**Henrique da Ponte**

**946930-5**

**ECE 133**

**Prof. Alizadeh**

**Weekly Report 1**

**Problem 1**

**a)**

**Code**

import cvxpy as cp

import numpy as np

import random

# ==================== Helper Functions ====================

# Function to compute the node-edge incidence matrix for a directed graph

def incidence\_matrix(h, t, nodes):

m = len(nodes)

n = len(h)

A = np.zeros((m, n))

for j in range(n):

# Finding the index of the start node (head) of edge j

tail = nodes.index(h[j])

# Finding the index of the end node (tail) of edge j

head = nodes.index(t[j])

A[tail, j] = 1

A[head, j] = -1

return A

def generate\_all\_connected\_graph(n\_nodes):

"""Generate an all-connected graph with random capacities."""

h, t, weights = [], [], []

for i in range(1, n\_nodes + 1):

for j in range(i + 1, n\_nodes + 1):

h.append(i)

t.append(j)

weights.append(random.randint(1, 10))

return h, t, weights

def computeS(h, c):

'''

Function to compute the column vector s

with a 1 for every edge leaving the source

and a 0 for every other edge

'''

s = np.zeros(len(c))

for i in range(len(h)):

if h[i] == 1:

s[i] = 1

return s

def computeL(A, source, target):

'''

Function to compute the matrix L

by removing rows corresponding to source and destination

'''

L = np.delete(A, [source-1, target-1], 0)

return L

def solve\_max\_flow(s, L, c):

"""Solve the max flow problem for the given graph."""

f = cp.Variable(len(c)) # Defining decision variable

objective = cp.Maximize(s.T @ f) # Defining the objective function

constraints = [L @ f == 0, 0 <= f, f <= c] # Defining the constraints

problem = cp.Problem(objective, constraints) # Defining the problem

problem.solve() # Solving the problem

maxFlow = round(problem.value) # Optimal value (max flow)

maxFlowEdges = np.around(f.value) # Optimal flow on each edge

return maxFlow, maxFlowEdges

# ==================== Problem 1.a ====================

# Defining the graph (used the same graph outlined in the homework)

h1 = [1, 1, 1, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 9] # head nodes

t1 = [2, 4, 8, 3, 7, 4, 6, 5, 6, 8, 7, 8, 9, 3] # tail nodes

c1 = [10, 10, 1, 10, 1, 10, 1, 1, 12, 12, 12, 12, 6, 4] # column vector of edge capacities.

s1 = computeS(h1, c1) # column vector with a 1 for every edge leaving the source and a 0 for every other edge

# Computing the node-edge incidence matrix for the graph

nodes1 = list(range(1, max(t1) + 1))

A1 = incidence\_matrix(h1, t1, nodes1)

# Computing L by removing rows corresponding to source and destination

L1 = computeL(A1, 1, max(t1))

# CVXPY Max Flow Problem Formulation as linear program

maxFlow1, maxFlowEdges1 = solve\_max\_flow(s1, L1, c1)

# Printing the results

print("Optimal value (max flow):", maxFlow1)

print("Optimal flow on each edge:", maxFlowEdges1)

Results



**b)**

**\*\* Note that although the function only took 1s to compute the problem, the program actually took way more time to run because it had to compute every iteration. Give this remark I considered 1s with e 5 node increment a reasonable amount of time\*\***

**Code**

# ==================== Problem 1.b ====================

import time

timeElapsed = 0

# Starting number of nodes

n = 5

while timeElapsed < 1:

n += 5 # Incrementing number of nodes by 2

# Generating all connected graphs with n nodes

h2, t2, c2 = generate\_all\_connected\_graph(n)

# Defining vector containing the number of all nodes

nodes2 = list(range(1, n + 1))

# Computing incidence matrix

A2 = incidence\_matrix(h2, t2, nodes2)

# Computing L

L2 = computeL(A2, 1, n)

# computing s

s2 = computeS(h2, c2)

start = time.time() # Starting timer

# Solving the max flow problem

maxFlow2, maxFlowEdges2 = solve\_max\_flow(s2, L2, c2)

end = time.time() # Ending timer

timeElapsed = end - start # Computing time elapsed

print("Graph with", n, "nodes took", timeElapsed, "seconds to solve")

print("Number of nodes in graph that took ", timeElapsed," seconds to solve:", n)

print("Optimal value (max flow):", maxFlow2)

print("Optimal flow on each edge:", maxFlowEdges2)

**Results**

**A screenshot of a computer

Description automatically generated**