**Main()**

{

Read from the command line parameters of the file names, initial type and infer type.

readInIntervMatrix():

read in intervention matrix and save into intervMatrix

readInActivationMatrix():

if initType == 3:

read in activation matrix and save into combinedMatrix

readInPhosphMatrix():

read in phosphorylation matrix

initialize activation part of combinedMatrix based on initial type from command line, initType (global variable)

if ( initType == 0 )

set the initial value to random 0/1

elif ( initType == 1 )

set the initial value to phosphorylation value

elif ( initType == 2 )

check intervention

if intervention value == -1, set the initial value to 0

elif intervention value == 1, set the initial value to 1

elif intervention value == 0, set the initial value to phosphorylation value

elif( initType == 3 )

do nothing (activation part has been initialized in readInActivationMatrix)

append phosphorylation matrix to combined matrix

readInEdges():

read in edge information, save to edgeList

buildNetwork():

create nodeList

add parent and children to each node

calculateCPTforEachNode()

while true:

calculateJointProbForAllNodes () --> JointProbAll

if not converged (change of JointProbAll > 0.1%)

getInferStates()

else

output combinedMatrixx

output activationMatrix

output CPT for each node

break

}

**Data Structure:**

global variables:

initType 0 : random 1 : copy 2 : copy+interv 3 : user input

inferType 0 : no intervention 1 : with intervention

**Class Node:**

int index

string name

char type

int[] children

int[] parents

double[] CPT1 : only save P(A=1|Pa), P(A=0|Pa)=1-P(A=1|Pa)

calculateCPT(combinedMatrix)

**Class Data:**

int[] interventMatrix

int[] combinedMatrix ( activation + phosphorylation )

double[] inferActivationMatrix

string[] edgeList

Node[] nodeList

calculateCPTforEachNode()

getInferStates()

calculateJointProbForAllNodes()

**Functions in Class Data:**

**Data::calculateCPTforEachNode():**

{ foreach node

Node.calculateCPT(combinedMatrix)

}

**Data::getInferStates()**

{

Foreach case c

Foreach activationNode A

{ inferActivation (c,A)

sample a random number r

if r > inferA set action of A to 0

else set A to 1

update A’s CPT and A’s children’s CPT

}

}

**inferActivation(c, A) [ abstract version ]**

{

if consider intervention ( inferType == 1 )

check intervention of current case and protein

if has intervention

set the infer A according to intervention type

if not

set infer A to P(A|.) = P( A|Pa(A) ) ∏ P(Ac|Pa(Ac) )

while, if Ac has intervention, then set P(Ac|Pa(Ac) ) = 1

if don’t consider intervention ( inferType == 0 )

set infer A to P(A|.) = P( A|Pa(A) ) ∏ P(Ac|Pa(Ac) )

}

**inferActivation (c, A) [ detailed version ]**

{

if ( inferType == 1 )

{

check intervention:

if ( intervention A == 1 )

set inferA to 1

elif ( intervention A == -1 )

set inferA to 0

elif ( intervention A == 0 )

inferA = calculateProbA (c, A)

}

else [ inferType == 0 ]

inferA = calculateProbA (c, A)

}

**calculateProbA (c, A)**

{

get the parent of A

find the value of A and A’s parents in combinedMatrix

lookup the CPT value of A : CPT\_A for A=1, A=0

get the children of A

foreach children Ac

get Ac’s intervention intervAc

if ( inferType == 1 and intervAc != 0 )

set CPT\_Ac to 1 for A=0 and A=1

else

get the parent of Ac

find the value of Ac and Ac’s parent in combinedMatrix

lookup the CPT value of Ac : CPT\_Ac for A=0, A=1

inferA = 1 / { 1 + exp [ log ( P(A=0)/P(A=1) ) ] }

while log ( P(A=0)/P(A=1) ) = log[P(A=0)] - log[P(A=1) ]

log [ P(A=0)] = log [CPT\_A(A=0)] + ∑ log [CPT\_Ac (A=0) ]

log [ P(A=1)] = log [CPT\_A(A=1)] + ∑ log [CPT\_Ac (A=1) ]

return inferA

}

**Data::calculateJointProbForAllNodes()**

{

In combinedMatrix

foreach case

foreach protein

get A’s intervention intervA

if (inferType == 1 and intervA != 0 )

P(A|Pa(A) = 1

else

P(A|Pa(A) = CPT\_A

lg [ JointProbAll ] += lg [ P(A|Pa(A) )

}

**Functions in Class Node:**

**Node::calculateCPT (combinedMatrix)**

{

foreach case

get the node and its parent value in combinedMatrix

count the combination of those column values:

N = numOfparents

If NodeValue == 0

Count0 [ ] ++

else

count1 [ ] ++

CPT1[ ] = count1[ ] / ( count0[ ] + count1[ ] )

}

Note: In code implementation, readinMatrices, readinEdges and buildNetwork are implemented in the constructor of Data class. In the main() code, it only creates an object of Data class.