Image Segmentation

Ethan Torres
ethanjt2

January 2, 2024

1 Problem Formulation

$$\textbf{min obj } z = |y(i,j) - y(k,l)|$$

$$\textbf{s.t.} \begin{cases} \sum_{(s,j)\in X} x_{s,j} = 1 & \textbf{source node} \\ \sum_{(i,j)\in X} x_{i,j} - \sum_{(j,i)\in X} x_{j,i} = 0 & \textbf{flow conservation} \\ -\sum_{(i,t)\in X} x_{i,t} = -1 & \textbf{terminal node} \\ x_{(i,j)} \geq 0 \ \ \forall (i,j) \in X \end{cases}$$

We then clearly have that $b \in (-1, 0, 1)$ given the domain of i and j in X. We then use this same formulation in the dual problem:

$$\textbf{max obj } b^T p$$

$$\textbf{s.t.} \left\{ A^T p \leq c \right.$$

where the constraint A is the same, but transposed, and the values of c, which is the absolute difference between the intensities of the pixels, becomes the new b represented by c. Using the code that I had this calculation could be explicitly verified both of which returned the optimal value of 317. The complementary slackness value is also printed out step by step to verify.

2 Code Explanation

The code was quite clear to follow (if you were good at python – which I was not so it took my quite some time) but once you understood it, it made sense. The image gets opened and the picture gets split up into what is effectively a grid with each pixel being at the center. This pixel can be thought of as a node and the code is as follows:

$$\begin{aligned} h &= im.height \\ w &= im.width \\ intensity &= list(im.getdata()) \\ \textbf{def idx}(i,j): & \\ \textbf{return } & w * (i-1) + j \end{aligned}$$

This introduces the indexing which is later assigned in my code to coords and nodes, then ngbrs. After that, the edges are then drawn to be the digraph using the networkx package:

$$G = nx.DiGraph()$$
$$G.add - nodes - from(nodes)$$
$$G.add - edges - from(edges)$$

Then I create my incidence matrix, which is the flow conservation as well as the source to the terminal node traversal as represented by a large matrix filled with 0s, 1s, and -1s. This correctly maps out the A matrix, which is the constraint the the primal must abide by. Finally, a correct weight function must be written in that takes into account the objective function:

$$\textbf{def get-weight}(before - node, last - node):$$
$$\textbf{if } last - node >= t:$$
$$\textbf{return} 0$$
$$\textbf{else}:$$
$$\textbf{return } np.abs(intensity[before - node - 1] - intensity[last - node - 1])$$

We then define our f and our b:

$$\mathbf{f} = m.addMVar(shape = G.number - of - edges(),$$
$$vtype = GRB.CONTINUOUS, lb = 0, name = "")$$
$$\mathbf{b} = np.zeros(G.number - of - nodes())$$
$$\mathbf{b}[\textbf{idx}(1, 50) - 1] = -1$$
$$\mathbf{b}[-1] = 1$$
$$\mathbf{c} = np.zeros(G.number - of - edges())$$
$$\textbf{for i in range}(G.number - of - edges()):$$
$$\mathbf{c}[i] = \textbf{get-weight}(edges[i][0], edges[i][1])$$

Finally, we solve the matrix using the Gurobi package:

$$\mathbf{obj} = \mathbf{c}@\mathbf{f}$$
$$m.addConstr(\mathbf{A}@\mathbf{f} == \mathbf{b})$$
$$m.setObjective(obj, GRB.MINIMIZE)$$
$$m.optimize()$$
$$\mathbf{flows} = m.getAttr("X", m.getVars())$$
$$\textbf{print}("Primalobjective:", m.getObjective().getValue())$$

This calls on the function using the formulation we created. We make sure to use GRB.Minimize to tell Gurobi to perform the minimization. We then print the solution at each stage of the 25 iterations. We then need to cut each of these solutions out to actually patch the image. This is then done by

writing a piece of code that deletes each of the solutions so that we can properly cut the image:

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**]
$\mathbf{w} = \mathbf{w} - 1$

The dual is solved in the same manner.