Project 1 - Bayesian Structure Learning

Ken Oung Yong Quan

Computing The Bayesian Score

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

The Bayesian Score is computed using formula 2.83 as given in the textbook. Since we assume the priors to be all 1s, we can reduce the equation to the following, where r_i refers to the number of possible instantiations of the ith node.

$$\ln P(G|D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \ln(\frac{\Gamma(r_i)}{\Gamma(r_i + N)}) + \sum_{k=1}^{r_i} \ln(\frac{\Gamma(1 + m_{ijk})}{\Gamma(1)})$$

We can observe that the computation of the score requires you to loop over each node. For every node i, we compute a local score based on the number of parental instantiations q_i and the distributions of the data for each parental instantiation. Keeping this in mind helps us to reduce computational time significantly, which was something I found out the hard way.

Vanilla K2 Algorithm

The original K2 algorithm focuses on optimizing one node at a time. For every node, we greedily add the best parent until the score cannot be improved by adding another parent. Since we're only adding parents to the node, only the local score for this node would be affected.

In order to reduce compute time, I computed the scores for each node separately, then summed over these values to obtain the bayesian score for the graph. When looping over each node, I would only compute the new score for that node, update its corresponding value in the array, then take another sum to get the new bayesian score. This reduces computational time by a whole order of magnitude.

Modified K2 algorithm

After observing some of the generated graphs from the standard K2 algorithm, I realised the direction of edges were sometimes counter-intuitive. For example, I would expect *survive* to be the child rather than the parent of *sex*. This led to me to test whether swapping the direction of edges would improve my result.

Adding this step at the end of the K2 algorithm improved my results marginally. I found that alternating between the K2 and swapping edge directions helped improve my results even further.

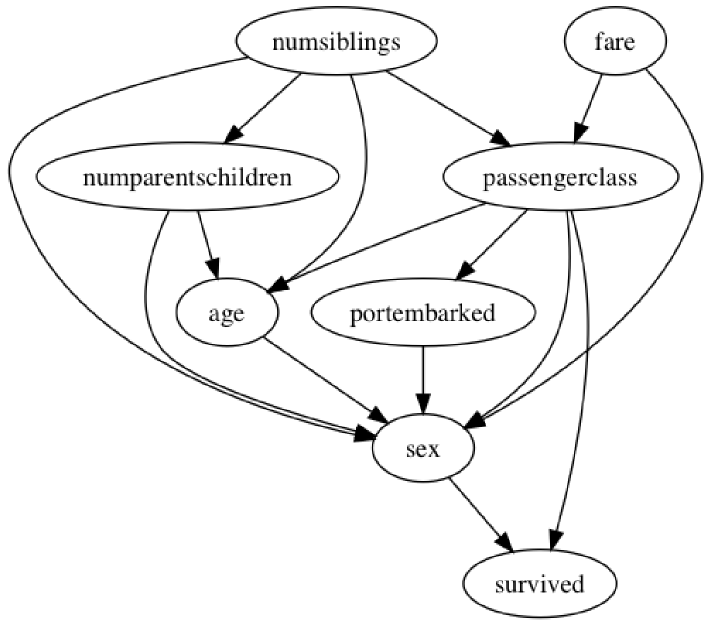
My final approach seems to be a hybrid between K2 and local search.

Rather than optimizing one node completely, I decide to introduce more ‘fairness’ to each node by giving everyone a chance. In every iteration, I add at most one parent to each node, then I check whether swapping the direction of each edge can improve the bayesian score. This gives every node a ‘chance’ to become the parent rather than the child node.

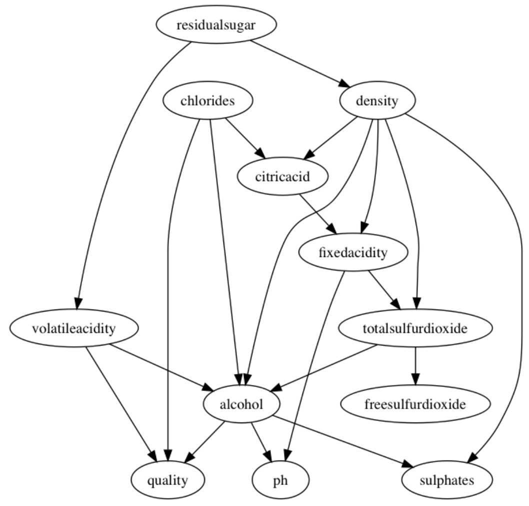
I repeat this until my score does not improve.

Shortcomings
The shortcomings of my method is clear. This is a completely greedy approach. The algorithm never makes a change that causes the score to drop, even if it might lead to a bigger gain afterwards.

Results
small.csv - 65.5s



medium.csv - 292.7s



large.csv - 2610.1s