5.

In [4]:

```
# Establish our values for input parameters
E = np.matrix([[-1, 0, 0, -1, 0, 1], [1, -1, -1, 0, 0, 0], [0, 0, 1, 1, -1, 0], [0, 1,
b = np.matrix([[-1], [0], [0], [1]])
c_T = np.matrix([1, 3, 1, 3, 1, 1]) # Define c_T as first cost vector

# Define and solve the CVXPY problem.
x = cp.Variable(shape=(6, 1))
prob = cp.Problem(cp.Minimize(c_T @ x), [E @ x == b, x >= 0])
prob.solve()
```

```
# Print result.
print('For C =', c_T)
print("The optimal value is", np.round_(prob.value, decimals = 1))
print("\nA solution x is:")
print(np.round_(x.value, decimals=1))
print("\nA dual solution is: ")
print(np.round_(prob.constraints[0].dual_value, decimals=1), '\nand \n', np.round_(prob
```

```
For C = [[1 3 1 3 1 1]]
The optimal value is 3.0

A solution x is:
[[ 1.]
 [ 0.]
 [ 1.]
 [ 0.]
 [ 1.]
 [-0.]]

A dual solution is:
[[ 1.5]
 [ 0.5]
 [-0.5]
 [-1.5]]
and
 [[0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [4.]]
```

In [5]:

```
# Define c_T as second cost vector
c_T = np.matrix([1, 2, 1, 3, 1, 1])

# Define and solve the CVXPY problem.
x = cp.Variable(shape=(6, 1))
prob = cp.Problem(cp.Minimize(c_T @ x), [E @ x == b, x >= 0])
prob.solve()

# Print result.
print('For C =', c_T)
print("The optimal value is", np.round_(prob.value, decimals = 1))
print("\nA solution x is:")
print(np.round_(x.value, decimals=1))
print("\nA dual solution is: ")
print(np.round_(prob.constraints[0].dual_value, decimals=1), '\nand \n', np.round_(prob
```

```
For C = [[1 2 1 3 1 1]]
The optimal value is 3.0

A solution x is:
[[ 1. ]
 [ 0.5]
 [ 0.5]
 [ 0. ]
 [ 0.5]
 [-0. ]]

A dual solution is:
[[ 1.5]
```

```
 [ 0.5]
 [-0.5]
 [-1.5]]
and
 [[0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [4.]]
```

(b)

Using, cvxpy we can solve problem (2) for the given cost vectors.

For both cost vectors, the resulting optimal primal and dual variables are outputted in console below. The solution identifies which route or routes are the optimal path for the given graph structure.

For $c^T = \begin{bmatrix} 1 & 3 & 1 & 3 & 1 & 1 \end{bmatrix}$, the optimal solution path is route 2 which is the edges $1 \Rightarrow 3 \Rightarrow 5$ to go from the origin to the destination.

For $c^T = \begin{bmatrix} 1 & 2 & 1 & 3 & 1 & 1 \end{bmatrix}$, there are two paths that have the same cost (i.e. two optimal solution paths); routes 1 and 2 defined by the edges: $1 \Rightarrow 3 \Rightarrow 5$ and $1 \Rightarrow 2$. Similar to part (a), the output demonstrates this by outputting a value of 0.5 almost like a probability of paths that can be accessed. The decision at node 2 to go down edges 2 or 3 lead to equivalent final costs, thus are represented by a 0.5.

In [8]:

```
# Establish our values for input parameters
R = np.matrix([[1, 1, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1], [0, 1, 1], [0, 0, 0]])
c_T = np.matrix([1, 3, 1, 3, 1, 1]) # Define c_T as first cost vector
ones = np.ones(shape=(3, 1))

# Define and solve the CVXPY problem.
z = cp.Variable(shape=(3, 1))
prob = cp.Problem(cp.Minimize(c_T @ R @ z), [ones.T @ z == 1, z >= 0])
prob.solve()

# Print result.
print('For C =', c_T)
print("The optimal value is", np.round_(prob.value, decimals = 1))
print("\nA solution z is:")
print(np.round_(z.value, decimals=1))
print("\nA dual solution is: ")
print(np.round_(prob.constraints[0].dual_value, decimals=1), '\nand \n', np.round_(prob
```

```
For C = [[1 3 1 3 1 1]]
The optimal value is 3.0

A solution z is:
[[0.]
 [1.]
 [0.]]

A dual solution is:
[[-3.]]
and
```

```
[[1.]
[0.]
[1.]]
```

In [9]:

```
# Establish our values for input parameters
R = np.matrix([[1, 1, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1], [0, 1, 1], [0, 0, 0]])
c_T = np.matrix([1, 2, 1, 3, 1, 1]) # Define c_T as first cost vector
ones = np.ones(shape=(3, 1))

# Define and solve the CVXPY problem.
z = cp.Variable(shape=(3, 1))
prob = cp.Problem(cp.Minimize(c_T @ R @ z), [ones.T @ z == 1, z >= 0])
prob.solve()

# Print result.
print('For C =', c_T)
print("The optimal value is", np.round_(prob.value, decimals = 1))
print("\nA solution z is:")
print(np.round_(z.value, decimals=1))
print("\nA dual solution is: ")
print(np.round_(prob.constraints[0].dual_value, decimals=1), '\nand \n', np.round_(prob
```

```
For C = [[1 2 1 3 1 1]]
The optimal value is 3.0

A solution z is:
[[0.5]
 [0.5]
 [0. ]]

A dual solution is:
[[-3.]]
and
 [[0.]
 [0.]
 [1.]]
```