

1. Out put images `q1_Superpixels.png` and `q1_result.png` are shown in Figure 1 and 2 correspondingly. This super-pixel covers a part of two yellow peppers of the lower middle of the input image.



Figure 1: Superpixels of Question 1

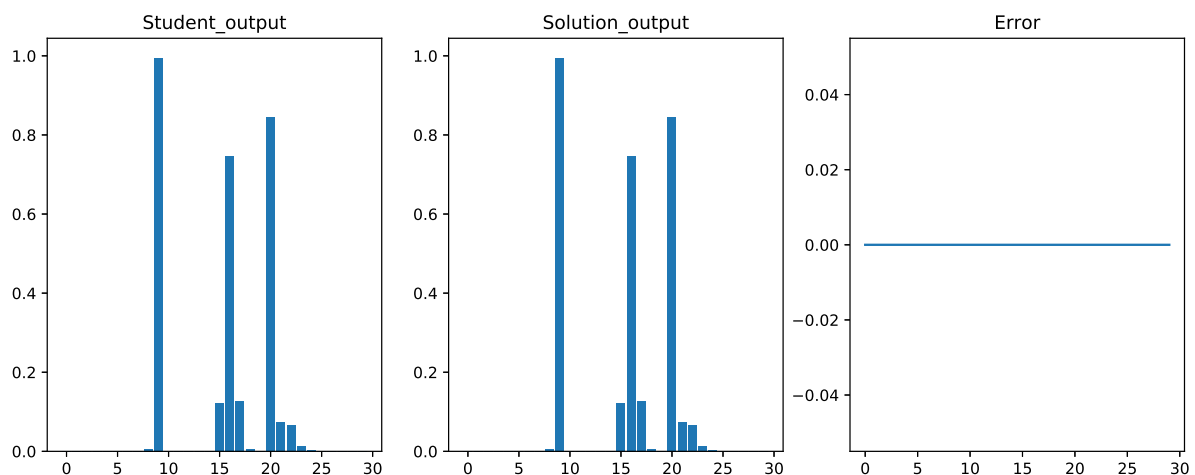


Figure 2: Results of Question 1

2. (a) Out put images `q2_Superpixels.png` and `q2_result.png` are shown in Figure 3 and 4 correspondingly.

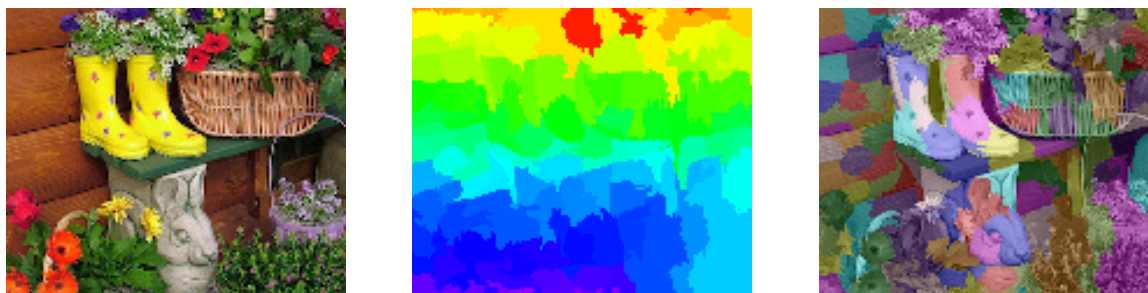


Figure 3: Superpixels of Question 2

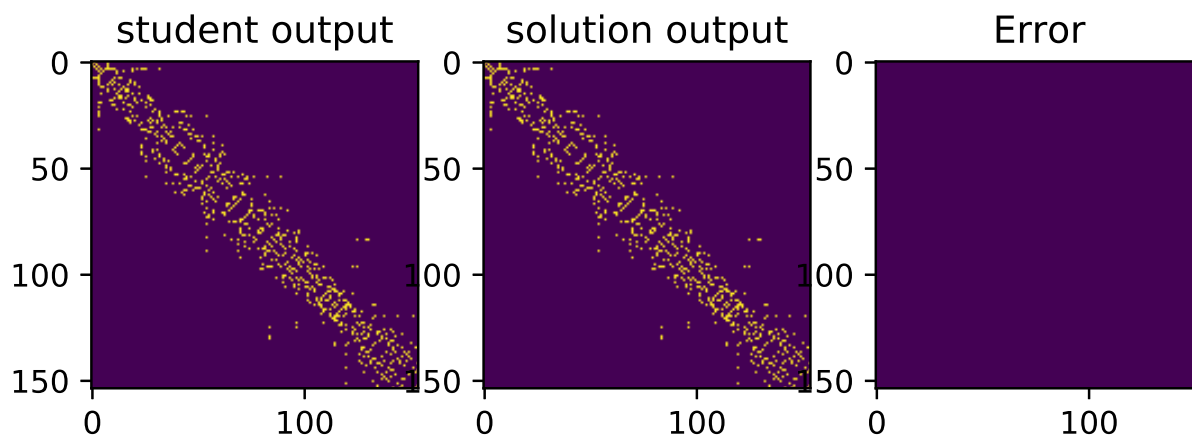


Figure 4: Results of Question 2

- (b) We find the average node degree is 5.33766 for the adjacency matrix in this problem.
- (c) The adjacency graph is not perfectly banded diagonal matrix because superpixels in images have various sizes. We can see that some superpixels are cover a lot of space along the y direction, while narrow along x direction. In this way, it is able to adjunct to some superpixel, whose centroid is far from its centroid. Moreover, some superpixel are relatively small and only adjunct to four superpixels around it.
3. (a) Out put images q3_Superpixels.png and q3_result.png are shown in Figure 5 and 6 correspondingly.

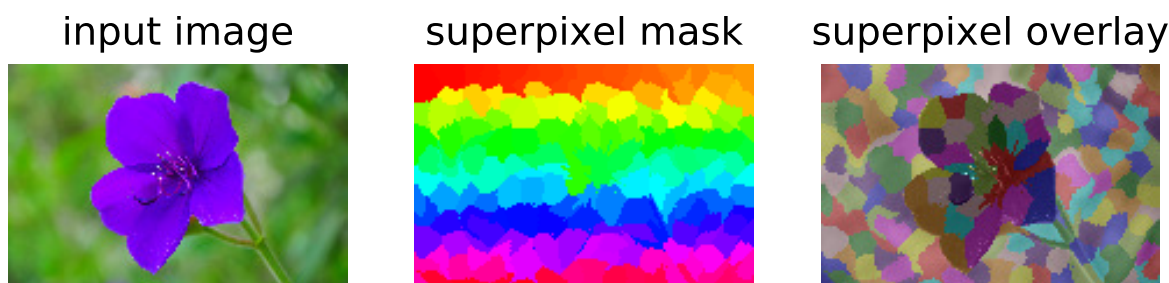


Figure 5: Superpixels of Question 3

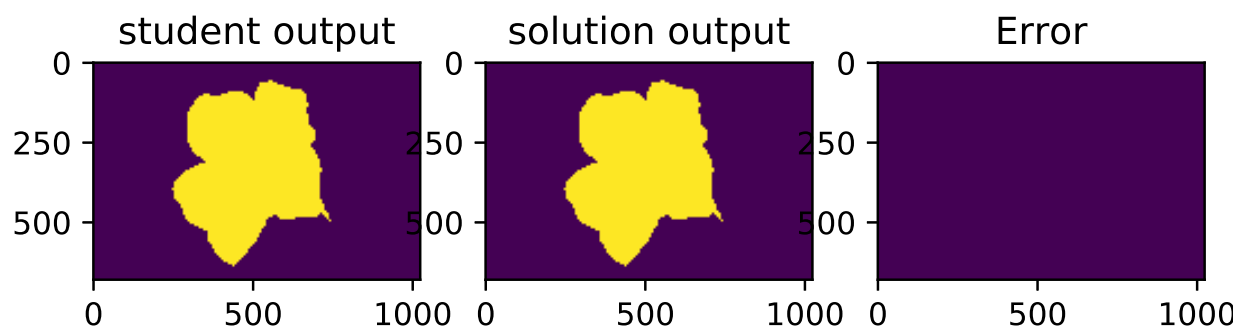


Figure 6: Results of Question 3

- (b) We have shown Adjacency matrix and capacity image in the figure 7. We know that adjacency matrix is an unweighted bidirected graph, and capacity matrix is an weighted bidirected graph. Capacity image has two extra nodes, source and sink than the adjacency matrix. All edges in adjacency matrix are preserved in capacity matrix, and source and sink are connected to all nodes besides each other.

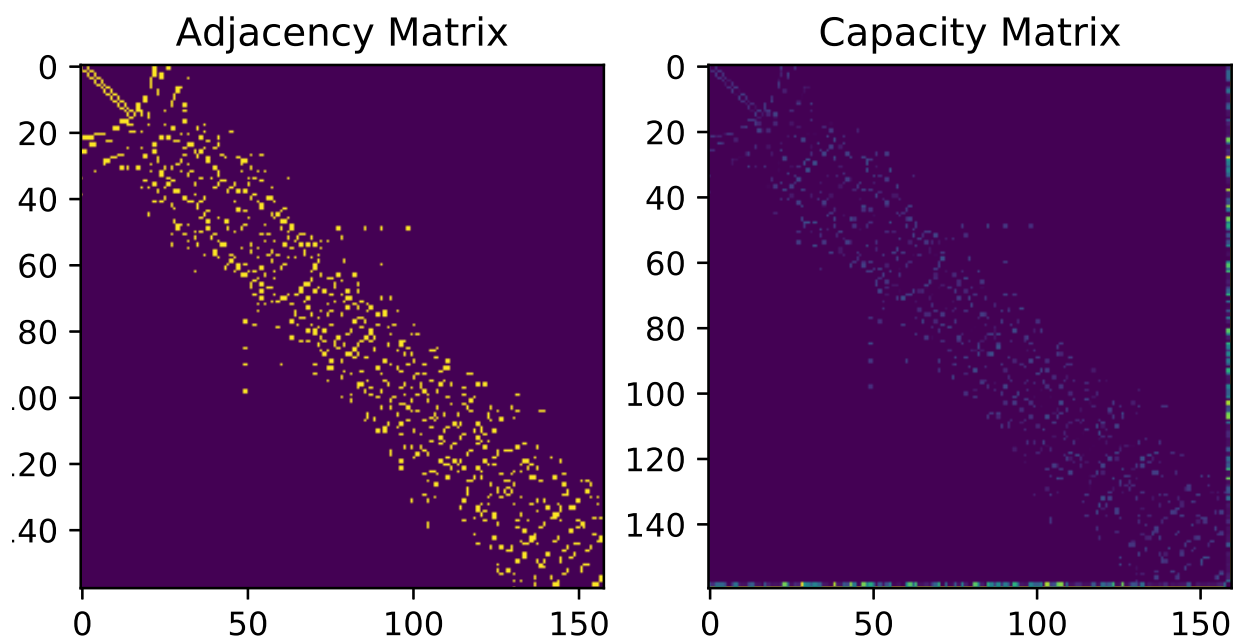


Figure 7: Capacity Image of Question 3

- (c)
4. (a) Flower segmentation is shown in Figure 8 and 9.

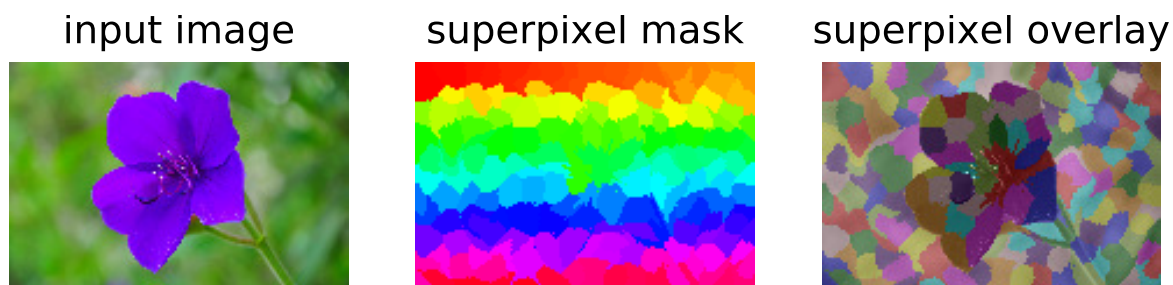


Figure 8: Superpixels of Question 4a

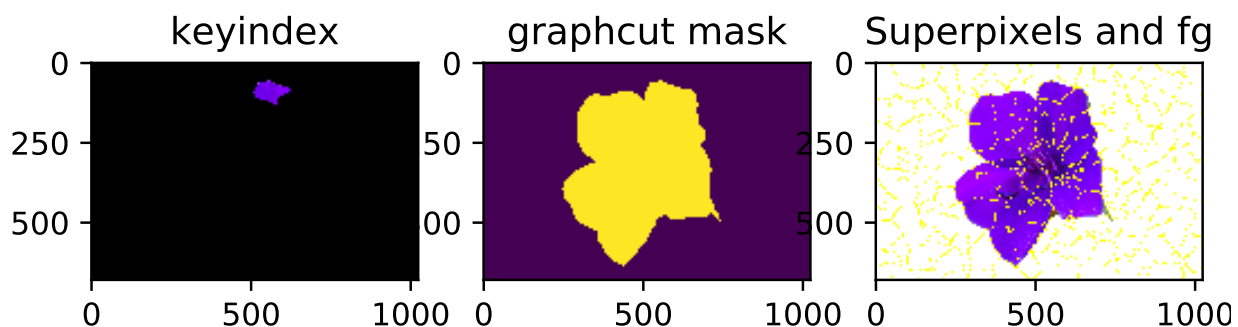


Figure 9: Results of Question 4a

- (b) Flag segmentation is shown in Figure 10 and 11. We can see that if we click on a red stripe, we are not able to segment all red stripes in the images.

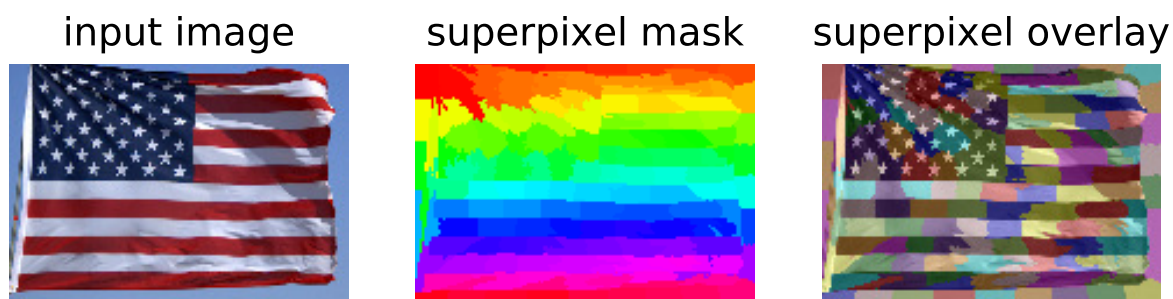


Figure 10: Superpixels of Question 4b

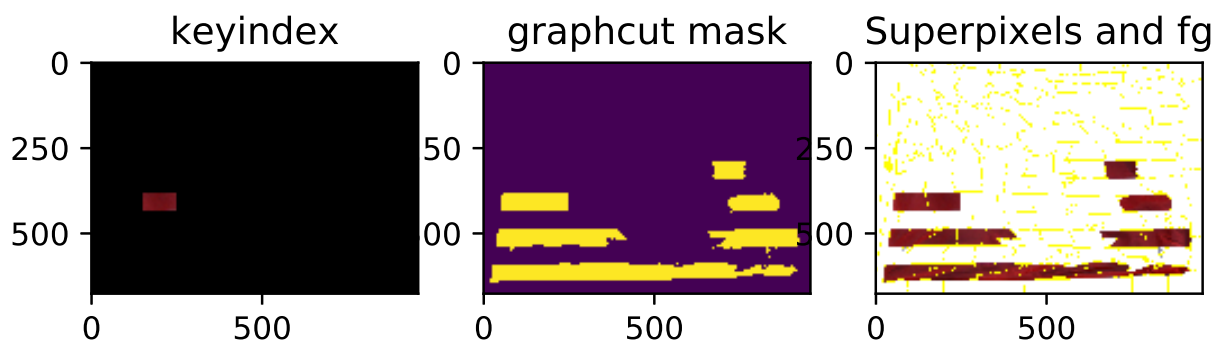


Figure 11: Results of Question 4b

- (c) Porch segmentation is shown in Figure 12 and 13. It is clear that we can segment two boots perfectly, since the color of both boots are quite constant, and it is unique than other section in the image.



Figure 12: Superpixels of Question 4c

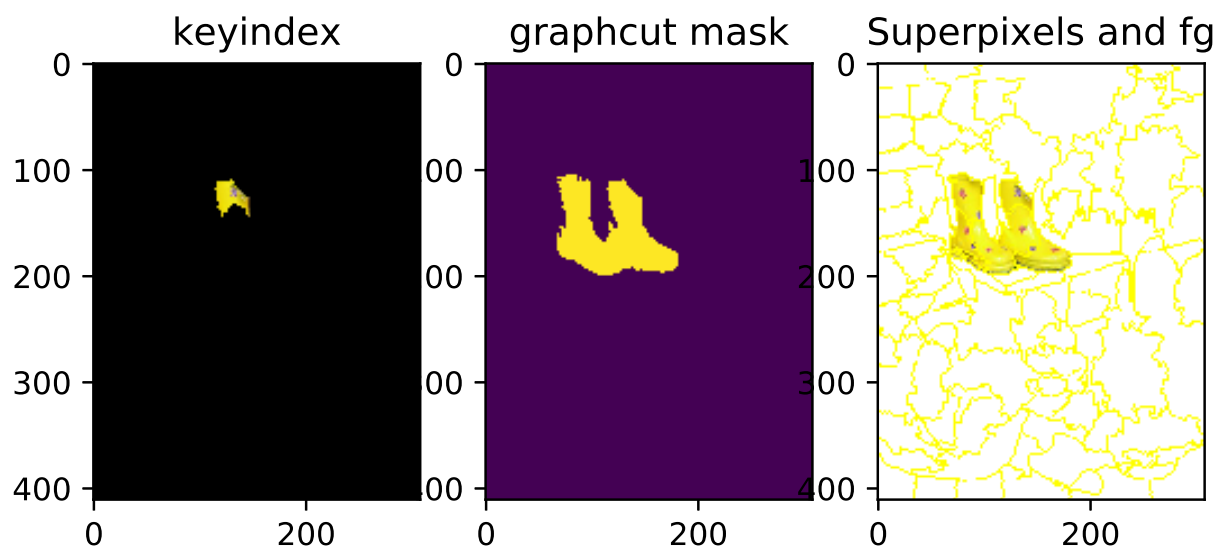


Figure 13: Results of Question 4c

(d) If we try to segment the basket of porch image, the result is shown in Figure 14.

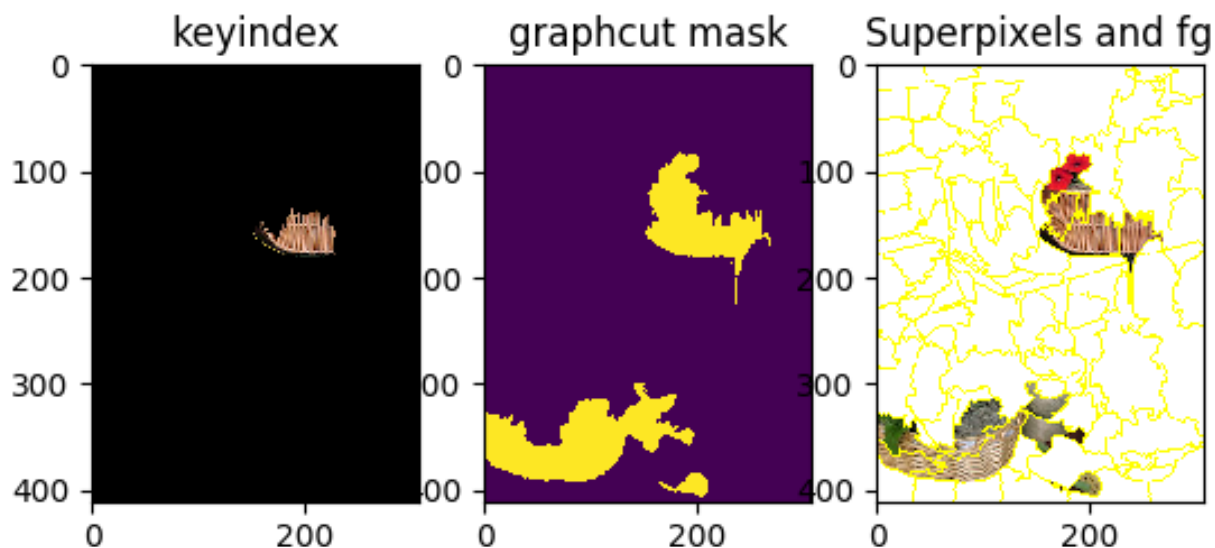


Figure 14: Results of Question 4c

It is clear that we are not able to segment either of the baskets completely. This is caused by the perceptual similarity of two basket even if they are spatially away from each other. One way to solve this is to make sure that all segmented section are connected in adjacency graph. In this way we can separate two similar items but are not spatially connected.

Appendix

Q1:

```
1 def histvec(img,mask,b):
2     '''
3     Function to find the color histogram of the image.
4
5     Args:
6     ----
7     img: input image
8     mask: Super pixel mask. Each pixel location will have the
9           superpixel label corresponding to it
10    b: number of bins in the histogram
11    Return:
12    -----
13    hist_vector: 1-D vector having the histogram of all three
14                  channels appended
15    '''
16
17    img_in_SP = img[mask,:].astype(dtype=np.int64)
18    total_location = img_in_SP.shape[0]
19
20    hist_vector = np.zeros(3*b)
21    ub_unit = 256.0/b
22
23    '''loop through all bins'''
24    for i in range(b):
25        ub_cur = ((i+1)*ub_unit)
26        '''loop through rgb channels'''
27        for j in range(3):
28            cur_idx = np.argwhere(img_in_SP[:,j]<=ub_cur)
29            hist_vector[j*b+i]+=len(cur_idx)
30            img_in_SP[cur_idx,j] = 300
31
32    '''Normalize Histogram'''
33    hist_vector=hist_vector/total_location
34    return hist_vector
```


Q2:

```
1 def seg_neighbor(svMap):
2     '''
3     Function to find adjacency matrix
4     Args:
5     ----
6     svMap: Super pixel mask. Each pixel location will have the
7           superpixel label corresponding to it.
8
9     Return:
10    -----
11    Bmap: a binary adjacency matrix NxN (N being the number of
12          superpixels in svMap).
13    '''
14    segmentList = np.unique(svMap)
15    segmentNum = segmentList.shape[0]
16    # FILL IN THE CODE HERE to calculate the adjacency
17    Bmap = np.zeros([segmentNum, segmentNum])
18    height,width = svMap.shape
19    for i in range(height):
20        for j in range(width):
21            '''check eight connectivity'''
22            y_u = min(i+1, height-1)
23            x_u = min(j+1, width-1)
24            x_l = max(j-1, 0)
25            ''' check lower'''
26            if svMap[i,j] != svMap[y_u,j]:
27                Bmap[svMap[i,j],svMap[y_u,j]] = 1
28                Bmap[svMap[y_u,j],svMap[i,j]] = 1
29            ''' check left'''
30            if svMap[i,j] != svMap[i,x_u]:
31                Bmap[svMap[i,j],svMap[i,x_u]] = 1
32                Bmap[svMap[i,x_u],svMap[i,j]] = 1
33            ''' check lower left'''
34            if svMap[i,j] != svMap[y_u,x_u]:
35                Bmap[svMap[i,j],svMap[y_u,x_u]] = 1
36                Bmap[svMap[y_u,x_u],svMap[i,j]] = 1
37            ''' check lower right'''
38            if svMap[i,j] != svMap[y_u,x_l]:
39                Bmap[svMap[i,j],svMap[y_u,x_l]] = 1
40                Bmap[svMap[y_u,x_l],svMap[i,j]] = 1
41
42    return Bmap
```



```
1 def ave_deg(adj_mat):
2     '''
3     Calculate average node of an adj_mat
4     '''
5     total_deg = np.sum(adj_mat)
6     num_nodes = adj_mat.shape[0]
7     return total_deg/num_nodes
```

Q3:

```
1 def graphcut(S,C,hist_values, keyindex, plt_img=False):
2
3     dnorm = 2*np.square(np.prod(np.divide(S.shape,2)))
4
5     k = len(C)
6     # Generate capacity matrix
7     capacity = np.zeros((k+2,k+2)) # initialize the zero-valued
        capacity matrix
8     source = k # set the index of the source node
9     sink = k+1 # set the index of the sink node
10
11     # FILL IN CODE HERE to generate the capacity matrix using the
        description above.
12     capacity[k+1,keyindex]=0
13     capacity[keyindex,k+1]=0
14     capacity[k,keyindex]=3
15     capacity[keyindex,k]=3
16     for i in range(k):
17         for j in range((i+1),k):
18             His_sim_cur = hist_intersect(hist_values[i],
                hist_values[j])
19             dis_cen = np.array(C[i])-np.array(C[j])
20             Spa_sim_cur = np.exp(-1*np.linalg.norm(dis_cen, ord=2)
                /dnorm)
21             Capa_cur = His_sim_cur*Spa_sim_cur
22             if adjacency[i,j]==1:
23                 capacity[i,j] = 0.25*Capa_cur
24                 capacity[j,i] = 0.25*Capa_cur
25             if i==keyindex:
26                 capacity[k,j] = Capa_cur
27                 capacity[j,k] = Capa_cur
28                 capacity[k+1,j] = 3-Capa_cur
29                 capacity[j,k+1] = 3-Capa_cur
30             elif j==keyindex:
```

```

31         capacity[k,i] = Capa_cur
32         capacity[i,k] = Capa_cur
33         capacity[k+1,i] = 3-Capa_cur
34         capacity[i,k+1] = 3-Capa_cur
35
36     #Compute the cut (this code is provided to you)
37     _,current_flow = ff_max_flow(source, sink, capacity, k+2)
38
39     reachable_node = bfs_residual_reachable(source, current_flow,
40         capacity)
41
42     '''create mask'''
43     B = np.zeros_like(S)
44     for i in range(len(reachable_node)):
45         cur_idx = (S==reachable_node[i])
46         B[cur_idx] = 1
47     return B
48
49 def bfs_residual_reachable(start, current_flow, capacity):
50
51     num_node = capacity.shape[0]-2
52     q = deque([])
53     q.append(start)
54     node_hist = np.zeros(num_node, dtype=bool)
55     visited_node = []
56
57     while len(q) != 0:
58         u = q.popleft()
59         for i in range(num_node):
60             if ~node_hist[i]:
61                 if current_flow[u,i]<capacity[u,i]:
62                     node_hist[i]=True
63                     visited_node.append(i)
64                     q.append(i)
65     return visited_node

```