IE 411
Optimization of Large-Scale Linear Systems
Final Project

Submitted by: Jayesh Yevale
NetID: jyeval2

Introduction

We consider a Image for the seam carving problem. In this project, our primary objective is to tackle a large-scale linear problem using a standard programming language, which is Python in this project, in conjuction with widely known optimization tool known as Gurobi. The image seam carving problem can be formulated as a max-flow problem.

In this project, we plan to utilize our knowledge of IE 411 - Optimization of Large Systems, background to implement the tool for image resizing. We will use max-flow technique to resize the image iteratively for identitfying the nodes to be eliminated. Every iteration we will identify and eliminate the nodes for next iteration. This will help us to find the final "resized" image as expected.

Problem Formulation

For the problem formulation, we decided to use the given information such as the cost is the sum of absolute intensity differences, and the node traversal assumption given in the project problem documentation.

We will consider the image to be a set of nodes, where each pixel denotes an independent node. Let us us consider the graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ are nodes and $\mathcal{E}$ are edges. The graph is created with nodes $(i, j) \in \mathcal{I} \times \mathcal{J}$, where $\mathcal{I} = \{1, \dots, m\}$ and $\mathcal{J} = \{1, \dots, n\}$. The nodes corresponds to pixels in the input image. Node $(k, l) \in \mathcal{I} \times \mathcal{J}$ can be traversed from node $(i, j) \in \mathcal{I} \times \mathcal{J}$ if: (a) $k = i + 1$, and (b) l is either $j - 1, j$ or $j + 1$.

The graph also has a dummy terminal node t that can be traversed from any nodes (m, j), $j = 1, \dots, n$.

The cost of traversing from node (i, j) to node (k, l) is given by $|y(i, j) - y(k, l)|$, where y is intensity of pixels.

Let us now formulate a linear program to compute the shortest path from source node $(1, j)$ (where $j = 50$ in our given problem) to the terminal node t. The optimal path of the linear program would correspond to a seam. We will repeat the steps for 25 iteration to find the final image.

Primal Formulation

Decision variable:

f : represents the actual flow on a edge.

Parameters

$\mathbf{A}$: Incidence matrix, which captures the node-node relationship $\in \mathbb{R}^{m \times n}$.

$$b = \begin{cases} -1, & \text{for source node } s \\ 1, & \text{for terminal node } t \\ 0, & \text{else} \end{cases}$$

c: Capacity of an edge (denoted as “Cap” in coding)

$$\begin{aligned} \min \quad & cf \\ \text{s.t.} \quad & \\ & \mathbf{A}f = b \\ & f \geq 0 \end{aligned}$$

Dual formulation:

Decision variable:

z : represents the new dual decision variable.

$$\begin{aligned} \max \quad & zb \\ \text{s.t.} \quad & \\ & \mathbf{A}^\top z \leq c \\ & z \text{ free} \end{aligned}$$

Complementary slackness:

For complementary slackness, we considered the matrix version as shown in class (for image segmentation).

$$z\left(\mathbf{A}f - b\right) = 0$$
$$f\left(c - \mathbf{A}^\top z\right) = 0$$

This holds true for our cases. Hence, verified.

Please see the code in next page.

[15]:
```python
from PIL import Image
import gurobipy as gp
from gurobipy import GRB
from gurobipy import *
import gurobipy as gp
import numpy as np
import networkx as nx
import time
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

[16]:
```python
im = Image.open('tower.png').convert('L') #convert to grayscale
h = im.height
w = im.width

tic = time.time()

#grayscale values; list of size h*w , where each values is integer in range␣
↪[0,255]
intensity = list(im.getdata())

def get_weight(n1,n2):
    if n2 == t:
        diff = 0
    else:
        diff = intensity[n1-1] - intensity[n2-1]
    return np.abs(diff)

#define indexes
def idx(i,j):
    return w*(i-1)+j

#specify source node
s = 50
```

[17]:
```python
for q in range(25):

    #list of coordinates
    coords = [(i,j) for i in range(1,h+1) for j in range(1,w+1)]

    nodes = [idx(i,j) for i,j in coords]

    #add destination (t) to nodes
    t = h*w+1
    nodes.append(t)
```

```
#define neighbors
ngbrs = {idx(i,j): [idx(i2,j2) for (i2,j2) in [(i+1,j-1),(i+1,j),(i+1,j+1)]
                      if 1<= i2 <= h and 1<= j2 <= w] for (i,j) in coords}

#add neighbors for destination nodes
for (i,j) in coords:
    if i ==h:
        ngbrs[idx(i,j)].append(t)

#define edges
edges = [(n1,n2) for n1 in nodes[:-1] for n2 in ngbrs[n1]]


#define incident matrix
G = nx.DiGraph()
G.add_nodes_from(nodes)
G.add_edges_from(edges)
A = nx.incidence_matrix(G, oriented= True)

#intialization
cap = np.zeros(G.number_of_edges())
r = np.zeros(G.number_of_edges())
for i in range(G.number_of_edges()):
    cap[i] = get_weight(edges[i][0],edges[i][1])

#define the primal problem
m = gp.Model("Primal Image Seaming")
m.setParam('OutputFlag', 0)
m.Params.LogToConsole = 0
m.Params.Method = 0
f = m.addMVar(G.number_of_edges(), vtype=GRB.CONTINUOUS,lb=0)
b = np.zeros(G.number_of_nodes())
b[s-1] = -1
b[-1] = 1
m.addConstr(A@f==b)
obj= cap@f


m.setObjective(obj, GRB.MINIMIZE)
m.optimize()
print("\nIteration: ",q+1,"\nPrimal Objective: ",m.getObjective().getValue())

if q == 0:
    toc = time.time()
    #remove the seam nodes
    flows = m.getAttr("X", m.getVars())
    to_remove = [] # get the indices of nodes (pixels) to be removed
    for i in range(G.number_of_edges()):
        if flows[i] >= 0.9:
            to_remove.append(edges[i][0]-1) #0 indexed
    intensity = [I for idx,I in enumerate(intensity) if idx not in to_remove]
    w = w - 1

    #DUAL PROBLEM
    mm = gp.Model("Dual Image Seaming")
    mm.Params.LogToConsole = 0
    mm.Params.Method = 0
    z = mm.addMVar(shape=G.number_of_nodes(), vtype=GRB.CONTINUOUS, lb=0,␣
↪name="z")
    obj = b@z

    mm.addConstr(cap>=A.transpose()@z)
    mm.setObjective(obj, GRB.MAXIMIZE)
    mm.optimize()
    print("Dual objective:   ", mm.getObjective().getValue())
    dual_vars = mm.getAttr("X", mm.getVars())

    #Check complementary slackness conditions
    cuts = dual_vars[:G.number_of_edges()]
    y = dual_vars[G.number_of_edges():]
    cmaxflows = A@flows - b
    print('Complementary slackness: ', cmaxflows@cuts)
    Aty_plus_z_min_r = cap - A.transpose()@dual_vars
    print('Complementary slackness: ', Aty_plus_z_min_r@flows)
```

```
Iteration:  1
Primal Objective:  327.0
Dual objective:    327.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  2
Primal Objective:  321.0
Dual objective:    321.0
Complementary slackness:  0.0
Complementary slackness:  0.0
```

```
Iteration:  3
Primal Objective:  319.0
Dual objective:    319.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  4
Primal Objective:  319.0
Dual objective:    319.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  5
Primal Objective:  315.0
Dual objective:    315.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  6
Primal Objective:  315.0
Dual objective:    315.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  7
Primal Objective:  315.0
Dual objective:    315.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  8
Primal Objective:  315.0
Dual objective:    315.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  9
Primal Objective:  311.0
Dual objective:    311.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  10
Primal Objective:  311.0
Dual objective:    311.0
Complementary slackness:  0.0
Complementary slackness:  0.0
```

```
Iteration:  11
Primal Objective:  311.0
Dual objective:    311.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  12
Primal Objective:  311.0
Dual objective:    311.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  13
Primal Objective:  307.0
Dual objective:    307.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  14
Primal Objective:  307.0
Dual objective:    307.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  15
Primal Objective:  307.0
Dual objective:    307.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  16
Primal Objective:  307.0
Dual objective:    307.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  17
Primal Objective:  306.0
Dual objective:    306.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  18
Primal Objective:  306.0
Dual objective:    306.0
Complementary slackness:  0.0
Complementary slackness:  0.0
```

```
Iteration:  19
Primal Objective:  300.0
Dual objective:    300.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  20
Primal Objective:  296.0
Dual objective:    296.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  21
Primal Objective:  296.0
Dual objective:    296.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  22
Primal Objective:  296.0
Dual objective:    296.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  23
Primal Objective:  296.0
Dual objective:    296.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  24
Primal Objective:  296.0
Dual objective:    296.0
Complementary slackness:  0.0
Complementary slackness:  0.0

Iteration:  25
Primal Objective:  296.0
Dual objective:    296.0
Complementary slackness:  0.0
Complementary slackness:  0.0
```

[18]:
```
# resize final intensities and save as an image
arr = np.reshape(intensity,(h,w)).astype('uint8')
final_image = Image.fromarray(arr)
final_image.save('final_image.png')


print ('elapsed ', toc - tic)
```

```
elapsed  1.6955349445343018
```

[19]:
```
final_image
```

[19]: