CS 578B (Fall 2015) - Bareinboim - HW 2

Debasmit Das

October 31, 2015

Problem 1

Part 1

We know that John called. So, we have the evidence that the variable 'John Calls' is 'True'. Now, based on that we have to answer all the possible queries over all the other 4 variables. Since, each of the other variables can take binary values we will have a query table over 2^4 entries. For our analysis we assume that 'true' is equivalent to '1' and 'false' is equivalent '0'
From the Bayesian Network, we know that -

$$P(B, E, A, M, J) = P(B)P(E)P(A|B, E)P(M|A)P(J|A)$$

But, we know that J is 'true'. So we have to find basically

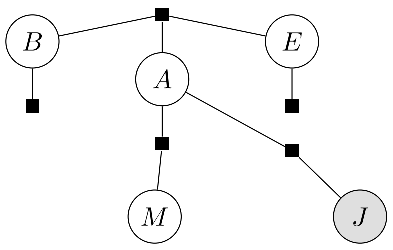
$P(B = b, E = e, A = a, M = m|J = T) = \alpha P(B = b)P(E = e)P(A = a|B = b, E = e)P(M = m|A = a)P(J = 1|A = a)$ for all values a, b, e, m

α is the normalizing constant given by $\sum_{b,e,a,m} P(b, e, a, m, 1)$. From the table, $\alpha = \mathbf{0.0522}$

The probability values have been rounded off to 4 decimal places. The **Brute-Force** table is given below.

B	E	A	M	$P(b, e, a, m, 1)$	$P(b, e, a, m\|J = 1)$
0	0	0	0	**0.0493**	**0.9444**
0	0	0	1	**0.0005**	**0.0096**
0	0	1	0	**0.0003**	**0.0057**
0	0	1	1	**0.0006**	**0.0115**
0	1	0	0	**0.0001**	**0.0019**
0	1	0	1	**0.0000**	**0.0000**
0	1	1	0	**0.0001**	**0.0019**
0	1	1	1	**0.0004**	**0.0077**
1	0	0	0	**0.0000**	**0.0000**
1	0	0	1	**0.0000**	**0.0000**
1	0	1	0	**0.0003**	**0.0057**
1	0	1	1	**0.0006**	**0.0115**
1	1	0	0	**0.0000**	**0.0000**
1	1	0	1	**0.0000**	**0.0000**
1	1	1	0	**0.0000**	**0.0000**
1	1	1	1	**0.0000**	**0.0000**

Now we will use the sum-product algorithm to compute the probability table. First, we need to convert the given graph into its factor graph form. The factor graph with the evidence variable looks as follows



Now, sum-product message passing is applied on this Factor Graph. Each factor represents the local conditional probability in the distribution. Now accordingly message is passed from node to factor and factor to node. Since J has been observed the message from J will be [1 0]. We assume node J to be the root node and M and the other factors to be the leaf nodes. Once message from the leaf node to root node and back to leaf node to root node is carried out, the sum product algorithm is over. Then we multiply all the incoming messages of the factors and take there product to get the joint distribution $P(B, E, A, M, J = 1)$ and then normalize it to get $P(B, E, A, M|J = 1)$ Doing the calculation we get a similar table.

B	E	A	M	$P(b, e, a, m, 1)$	$P(b, e, a, m\|J = 1)$
0	0	0	0	**0.0493**	**0.9444**
0	0	0	1	**0.0005**	**0.0096**
0	0	1	0	**0.0003**	**0.0057**
0	0	1	1	**0.0006**	**0.0115**
0	1	0	0	**0.0001**	**0.0019**
0	1	0	1	**0.0000**	**0.0000**
0	1	1	0	**0.0001**	**0.0019**
0	1	1	1	**0.0004**	**0.0077**
1	0	0	0	**0.0000**	**0.0000**
1	0	0	1	**0.0000**	**0.0000**
1	0	1	0	**0.0003**	**0.0057**
1	0	1	1	**0.0006**	**0.0115**
1	1	0	0	**0.0000**	**0.0000**
1	1	0	1	**0.0000**	**0.0000**
1	1	1	0	**0.0000**	**0.0000**
1	1	1	1	**0.0000**	**0.0000**

Part 2

The BP is run on the ALARM network as instructed in the README. When the running ends we get the following results for the variable 'KinkedTube' and the results are stored in the result.txt file. From that we get -
$P(KinkedTube =' TRUE') = 0.04$ and $P(KinkedTube =' FALSE') = 0.96$, given SaO2 = NORMAL, BP = NORMAL, ArtCO2 = NORMAL, Press = NORMAL and ExpCO2 = LOW

Part 3

We are given a graph G, evidence set as Z and we want to find whether true marginal of a node N exists when belief propagation is applied.

The Graph can be a DAG or be undirected (**Not mentioned in the problem**. So, the graph can be representing BN or MRF. So we need a generalized procedure to check whether BP converges to a true marginal. We will have to convert the given graph G into its corresponding **Factor Graph** and then check for **cycles** in the Factor Graph. If there are cycles in **Factor Graphs** then messages may be sent again and again and then algorithm will not converge in finite number of iterations (We do not apply Loopy BP). In fact, since there is no optimal scheduling for transferring messages convergence to true marginal does not take place So to find whether belief propagation converges to true value we should check whether there are cycles in the Factor Graph. So the Algorithm has the following steps -

- **Creation of a Factor Graph** The Factor Graph is a bipartite graph where one set of nodes constitute the random variables and the other set of nodes constitute the factors. In case of probability distribution, the factors represent conditional probabilities $P(x|Pa(x))$. In the factor graph, nodes are connected to those factors which contain those node variables. Given a Bayesian network G and a set of observed variable Z, it is easy to obtain a factor graph representation of the conditional distribution $P(N|Z)$, by the following general rule is as follows:(i) associate a variable node with each non-observed variable (i.e. each variable in except Z); (ii) for each variable in $Pa(G) - Z$, add a degree 1 function node connected uniquely to that variable; (iii) for each non observed vertex v which is not in $Pa(G)$, add a function node and connect it to v and to all the parents of v (iv) finally, for each observed variable Z, add a function node and connect it to all the parents of Z Please refer [1] for more information on creating factor graphs from Bayesian Networks/Markov Networks.

- **Checking Cycles in a Factor Graph** Now, since the Factor Graph is formed we can check for cycles in the Factor Graph which reduces to finding cycles in an undirected graph. We can apply **Breadth first search** to detect cycles. While applying BFS For every visited vertex v, if there is an adjacent u such that u is already visited and u is not parent of v, then there is a cycle in graph. If we dont find such an adjacent for any vertex, we say that there is no cycle.

Problem 2

We generate Samples using the random sampling methods in Python. Since X_0 ,Y_0 and the e's are Bernoulli we use Binomial(n,1) to generate the sample. For testing independencies we use the G-Square-Test which is available readily as a python module (gsq). Then we apply the IC algorithm and IC* Algorithm respectively.

Part 1

When IC Algorithm is applied, the following structures are obtained for different combinations of n and p. Note, the structures obtained will be different every time because the random samples generated are different.

Figure 1: n=100, p=0 (IC Algorithm)

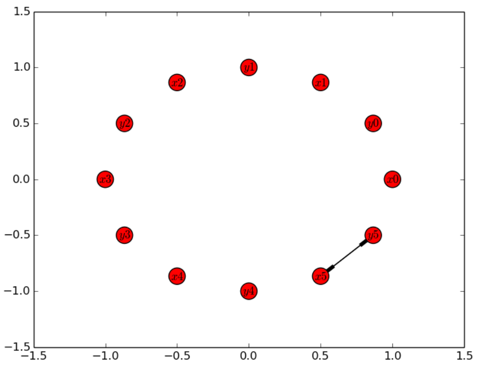
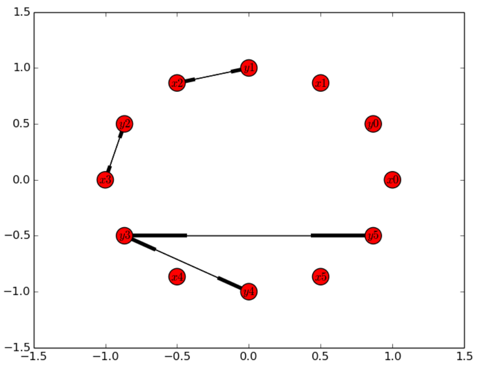


Figure 2: n=100, p=0.05 (IC Algorithm)



Part 2

When IC* Algorithm is applied, the following structures are obtained. Interestingly, the structure does not contain latent variables.

Figure 3: n=100, p=0.2 (IC Algorithm)

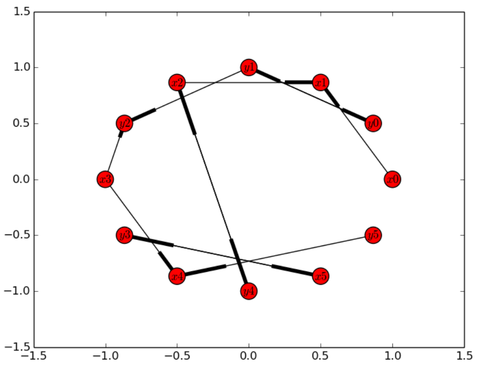
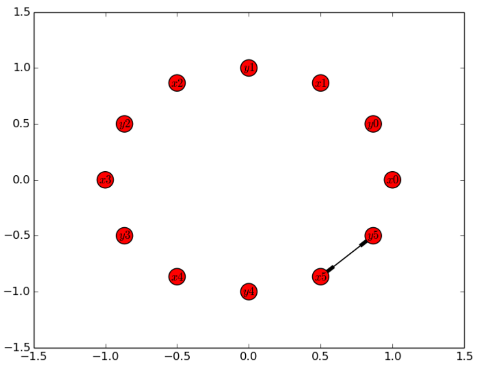


Figure 4: n=1000, p=0 (IC Algorithm)



Part 3

There can be many reasons for the discovered DAG not being same as the data generating model. Firstly, there may be two or more graphs faithful to the same distribution. If we look closely to the figure we can see it is just a reversal of directions in some of the edges of the graph. Moreover, setting α lower or generating more samples might prevent the problem.

Part 4

In the case of $n = 100$ and $p = 0$ the number of samples is very less and therefore the the discovered Graph structure can never match the data generating model. Also, $p = 0$ implies that the disturbances are deterministic and since X_0 and Y_0 are independent this independencies propagate through the graph. So we find very few edges in the discovered structure because every node variable does not have the same distribution.

Figure 5: n=1000, p=0.05 (IC Algorithm)

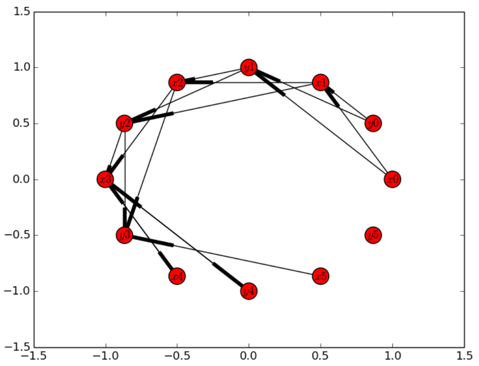
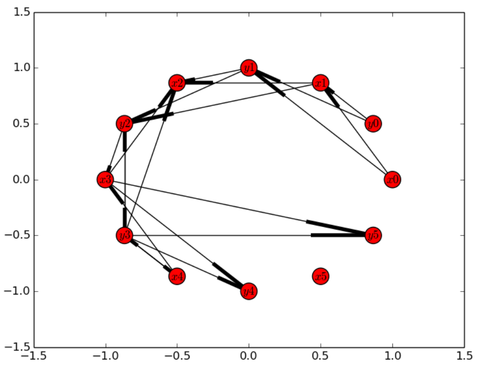


Figure 6: n=1000, p=0.2 (IC Algorithm)



REFERENCES

[1] Lars Schmidt-Thieme, Information Systems and Machine Learning Lab (ISMLL), Institute BW/WI and Institute for Computer Science, University of Hildesheim Course on Bayesian Networks, summer term 2010

[2] Acid, Silvia, and Luis M. De Campos. "An algorithm for finding minimum d-separating sets in belief networks." Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1996.

Figure 7: n=100, p=0 (IC* Algorithm)

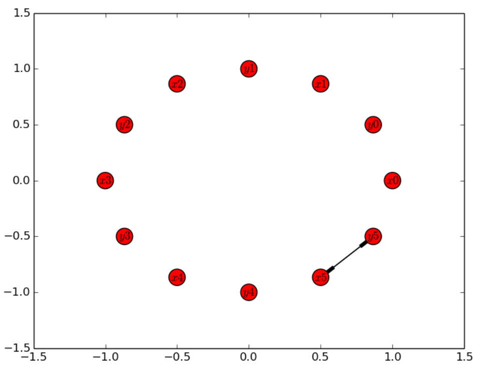


Figure 8: n=100, p=0.05 (IC* Algorithm)

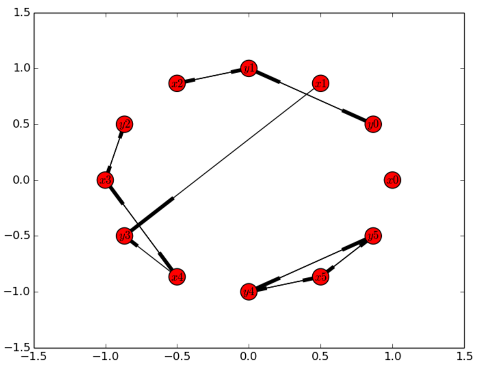


Figure 9: n=100, p=0.2 (IC* Algorithm)

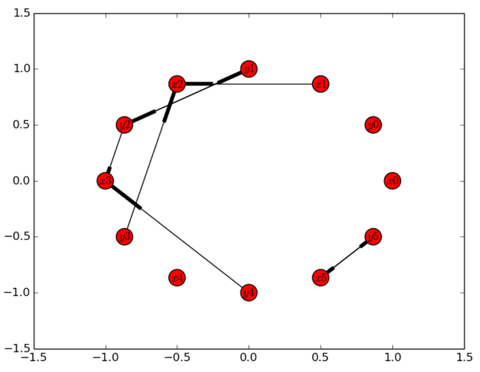


Figure 10: n=1000, p=0 (IC* Algorithm)

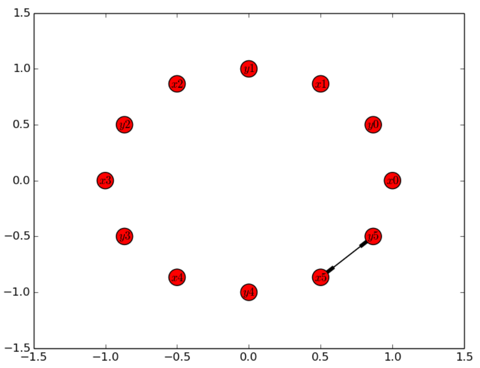


Figure 11: n=1000, p=0.05 (IC* Algorithm)

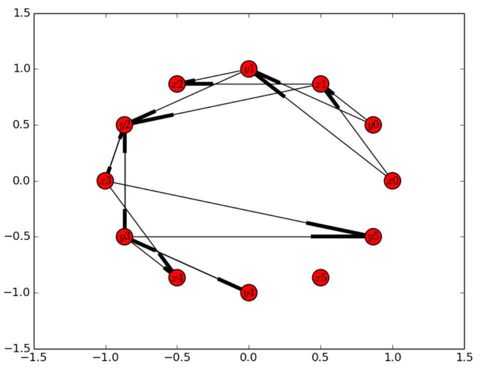
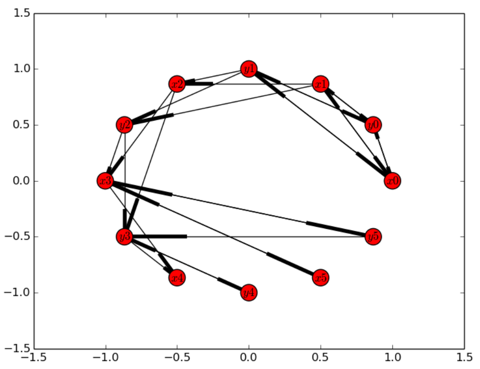


Figure 12: n=1000, p=0.2 (IC* Algorithm)


y1
x1
y0
x0
y5
x5
y4
x4
y3
x3
y2
x2