AA228 – Decision Making Under Uncertainty Project 1

Name: Mandeep Singh SUID: 06406971

Introduction: Bayesian Structure Learning is a method with which we can find the Bayesian network given the data.

Algorithm Implementation:

The algorithm was broken down into below components:

1) Initialize an unconnected graph(**initial_G**) with all the nodes from the given data set.
2) Find m_ijk for graph (**initial_G**). Refer to function ***find_m_ijk(<data>, <Graph>)*** in python code.
3) Compute Bayesian score for the graph (**initial_G**) as given by equation below:

$$\sum_{i=1}^{n}\sum_{j=1}^{q_i}\ln\left(\frac{\Gamma(\alpha_{ij0})}{\Gamma(\alpha_{ij0}+m_{ij0})}\right)+\sum_{k=1}^{r_i}\ln\left(\frac{\Gamma(\alpha_{ijk}+m_{ijk})}{\Gamma(\alpha_{ijk})}\right)$$

   a. As the given prior is uniform so the ln(P(G)) term is same for all iterations so it can be omitted.
   b. The alpha_ijk are '1' again because of uniform prior.
   c. alpha_ij0 and m_ij0 are calculated in this iteration itself.

   Refer to function ***find_bayesian_score(<Graph>, <m_ijk>)*** in python code.

4) For each node in graph(**initial_G**) :
   a. Connect a parent to that node. This becomes graph **G**
      i. Check if graph is cyclic, if not then compute Bayesian score for this new graph(**G**) similar to step 2 & 3.
      ii. If Bayesian score is better than previous recorded score, update the graph (**initial_G)** with new optimal graph(**G**) that will be used for next node iteration.
   b. Once all nodes are traversed the graph(**G**) will be the final optimal graph and Bayesian score of this graph(**G)** will be the final Bayesian score.

   Refer to bottom part of function ***compute(infile, outfile)***where graph iterations are carried out.

find_m_ijk() function:

In this function, for each data sample, parental values of a give node "i" are stored as list (***G.nodes[i]["list_values_at_j"])***) in the "i" node attribute of graph. The indices of the list that stores parental values gives "j". The values of given node 'i' are stored as a list (***G.nodes[i]["value_of_ri"] = []***) in the "i" node attribute of graph. The indices of this list gives "k".

Once for all the data samples the "j" & "k" node information is stored as node attributes. Based on this information we do sample loop iteration(for each sample) to find **m_ijk[index]** which is count of m_ijk[index] based on the "j" & "k" node attributes.

compute(infile, outfile) function:

In the bottom part of this function are the loops that iterate over node list of a given graph and finds the Bayesian score. The 1st loop acts as **Local Directed Graph Search**(algorithm 2.9 from DMU textbook). This loop basically adds a parent to existing best graph available and finds the Bayesian score. If score is better the graph is updated, if not continues.

The 2nd loop and 3rd loop is the **K2Search** (algorithm 2.8 from DMU textbook). The algorithm is described in Step 4 above.

Improvements in the code that must be explored: Have a 4th loop which does **K2Search,** starting at a different node as a starting point. Do this for all nodes in the graph and pick the graph with best Bayesian score.

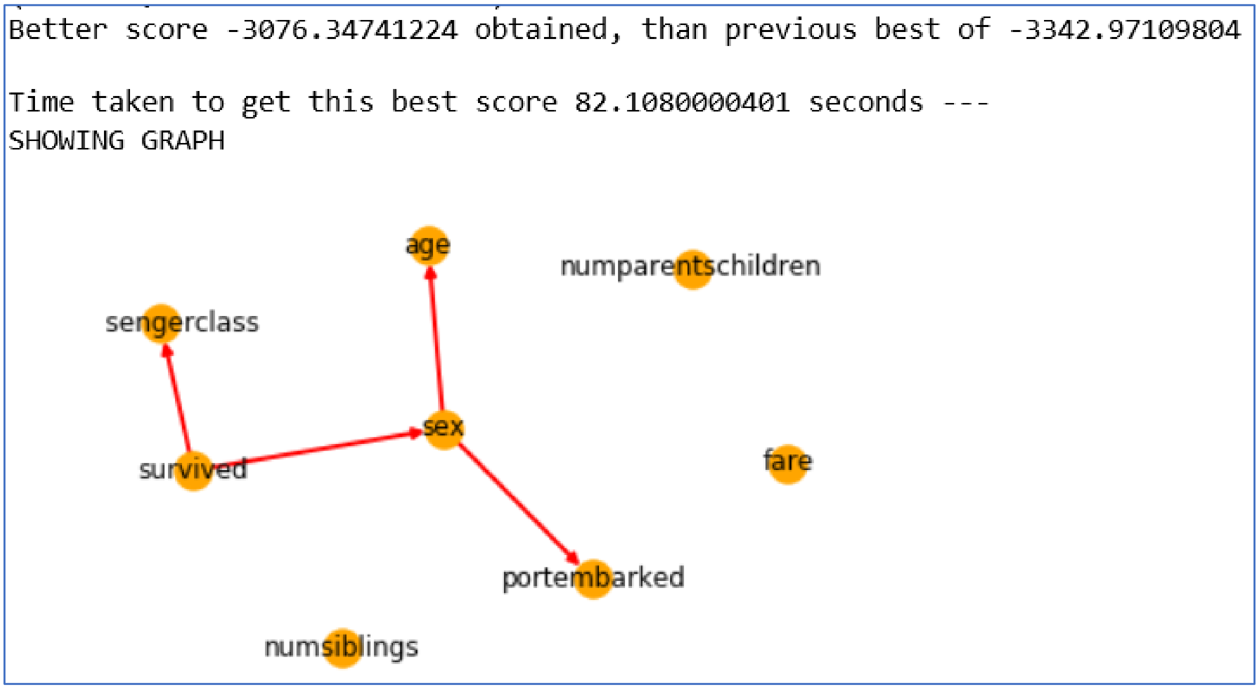
Explore mechanism where m_ijk can be efficiently calculated.

Runtime and Best Bayesian Score Table

Data Set	**Best Bayesian Score**	**Runtime (seconds)**
small.csv	-3076.34	82.10
medium.csv	-43133.74	150.55
Large.csv	-426340.15	16027 and running

Graph Outputs:

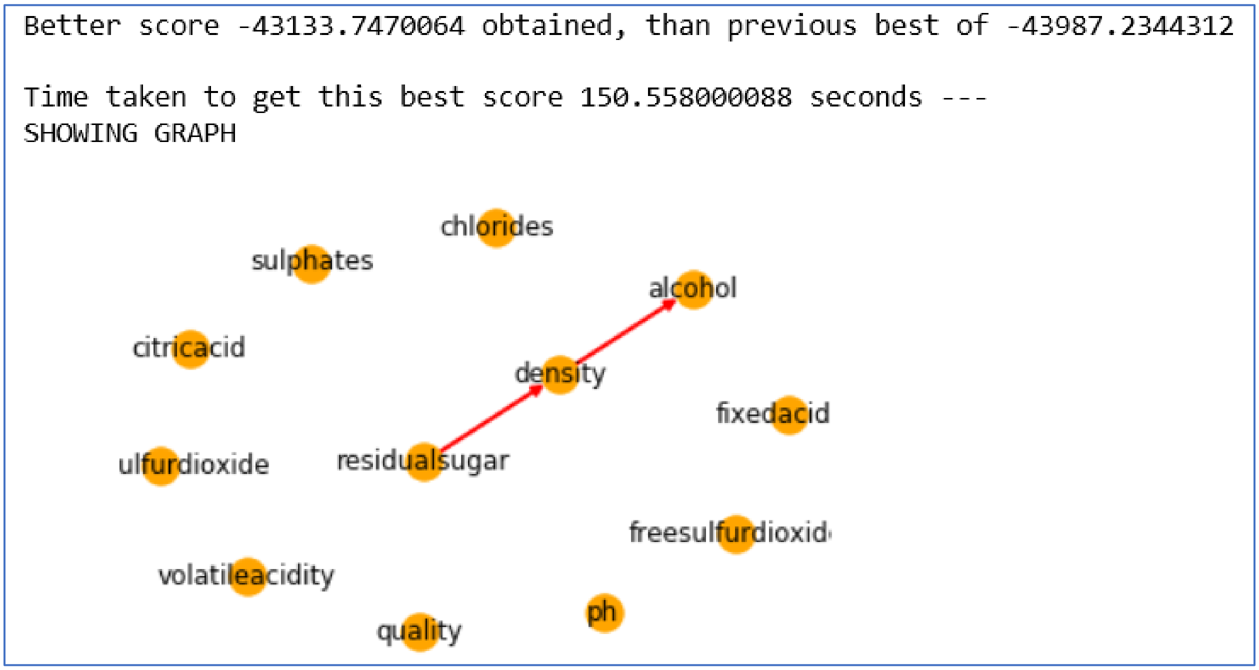
Data Set: small.csv



```
Better score -3076.34741224 obtained, than previous best of -3342.97109804

Time taken to get this best score 82.1080000401 seconds ---
SHOWING GRAPH
```

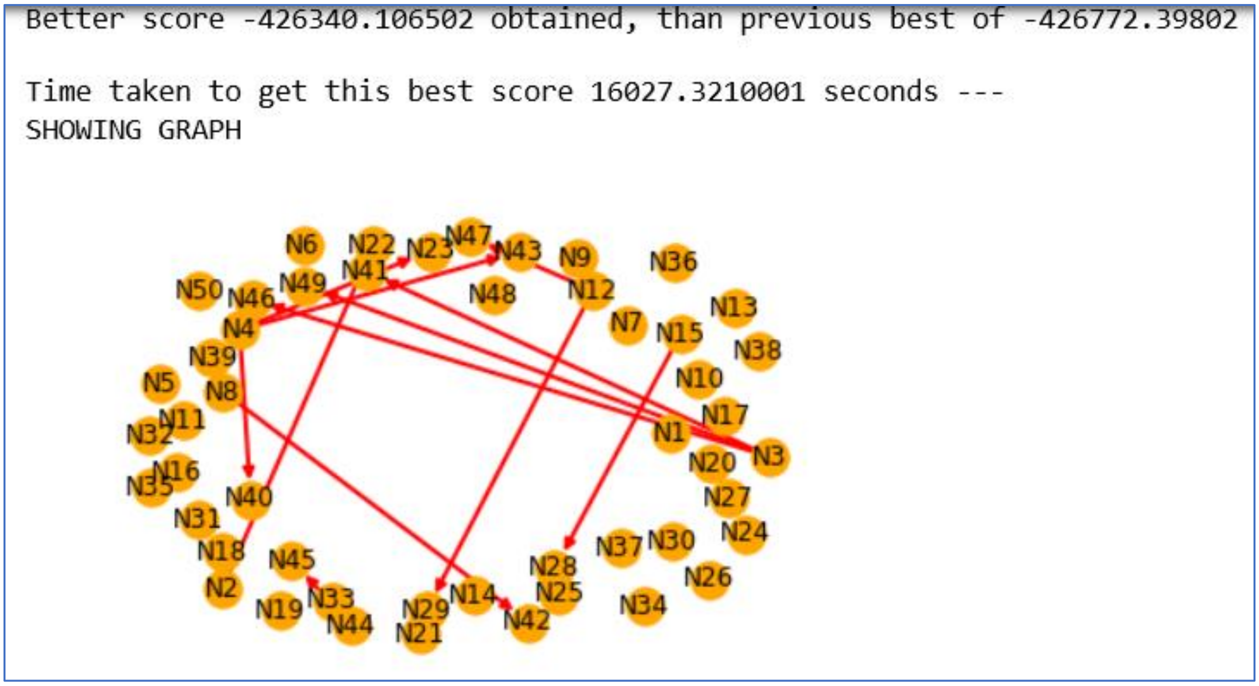
Data Set: medium.csv



```
Better score -43133.7470064 obtained, than previous best of -43987.2344312

Time taken to get this best score 150.558000088 seconds ---
SHOWING GRAPH
```

DataSet: large.csv



```
Better score -426340.106502 obtained, than previous best of -426772.39802

Time taken to get this best score 16027.3210001 seconds ---
SHOWING GRAPH
```