CS6347 Homework 2:

Thennannamalai Malligarjunan – txm230003

Homework 2

Txm230003 - Thenn Annamalai M.

①. MRF for Graph Coloring

1) → Since the graph is a 2n cycle, it always has even number of vertices.

→ For the coloring problem, we know that two adjacent vertices must not be of the same color.

→ If $k < 2$, then with that budget its not possible to color the vertices since its guaranteed that the graph will contain atleast 2 vertices.

2) → given $\phi_i(x_i) = e^{w_{x_i}}$

and $w_a = a$ for all $a \in \{1, \ldots, k\}$.

Ignoring the partition function for now since its a constant.

$P(X_V)$ is ~~given~~ proportional to $\phi_1(x_1) \cdot \phi_2(x_2) \ldots \phi_{2n}(x_{2n})$.

→ It can be observed that

$e^k > e^{k-1} > e^{k-2} > \ldots e^1$ $[\text{and } \phi_i(x_i) = e^{x_i}]$

→ So we can select colors k & $k-1$ alternately.

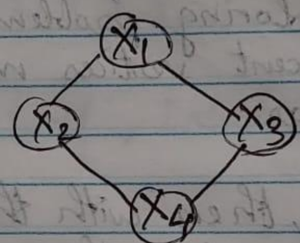
The patterns are,

$[k, k-1, k, k-1, \dots]$

(or) $[k-1, k, k-1, k, \dots]$

3). $n=2,\ k=3$

$G=(4,4)$.



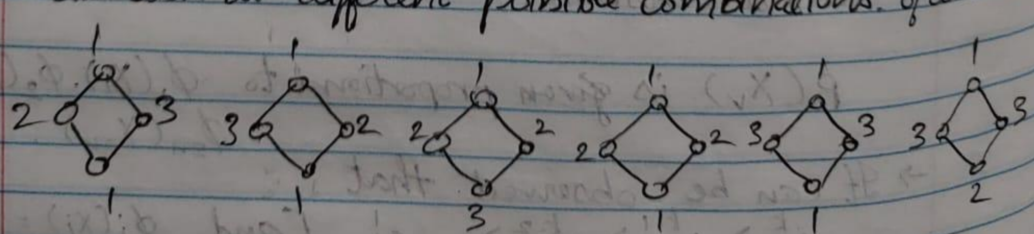
WKT

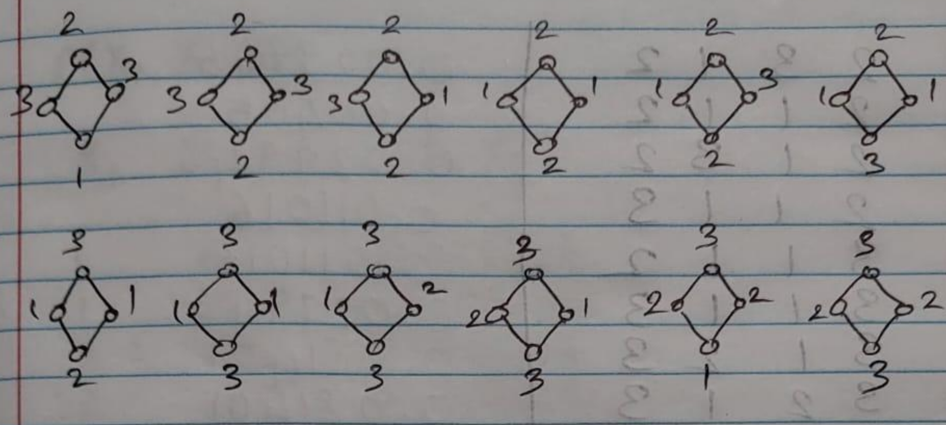
$$P(X_i) = \sum_{X_{V\setminus i}} P(X_V)$$

given $\psi_{ij}(x_i, x_j) = \mathbb{1}_{x_i \neq x_j}$

Finding the joint Probability distribution $P(X_V)$.

Lets look at different possible combinations of colors.





→ The different sum of colors possible are 6, 7, 8, 9, 10.

→ ~~counting~~ Summing over all ~~probab~~ combinations of ϕ functions, we get

$$Z = 6e^{8} + 4e^{7} + 2e^{6} + 4e^{9} + 2e^{10}.$$

$$Z = 99544.40544$$

$P(X_V)$:

X_1	X_2	X_3	X_4	$P(X_1, X_2, X_3, X_4)$
1	2	3	1	0.011016
1	3	2	1	0.011016
1	2	2	3	0.029946
1	2	2	1	0.004052
1	3	3	1	0.029946
1	3	3	2	0.081401
2	3	3	1	0.081401
2	3	3	2	0.221272

2	3	1	2	0.029946
2	1	1	2	0.004052
2	1	3	2	0.029946
2	1	1	3	0.011016
3	1	1	2	0.011016
3	1	1	3	0.029946
3	1	2	3	0.081401
3	2	1	3	0.081401
3	2	2	1	0.029946
3	2	2	3	0.221272

$P(X_1)$:

X_1	$P(X_1)$
1	0.167377
2	0.377633
3	0.454982

$P(X_3)$:

X_3	$P(X_3)$
1	0.167377
2	0.377633
3	0.454982

$P(X_2)$.

X_2	$P(X_2)$
1	0.167377
2	~~0.167677~~ 0.377633
3	0.454982

$P(X_4)$:

X_4	$P(X_4)$
1	0.167377
2	0.377633
3	0.454982

$P(X_1, X_2)$:

X_1	X_2	$P(X_1, X_2)$.
1	2	0.045014
1	3	0.122363
2	1	0.045014
2	3	0.332619
3	1	0.122363
3	2	0.332619

$P(X_2, X_4)$:

X_2	X_4	$P(X_2, X_4)$
1	2	0.045014
1	3	0.122363
2	1	0.045014
2	3	0.332619
3	1	0.122363
3	2	0.332619.

$P(X_1, X_3)$ & $P(X_3, X_4)$ have the same distribution as above.

→ Discussed with Nikhil Manda, Rohan Vishal Rachamadugu

Problem 2:

```
1. # -*- coding: utf-8 -*-

"""Assignment2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1VsRlykELadxKBJnJOAb8ePWikmEq-2Q-
"""

''' Discussed with Nikhil Manda'''

import numpy as np
import math

def psi(x_i, x_j):
  if x_i == x_j:
    return 1
  else:
    return 0

def phi(x_i):
  return math.exp(x_i)

def get_cliques(A):
  cliques = {}
  count = 0
  for i in range(len(A)):
    for j in range(len(A)):
      if A[i][j] == 1 and (i+1,j+1) not in cliques.values() and (j+1,i+1) not in
cliques.values() and i != j:
        cliques[count] = (i+1,j+1)
        count += 1
  return cliques


def get_clique_subset(cliques, node, C):
  cprime = []

  for t in range(len(cliques)):
    if node in cliques[t]:
      cprime.append(t)

  if C in cprime:
```

```
    cprime.remove(C)

  return cprime

# Calculate the normalized beliefs for each x_i (color) and each i (node)
def compute_beliefs(A, n, w, message_ci):
  # Create beliefs matrix
  beliefs = np.zeros((n,len(w)))

  # Calculate the beliefs using the equation
  for i in range(n):
    for x_i in range(len(w)):
      curr_prod = phi(w[x_i])
      for k in range(n):
        if A[k][i] == 1:
          curr_prod *= message_ci[x_i][k][i]
      beliefs[i][x_i] = curr_prod

  # Normalize the beliefs
  for i in range(n):
    curr_sum = sum(beliefs[i])
    for x_i in range(len(w)):
      if beliefs[i][x_i] != 0:
        beliefs[i][x_i] /= curr_sum
  return beliefs

# Calculate the normalized pairwise beliefs for each x_, x_j and each i,j
def compute_pairwise_beliefs(A, n, w, message_ci):
  # Create pairwise beliefs matrix
  pairwise_beliefs = np.zeros((n,n,len(w),len(w)))

  # Calculate the pairwise beliefs
  for i in range(n):
    for j in range(n):
      for x_i in range(len(w)):
        for x_j in range(len(w)):
          curr_prod = 1
          curr_prod *= phi(w[x_i]) * phi(w[x_j]) * psi(x_i,x_j)

          for k in range(n):
            if k != j and A[k][i] == 1:
              curr_prod *= message_ci[x_i][k][i]

          for k in range(n):
            if k != i and A[k][j] == 1:
```

```
              curr_prod *= message_ci[x_j][k][j]

          pairwise_beliefs[i][j][x_i][x_j] = curr_prod

  # Normalize the pairwise beliefs
  for i in range(n):
    for j in range(n):
      curr_sum = np.sum(pairwise_beliefs[i][j])
      for x_i in range(len(w)):
        for x_j in range(len(w)):
          if pairwise_beliefs[i][j][x_i][x_j] != 0:
            pairwise_beliefs[i][j][x_i][x_j] /= curr_sum

  return pairwise_beliefs

# to find Z
def compute_bethe_free_energy(beliefs, pairwise_beliefs, n, w, A):
  hi_sum = 0
  for i in range(n):
    for x_i in range(len(w)):
      hi_sum += math.log(beliefs[i][x_i] * beliefs[i][x_i])

  ij_sum = 0
  visited = []
  for i in range(n):
    for j in range(n):
      if (j,i) not in visited and A[i][j] == 1:
        for x_i in range(len(w)):
          for x_j in range(len(w)):
            if x_i != x_j:
              log_frac = pairwise_beliefs[i][j][x_i][x_j] / (beliefs[i][x_i] *
beliefs[j][x_j])
              ij_sum += math.log(log_frac ** pairwise_beliefs[i][j][x_i][x_j])
        visited.append((i,j))
  return -(hi_sum + ij_sum)

def update_messages(A, k, cliques, its):
    n = len(A)
    cliques = get_cliques(A)

    message_ic = np.ones((k, n, len(cliques)))
    message_ci = np.ones((k, len(cliques), n))

    norm_ci = np.zeros((len(cliques), n))
    norm_ic = np.zeros((n, len(cliques)))
```

```python
    # updates in time t
    converged = False
    for t in range(1,its+1):
      if converged:
        break
      # update messages from cliques to vertices

      prevmic = message_ic.copy()
      prevmci = message_ci.copy()

      for color in range(1,k+1):
        for c in range(len(cliques)):
          for i in range(n): #i was first node
            sum_prd = 0
            if (i+1) in cliques[c]: # node is present in the clique
              sumover_node = 0
              if cliques[c][0] != i+1:
                sumover_node = cliques[c][0]
              else:
                sumover_node = cliques[c][1]

              for x_j in range(1, k+1):
                  sum_prd += psi(color, x_j) * message_ic[x_j-1][sumover_node-
1][c]

            else:
              for x_i in range(1,k+1):
                for x_j in range(1, k+1):
                    sum_prd += psi(x_i, x_j) * message_ic[x_i-1][cliques[c][0]-
1][c] * message_ic[x_j-1][cliques[c][1]-1][c]

            message_ci[color-1][c][i] = sum_prd

      for c in range(len(cliques)):
        for i in range(n):
          for color in range(k):
            norm_ci[c][i] += message_ci[color][c][i]

      # normalize the messages
      for c in range(len(cliques)):
        for i in range(n):
          for color in range(k):
            message_ci[color][c][i] /= norm_ci[c][i]
```

```
      # udpate messages from vertices to cliques
      for color in range(1,k+1):
        for c in range(len(cliques)):
          for i in range(n):
            prd = 1
            cprime = get_clique_subset(cliques, i+1, c)

            for kc in cprime:
              prd *= message_ci[color-1][kc][i]

            prd *= phi(color)
            message_ic[color-1][i][c] = prd

      for c in range(len(cliques)):
        for i in range(n):
          for color in range(k):
            norm_ic[i][c] += message_ic[color][i][c]

      #normalize the messages
      for c in range(len(cliques)):
        for i in range(n):
          for color in range(k):
            message_ic[color][i][c] /= norm_ic[i][c]


      if(np.allclose(prevmci, message_ci)):
        converged = True
    '''
    print("6.MESSAGE C to I AFTER T ITERATIONS---")
    print(message_ci)
    print("7.MESSAGE I TO C AFTER T ITERATIONS-----")
    print(message_ic)
    '''
    return message_ci

def sumprod(A, w, its):
    k = len(w)
    n = len(A)
    cliques = get_cliques(A)
    # get the updated messages after its iterations
    message_ci = update_messages(A, k, cliques, its)
    # get the calculated beliefs
    beliefs = compute_beliefs(A, n, w, message_ci)
    pairwise_beliefs = compute_pairwise_beliefs(A, n, w, message_ci)
```

```
    bethe_free_energy = compute_bethe_free_energy(beliefs, pairwise_beliefs, n,
w, A)
    return np.exp(bethe_free_energy)

def maxprod(A, w, its):
  n = len(A)
  k = len(w)
  cliques = get_cliques

  # get the updated messages after its iterations
  # Normalize the messages (and beliefs) after every iteration
  message_ci = update_messages(A, k, cliques, its)
  beliefs = compute_beliefs(A, n, w, message_ci)

  # Create and find the maximizing assignment
  maximizing_assignment = np.zeros((n))
  for i in range(n):
    max_vals = np.flatnonzero(beliefs[i] == np.amax(beliefs[i]))
    #print(beliefs[i])
    if len(max_vals) == 1:
      maximizing_assignment[i] = max_vals[0]

  return maximizing_assignment

A = np.array([[0,1,1,0],
              [1,0,0,1],
              [1,0,0,1],
              [0,1,1,0]])

w = [1,2,3] #k = 3
its = 100

Z = sumprod(A, w, its)
max_prod_color_assignment = maxprod(A, w, its)

print("Partition function, Z =", Z)
print("MAP assignment =", max_prod_color_assignment)
```