CS6347 Homework 2:

Thennannamalai Malligarjunan – txm230003

A notebook with writing on it

Description automatically generated

A notebook with writing on it

Description automatically generated

A notebook with writing on it

Description automatically generated

A notebook with writing on it

Description automatically generated

A notebook with writing on it

Description automatically generated

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)))

    # 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)