Machine Learning for Computer Vision

Exercise 6

Kodai Matsuoke, Yuyan Li

June 2

1 Potts MRF with chain structure

We implemented a function which finds a argmin of Potts MRF with chain structure. The code is in Listing 1.

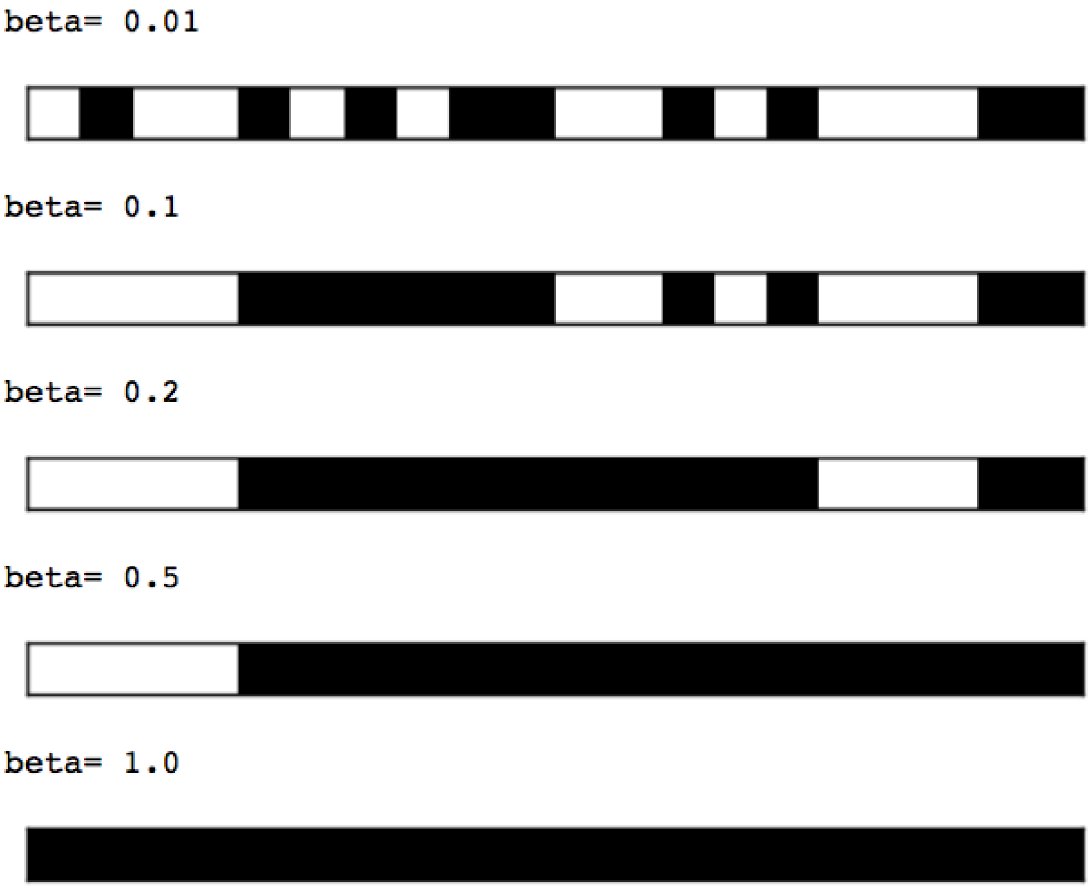
Listing 1: Solver of Potts MRF with chain structure

```
def XOR(ls):
    return sum(ls)%2


def chain_solver(unary, beta):
    l = len(unary)
    e = [(’s’,(0,0),unary[0,0]), (’s’,(0,1),unary[0,1]), ((l
        -1,0),’t’,0), ((l-1,1),’t’,0)]
    G = nx.DiGraph()
    labels = [(0,0),(0,1),(1,0),(1,1)]
    for i in range(l-1):
        for label in labels:
            e.append(((i,label[0]), (i+1,label[1]), unary[i+1,
                label[1]] + beta*XOR(label)))
    G.add_weighted_edges_from(e)
    shortest_path = nx.dijkstra_path(G,’s’,’t’)

    ls =[]
    for i in range(1,l+1):
        ls.append(shortest_path[i][1])

    return np.array(ls)
```



The result for different betas is in Fig ?? We can see from the result that the bigger the beta is, the more pixels are in the same state.
When unary energies are [-1,1], we cannot use Dijkstra Alogorithm. Istead, we can use Bellman-Ford alpgorithm with which we are able to handle negative weights. The other solution for this problem is to set the offset so that all the weights are positive. Since adding scalar to every unary elements does not change the structure, energies between [-1,1] is identical to [0,2].

2 Potts MRF with grid structure

2.1 Implementation

Code to optimize $E_h(X)$ and $E_v(X)$ is in Listing 2.

Listing 2: Solver of Potts MRF with grid structure

```
# A function which optimizes E_h(X)
def E_horizontal(unary, beta):
    height, width, values = unary.shape
    E_h = []
    for i in range(height):
        E_h.append(chain_solver(unary[i,:], beta))

    return np.array(E_h)

# A function which optimizes E_v(X)
def E_vertical(unary, beta):
    height, width, values = unary.shape
    E_v = []
    for i in range(width):
        E_v.append(chain_solver(unary[:,i],beta))

    return np.array(E_v).transpose()
```

The result for different betas is shown below. The bigger the beta is, the more pixels (either in horizontal or vertical direction) are in the same state.

2.2 Interpretation

By definition of E_h and E_v,

$$E_h(\hat{X}) \geq E_h(\hat{X}^h) \quad and \quad E_v(\hat{X}) \geq E_v(\hat{X}^v)$$

are always satisfied. So,

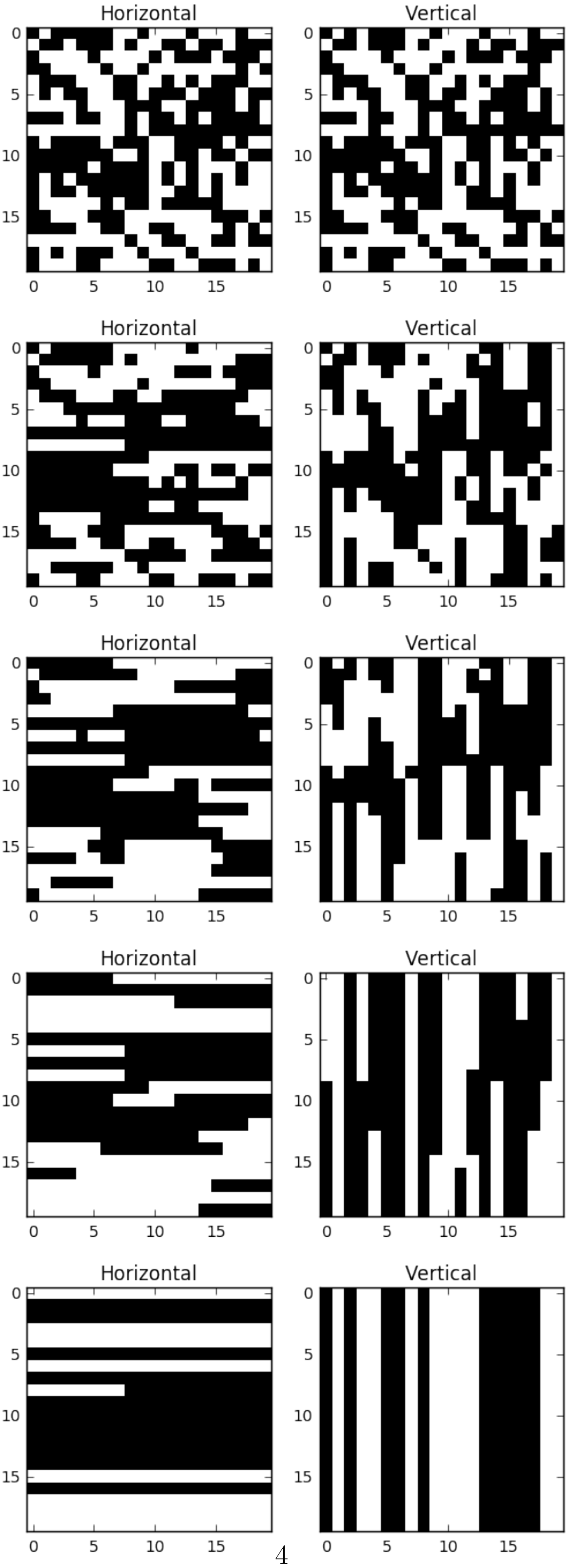
$$E(\hat{X}) = E_h(\hat{X}) + E_v(\hat{X}) \geq E_h(\hat{X}^h) + E_v(\hat{X}^v)$$

is concluded.
When $\hat{X}^h = \hat{X}^v = \hat{X}^{hv}$

$$\forall X, E(X) = E_h(X) + E_v(X) \geq E_h(\hat{X}^h) + E_v(\hat{X}^v) = E(\hat{X}^{hv})$$

This indicates

$$\hat{X}^{hv} = \hat{X}$$


Horizontal
Vertical
0
5
10
15
0
5
10
15