Learning Structure of Bayesian Belief Networks

Sampada Sonalkar(02M05601),
Babita Sharma(02305402),
[babita, sampadas]@cse.iitb.ac.in

April 30, 2003

1 Problem Statement

Given a dataset the goal is to learn a Bayesian Belief Network [2] that models it.

1.1 Bayesian Belief Networks

A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities. The conditional independence assumptions are represented by a directed acyclic graph(DAG). Each variable in the joint space is represented by a node in the Bayesian network. For each variable two types of information are specified. First, the network arcs represent that the variables are conditionally independent of it's non-descendants in the network given it's immediate predecessors. We say X is a descendant of Y if there is a directed path from Y to X. Second, a conditional probability table is given for each variable describing the probability distribution for that variable given the values of its immediate predecessors.

2 Our Approach

Given a set of training examples, the objective is to learn a Bayesian belief network that best represents the data. The work involves selecting the best graph from a space of 2^{n^2-n} graphs. We make use of genetic algorithm [3] to search for the best graph. The goodness of a graph is computed as a number which is called the fitness value of a graph. The graph with highest fitness value is desired. The working of genetic algorithm in general and their use in solving our problem is explained in the following subsection.

2.1 Genetic Algorithm

Algorithm is started with a set of solutions (represented by chromosomes/genes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness - the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.

2.1.1 Outline of the Basic Genetic Algorithm

The algorithm has the following steps:

1. **Start:** Generate random population of n chromosomes (suitable solutions for the problem)
2. **Fitness:** Evaluate the fitness f(x) of each chromosome x in the population
3. **New population** Create a new population by repeating following steps until the new population is complete
   (a) Selection: Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
   (b) Crossover: With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
   (c) Mutation: With a mutation probability mutate new offspring at each locus (position in chromosome).
   (d) Accepting: Place new offspring in a new population
4. **Replace** Use new generated population for a further run of algorithm
5. **Test** If the end condition is satisfied, stop, and return the best solution in current population
6. **Loop** Go to step 2

The operations such as crossover and mutation are explained in detail in stage 1 report. There is a public domain software called *GENESIS* [**?**] which contains code that implements a generic genetic algorithm.

3 Our Contribution

- **Encoding of a graph as a chromosome**
  We represent the directed acyclic graph of the network as a gene by writing out the rows of its adjacency matrix sequentially. This is illustrated in figure 1.

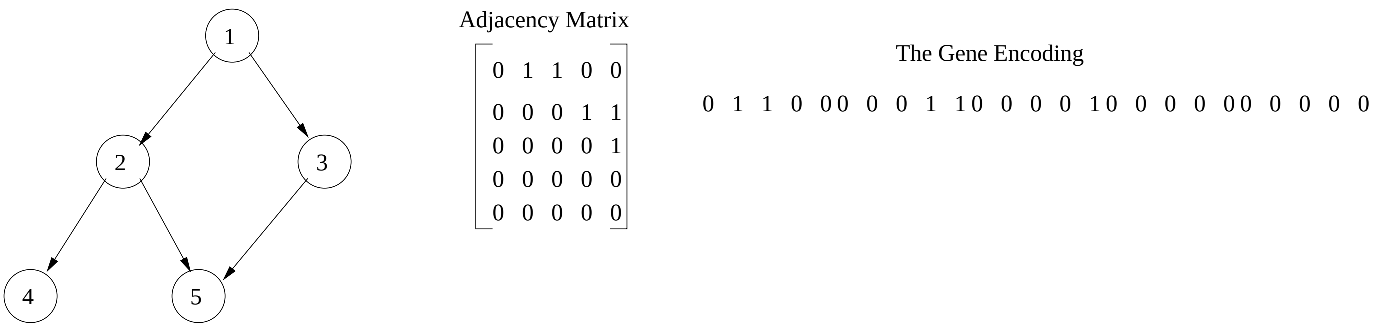


Figure 1: Encoding of a graph as a gene

- **Generating initial pool of genes**
  From the given dataset we determine the number of attributes(columns) in it. This is equal to the number of nodes that the DAG will contain. We put all graphs containing only one edge in the initial pool of chromosomes. This initial pool of genes is put in a

file called *init.dat* from where the genetic algorithm takes its input. There is an option provided by GENESIS to generate the initial set of genes randomly but in our case we have a constraint that the graphs should be acyclic hence we need to be careful in selecting the initial set of graphs. If the user has some domain knowledge about the dataset for which the Bayesian Belief network is being learnt, i.e. if the user is aware of dependencies between attributes then the initial pool can be modified to reflect the dependencies.

We have written a function *InitialPool* that takes the dataset as input, determines the number of attributes it has, generates all single edge graphs and puts them in the *init.dat* file in gene representation as explained above.

- **Cycle detection algorithm**
  In the genetic algorithm new graphs are generated from the old ones by performing crossover and mutation operations. These new graphs should be included in the next generation only if they are acyclic since a bayesian belief network is an acyclic graph. For crossover two genes with higher fitness values are selected. We have written a *detectCycle* function that checks if the input graph contains a cycle and if a cycle is found it assigns a very low value to the graph. In this way the graphs containing cycles are in a way excluded from a population.

- **Fitness function**
  The fitness function plays a central role in the genetic algorithm. The algorithm used in the function is as follows:

  Input : Graph (Bit string - gene)
  Output : Probability of generating test dataset from given graph
  Algorithm:

  1. *If graph has a cycle*
     Return(-1)
  2. *Find Parents of each attribute from graph*
  3. *Compute conditional probability table for each node(attribute) using training dataset*
  4. *Compute probability of seeing test dataset*
     - *Product = 1*
     - *For I = 1 to no of tuples in test dataset*
       *Product = Product * prob. Of generating tuple[I] from BBN*
  5. *Return(Product)*

4 Running the Software

There are 2 main steps in running the software:

1. At the command prompt type
   $*setup*
   Various parameters such as crossover rate, mutation rate, number of iterations,size(length) of each gene, number of best graphs to be saved at the end, etc. can be interactively set using setup.
   Alternately these parameters can be set in a file called *in.dat*.

2. To generate Bayesian Belief Network for a dataset say *temp.txt* type
   $\$ga.eval\ temp.txt\ dat$
   The output graphs in gene form are generated in a file called *min.dat*. They are also generated in a readable format (Eg. Edge from node a to node b) in a file called *edge_list.dat*.

5 Testing

We tested our code on the ALARM [5] dataset which is a standard benchmark dataset to assess the performance of learning algorithms for Bayesian Networks. This dataset consists of 20,000 cases. The dataset was synthetically generated from a Bayesian Network, introduced by Beinlich et al [5], of 37 nodes and 47 arcs. The network and the description of its attributes are shown in figure 2.

The results are presented in the next section.

Figure 2: The Alarm Network

5.1 Results

In this section we present the results of our structure learning method. We also discuss the shortcomings of the method and the improvements to be made.

We evaluated the original ALARM Network with our fitness function using the existing ALARM dataset and obtained a fitness of 0.268707.

We obtained a set of 10 graphs having high fitness values from our method. Two of these graphs are shown below. Their fitness values are 0.31511 and 0.3188 respectively. The number in brackets in output shown below corresponds to the attribute number given in the textual description of the ALARM dataset.

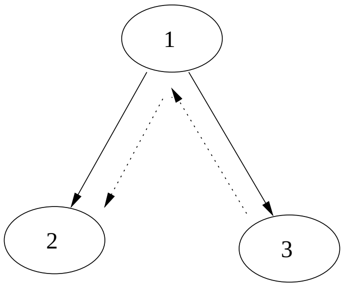
Graph : 0
Edge from 24 (15) to 1 (16)
Edge from 6 (21) to 5 (20)
Edge from 8 (23) to 5 (20)
Edge from 25 (34) to 5 (20)
Edge from 28 (11) to 7 (22)
Edge from 7 (22) to 8 (23)
Edge from 30 (27) to 11 (26)
Edge from 7 (22) to 16 (1)
Edge from 0 (12) to 24 (15)
Edge from 17 (2) to 24 (15)
Edge from 10 (25) to 31 (29)
Edge from 32 (6) to 31 (29)
Edge from 33 (7) to 32 (6)
Edge from 14 (31) to 33 (7)
Edge from 34 (8) to 33 (7)
Edge from 35 (9) to 34 (8)
Edge from 36 (5) to 35 (9)
End of adjacency list

Graph : 1
Edge from 1 (16) to 0 (12)
Edge from 24 (15) to 1 (16)
Edge from 6 (21) to 5 (20)
Edge from 25 (34) to 5 (20)
Edge from 8 (23) to 6 (21)
Edge from 28 (11) to 7 (22)
Edge from 7 (22) to 8 (23)
Edge from 30 (27) to 11 (26)
Edge from 17 (2) to 24 (15)
Edge from 10 (25) to 31 (29)
Edge from 32 (6) to 31 (29)
Edge from 33 (7) to 32 (6)
Edge from 14 (31) to 33 (7)
Edge from 26 (32) to 33 (7)
Edge from 34 (8) to 33 (7)
Edge from 12 (28) to 34 (8)
Edge from 35 (9) to 34 (8)
Edge from 36 (5) to 35 (9)

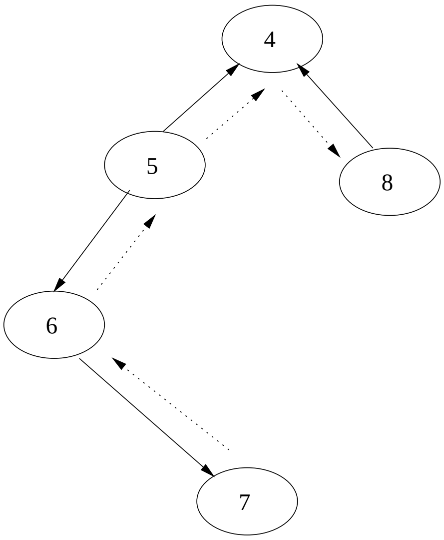
End of adjacency list

Although the dependencies do not directly correspond to the dependencies of the original network, we adopt the approach of Bayes' ball algorithm [?] to evaluate their correctness.

The Bayes' Ball algorithm is useful for identifying conditional irrelevance and requisite information. It identifies the valid dependencies between pairs of nodes that do not have direct dependencies. A few examples of valid dependencies are depicted below:



1-2 Valid Dependency



7-8 Invalid Dependency

Figure 3: Dependencies defined by Bayes Ball Algorithm

5.2 Diagnosis of Shortcomings and Aspects for Improvement

Our algorithm for learning the structure of a Bayesian belief network is in it's infant stages. We will now cite the various reasons due to which our method was not able to output a near exact structure.

- Sparsity of the Dataset
  We had 20,000 records of the ALARM dataset. But given the large number (37) of attributes, these aren't enough to detect all the dependencies of the network.

- Gradient Ascent Approach
  Some of the probability values in the conditional probability tables at the nodes were found to be 0.0 This indicates the sparse nature of the data. If some record in the test data has a combination of parent values and traget values that haven't occured in the training data, the model will output its likelihood as 0.

  Inorder to avoid this, we need to use the gradient ascent approach. Instead of directly computing the probabilty values from the training data, we start with a set of priors and continuously update these based on the training data seen by the algorithm.

- The parameters of the genetic algorithm such as crossover rate, mutation rate, generation gap need to be defined by a trial and error method.
- Introducing cycle detection early in the genetic operations will prevent the faulty genes from even entering the population.
- Introducing penalized log likelihoods can improve the likelihood values.
- Using cross-validation techniques helps to get over the sparsity of the dataset to a certain extent and gives results that are more robust to noise and overfitting.

6 Status of Project

The project is complete.

7 % contribution of each member of the group

There were 8 main stages in solving the problem

1. Understanding the problem
2. Reading about Bayesian belief networks.
3. Reading about Genetic algorithm
4. Understanding *GENESIS* code
5. Implementation of functions in C and integration with the existing code
6. Testing on the *Alarm* dataset
7. Studying the issues involved for improving the results
8. Reporting and presenting the work done

The participation of both the team members was equal in each of the above.

7.1 Total Number of hours spent

The time spent on the course project is approximately 90 hours.

7.2 Acknowledgment

We are extremely grateful to Ganesh Ramakrishnan for explaining to us the problem statement and helping us throughout the project with his guidance.

References

[1] Tom Mitchell: Machine Learning.

[2] David Heckerman: Tutorial on learning with Bayesian Networks. Microsoft Research, 1995.

[3] Mitchell, Melanie: Introduction to Genetic Algorithms. MIT Press, 1996.

[4] Ross D. Shachter: Bayes-Ball: The Rational Pastime (for Determining Irrelevance and Requisite Information in Belief Networks and Influence Diagrams) .

[5] L. Bienlich, H. Suermondt and G. Cooper: The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In Proc. of the second European Conference on Artificial Intelligence in Medicine, pages 247-256, Springer Verlag, 1989.