The Document for Problem 3 & Codes

Introduction

This is the document for describing the data structures used in my program, the major steps and the running result of this problem. For more details, you can refer to my source codes, where I have written comments very very explicitly.

Data Structure

I define two kind of classes: Node and Tree.

Node:

This class contains the following attributes:

index ---- id of the node

neighbors ---- list containing ids of all adjacent nodes

(e.g. for node 2, it's neighbors is [1, 4, 5])

potential ---- array of potential

(e.g. for node 2, it's potential is array([0.1, 0.9]))

dimension ---- dimensionality of the node potential

msgIn ----- The dict reserving the info of the message propogated from neighbor nodes towards this node.The key is the id of neibornode. The value is the value(array) of message.

msgOut ---- The dict reserving the info of the message propogated towards neighbor nodes. The key is the id of neibornode. The value is

the value(array) of message.

And this class contains the following function:

calculateMarginal ---- it calculates marginalization of this node after all messages not changed.

Tree:

The class tree is initialized with the list of all the nodes included.

It contains three function: propagation(), sendMsg(), recvMsg(). They realize the propagations of messages. Their functionality will be illustrated more in next section.

Main Steps:

1. Create the node, define all their neighbors and potentials and ids using the class Node, and use these nodes to form the tree using the class Tree.
2. Initialize messages for all nodes, for message in and message out are both [1,1] (if each node potential is 2 dimension)
3. Start computing messages,

For message out, the equation here is:

Message Out(from Xj to Xi) = $\sum_{Xj}$ Potential of edge $*$

$\prod$ all the message from Xk to Xj

Xk is Xj's neighbor node except Xi

The message out computed in the equation is using

$\prod$ all the message from Xk to Xj, the messages here is using the value of message-in iterated from previous turn

For message in, the message in of Xi received from Xj = message sent out from Xj to Xi

4. Iteration Step3 for p times until all the messages stop changing.

(p is a user-defined constant, here I assign 10 to it, and actually it works)

5. Compute marginal for each node.

The equation is:

Marginal(Xi) = $\frac{1}{Z} * \sum_{Xi}$ Product of all incoming messages

Z is normalization factor, which is the sum of all values for each dimension for the marginal potential of Xi

Result

The marginal potentials from node 1 to node 6 is as the screenshot:

```
r-229-105-25-172:sum-product Micky$ python sum_product.py
The potential of Node 1 is:
[ 0.59098405  0.40901595]

The potential of Node 2 is:
[ 0.10161794  0.89838206]

The potential of Node 3 is:
[ 0.59622558  0.40377442]

The potential of Node 4 is:
[ 0.06575926  0.93424074]

The potential of Node 5 is:
[ 0.56742955  0.43257045]

The potential of Node 6 is:
[ 0.12965594  0.87034406]
```

To simplify,

x1 = [0.591, 0.409], x2 = [0.102, 0.898], x3 = [0.596, 0.404]

x4 = [0.066, 0.934], x5 = [0.567, 0.433], x6 = [0.130, 0.870]