

# Learning on Bayesian Networks

Rose F. Liu, Rusmin Soetjipto  
rliu,rusmin@mit.edu

December 2, 2004

## 1 Introduction

In this paper, we analyze different algorithms involved in Bayesian Network Learning. In section 2 we describe the Maximum Likelihood Estimator for learning the parameters of a Bayes net given a network structure and dataset. In section 3 we analyze the BIC scoring metric for Bayesian nets. In section 4 we describe three ways of generating initial networks for the greedy search algorithm. In section 5 we describe and evaluate our implementation of the greedy local structure search algorithm. In section 6, we extend our greedy search to enable random restarts of growing magnitudes to help escape local maximums. In section 7, we evaluate different versions of the BIC scoring algorithm.

## 2 Parameter Learning with Maximum Likelihood Estimator

Given the network structures shown in Figures 1 and 2 and the datafiles dataA.txt and dataB.txt, we used Maximum Likelihood Estimator to obtain the parameters (CPT's) for the networks. We implemented two versions of Maximum Likelihood: without Bayesian correction and with Bayesian correction. For the Bayesian correction case we used the prior  $B(1, 1)$ . Probabilities calculated with Bayesian correction are less extreme. For example, if a particular sequence for a node's parents does not show up in the data, instead of getting an undefined probability for that node's CPT assignment when not using Bayesian Correction, we would get a probability of 0.5 using Bayesian Correction.

Tables 1-4 show the resulting conditional probability tables (CPT's) for net A. Tables 5-12 show the resulting conditional probability tables (CPT's) for net B. Note that the probabilities calculated with and without the Bayesian correction are shown.

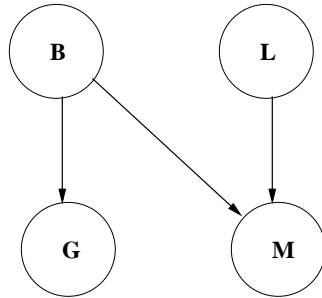


Figure 1: Network A (netA)

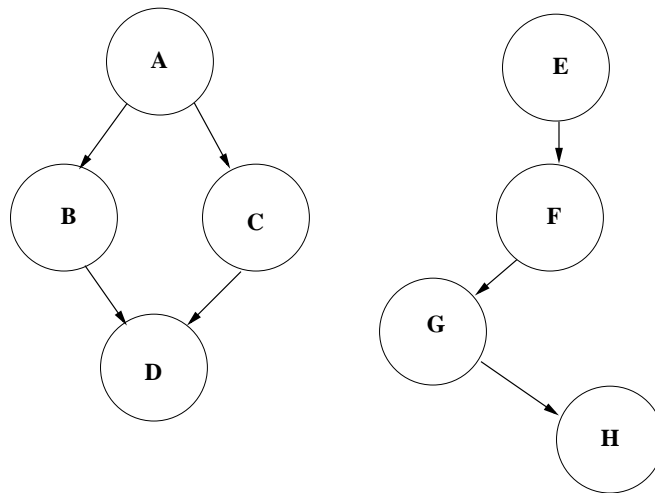


Figure 2: Network B (netB)

B	Without Correction	With Correction
false	0.06	0.06862745098039216
true	0.94	0.9313725490196079

Table 1: NetA: CPT for node B calculated with Maximum Likelihood

B	G	Without Correction	With Correction
false	false	1.0	0.875
false	true	0.0	0.125
true	false	0.05319148936170213	0.0625
true	true	0.9468085106382979	0.9375

Table 2: NetA: CPT for node G calculated with Maximum Likelihood

L	Without Correction	With Correction
false	0.32	0.3235294117647059
true	0.68	0.6764705882352942

Table 3: NetA: CPT for node L calculated with Maximum Likelihood

B	L	M	Without Correction	With Correction
false	false	false	1.0	0.75
false	false	true	0.0	0.25
false	true	false	1.0	0.8333333333333334
false	true	true	0.0	0.16666666666666666
true	false	false	0.9666666666666667	0.9375
true	false	true	0.03333333333333333	0.0625
true	true	false	0.109375	0.12121212121212122
true	true	true	0.890625	0.8787878787878788

Table 4: NetA: CPT for node M calculated with Maximum Likelihood

A	Without Correction	With Correction
false	0.4932	0.493202718912435
true	0.5068	0.5067972810875649

Table 5: NetB: CPT for node A calculated with Maximum Likelihood

A	B	Without Correction	With Correction
false	false	0.2733171127331711	0.27350081037277146
false	true	0.7266828872668288	0.7264991896272285
true	false	0.27111286503551696	0.27129337539432175
true	true	0.728887134964483	0.7287066246056783

Table 6: NetA: CPT for node B calculated with Maximum Likelihood

A	C	Without Correction	With Correction
false	false	0.370235198702352	0.3703403565640194
false	true	0.629764801297648	0.6296596434359806
true	false	0.356353591160221	0.35646687697160884
true	true	0.643646408839779	0.6435331230283912

Table 7: NetA: CPT for node C calculated with Maximum Likelihood

B	C	D	Without Correction	With Correction
false	false	false	0.15926493108728942	0.16030534351145037
false	false	true	0.8407350689127105	0.8396946564885496
false	true	false	0.1638418079096045	0.1647887323943662
false	true	true	0.8361581920903954	0.8352112676056338
true	false	false	0.19604471195184867	0.1965665236051502
true	false	true	0.8039552880481513	0.8034334763948497
true	true	false	0.22172859450726978	0.22195318805488298
true	true	true	0.7782714054927302	0.778046811945117

Table 8: NetA: CPT for node D calculated with Maximum Likelihood

E	Without Correction	With Correction
false	0.686	0.6859256297481008
true	0.314	0.3140743702518992

Table 9: NetB: CPT for node E calculated with Maximum Likelihood

E	F	Without Correction	With Correction
false	false	0.546064139941691	0.546037296037296
false	true	0.45393586005830905	0.45396270396270394
true	false	0.5356687898089172	0.5356234096692112
true	true	0.4643312101910828	0.4643765903307888

Table 10: NetA: CPT for node F calculated with Maximum Likelihood

F	G	Without Correction	With Correction
false	false	0.2778187177597642	0.27798232695139913
false	true	0.7221812822402358	0.7220176730486009
true	false	0.2655293088363955	0.26573426573426573
true	true	0.7344706911636045	0.7342657342657343

Table 11: NetA: CPT for node G calculated with Maximum Likelihood

G	H	Without Correction	With Correction
false	false	0.1550330639235856	0.15553925165077037
false	true	0.8449669360764144	0.8444607483492297
true	false	0.14976641934597418	0.14995880252677835
true	true	0.8502335806540259	0.8500411974732216

Table 12: NetA: CPT for node H calculated with Maximum Likelihood

BICScore component	Without Correction	With Correction
Log Likelihood Score	-189.56312766291126	-193.23553652286992
M (dataset size)	100	100
Dim[G]	8	8
Structure Penalty	-26.5754247590989	-26.5754247590989
BICScore	-216.13855242201015	-219.81096128196882

Table 13: BIC score for network A

### 3 Scoring of Graphs

In order to evaluate how good a network fits the data, we implemented the BIC scoring metric as shown in Equation 1. The first element of the BIC score  $l(\theta_G : D)$  is the Log Likelihood score which evaluates the log of the likelihood of the data given the network structure. If a network is learned based only on the Log Likelihood score, the resulting network tends to be very connected (complex). This is because structural search using the Log Likelihood score tends to overfit the data, losing generality. The second component of the BIC score  $-\frac{\log_2 M}{2} Dim[G]$ , is the structural penalty, which counterbalances the likelihood score by penalizing complex graphs. The complexity of the graph is determined by the number of independent parameters in the graph ( $Dim[G]$ ). In our implementation of BICScore, we added adjustable weights to the log likelihood component as well as the structural penalty component. This way we can experiment with different weightings of the BIC score, as described in section 7.

$$S(G : D) = l(\theta_G : D) - \frac{\log_2 M}{2} Dim[G] \quad (1)$$

In order to optimize the Greedy Search algorithm described in section 5, our implementation of the BIC score takes advantage of the fact that the BIC score is decomposable. As described in [1], a score is decomposable if it can be expressed in the form of Equation 2.  $FamScore(X|Pa : D)$  is a score measuring how well the the Parents of X in the graph serve as the parents of X in the dataset D.

$$Score(G : D) = \sum_i FamScore(X_i|Pa_i : D) \quad (2)$$

Our decomposition of the BIC score is shown in Equation 3. The BIC score for the network is the sum of the FamScore score of each node. And in this case  $FamScore = \sum_m (\log_2 \theta_i - \frac{\log_2 M}{2} Dim[i])$ .

$$S(G : D) = \sum_i \sum_m (\log_2 \theta_i - \frac{\log_2 M}{2} Dim[i]) \quad (3)$$

where m = size of dataset, i = node in graph G

Table 13 shows the BIC scores and their components for both versions of network A (with and without Bayesian correction). Note that the BICScore is calculated using  $\log_2$ . Also note that our BIC score for network A without Bayesian correction matches the score provided in dataA.txt.

Table 14 shows the BIC scores and their components for both versions of network B (with and without Bayesian correction). Note that the BICScore is calculated using  $\log_2$ .

BICScore component	Without Correction	With Correction
Log Likelihood Score	-34285.23558340326	-34285.24887441076
M (dataset size)	5000	5000
Dim[G]	16	16
Structure Penalty	-98.3016990363956	-98.3016990363956
BICScore	-34383.537282439655	-34383.55057344716

Table 14: BIC score for network B

Since the Log Likelihood Score for network B is very negative, that means that network B does not fit the data well. The structure penalty for network B is more negative than the structure penalty for network A. This is mostly because network B has more variables and therefore requires more parameters which increases Dim[netB]. Even if all the nodes in network B were not connected, the dimension of network B would be 8 which is the same as the dimension for network A (which has connected components). Since the Log Likelihood score dominates over the structural penalty score (Log Likelihood is much more negative than the structural penalty), the reason why the BICScore for netB is so low is not because the network is too complex. It is because the network does not fit the data well.

## 4 Initial Networks

We implemented three methods to generate initial networks for the local greedy structure search: ConnectNone, ConnectAll, and ConnectRandom. ConnectNone reads a data file and generates a network with no dependencies. ConnectAll reads a data file and generates a fully connected network, and ConnectRandom generates a randomly connected network. Since ConnectNone is trivial to implement, we are just going to describe the implementation of ConnectAll and ConnectRandom.

Our implementation of ConnectAll first orders the nodes in some random order. Each node in the list is set as a parent of all nodes that follow it in the ordering. For example, Node  $i$  will have parent nodes 0 to  $i-1$ . By connecting the nodes in this manner, ConnectAll always returns a legal network with no directed cycles. Also, since the ordering of nodes is randomized, the fully connected network generated by ConnectAll is not deterministic.

Our implementation of ConnectRandom first calls ConnectAll to get a fully connected graph. Then it randomly deletes a random number of edges. Since ConnectRandom starts with a legal network, and deleting edges will not create any cycles, ConnectRandom will always return a legal randomly connected network.

Figures 3 and 4 show two sample initial networks for dataA.txt. Figure 3 was generated using ConnectAll and Figure 4 was generated using ConnectRandom.

Figures 5 and 6 show two sample initial networks for dataB.txt. Figure 5 was generated using ConnectAll and Figure 6 was generated using ConnectRandom.

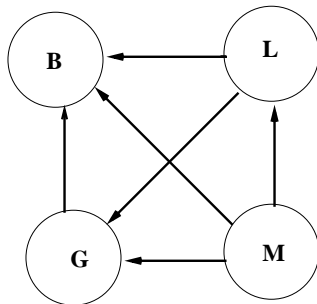


Figure 3: Initial Network for dataA generated by ConnectAll

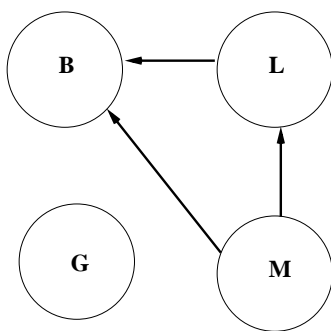


Figure 4: Initial Network for dataA generated by ConnectRandom

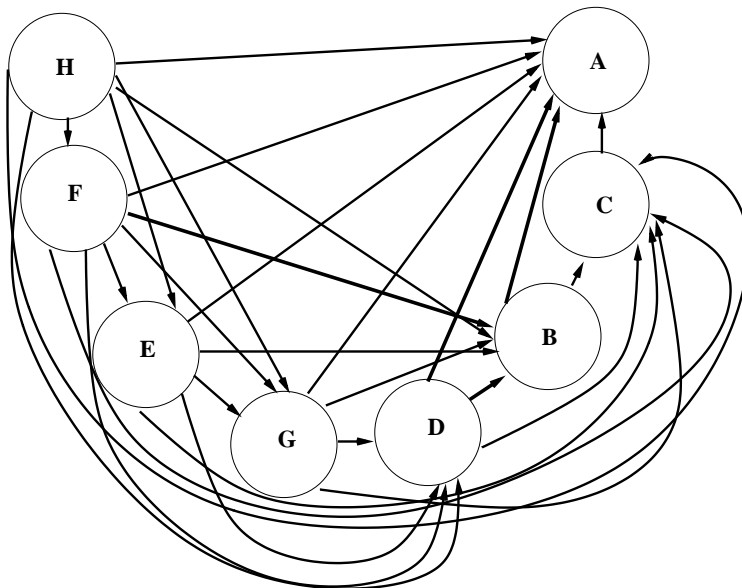


Figure 5: Initial Network for dataB generated by ConnectAll

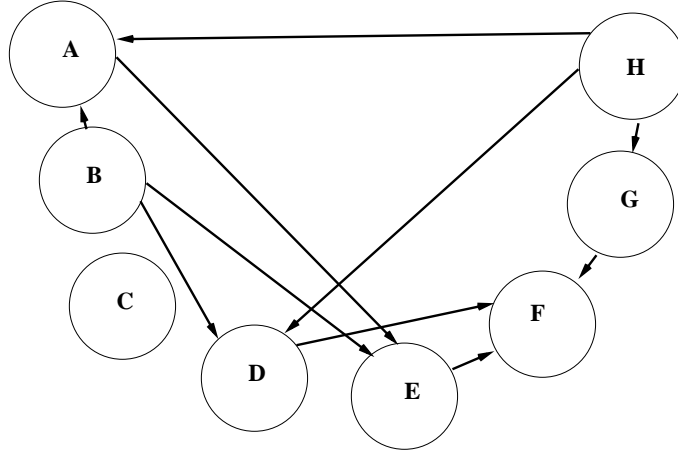


Figure 6: Initial Network for dataB generated by ConnectRandom

## 5 Greedy Structural Search Algorithm

### 5.1 Implementation

We implemented a first-ascent hill-climbing local greedy search through the space of Bayes net structures. Our implementation is similar to that described in Figure 14.7 in [1]. But instead of trying every possible one-step operation on the graph before updating the graph and moving on, our implementation is even more greedy. We keep trying another one-step operation only if we have not encountered an operation that improves the score. Once we encounter an operation that improves the score, we update the graph and move on. If we have not found an operation that improves the score after trying every possible operation, then we know that a local maximum has been found and the greedy algorithm exits. Because we don't always search through the entire list of possible operations, we needed to make sure that we are not biased towards some operations. Therefore, we randomly shuffle the list of all possible operations on the graph and then we search through this list. Figure 7 shows the pseudocode for our implementation.

Our Greedy search performs three types of operations on the graph:

- `addEdge(fromNode, toNode)` - Adds an edge starting at `fromNode` and pointing to `toNode`. `fromNode` will become the parent of `toNode`.
- `removeEdge(fromNode, toNode)` - Removes the edge between `fromNode` and `toNode`. `fromNode` will no longer be the parent of `toNode`.
- `reverseEdge(fromNode, toNode)` - Reverses the edge between `fromNode` and `toNode`. `fromNode` will now be the child of `toNode`.

The list of all possible operations to try for the current graph consists of these three types of operations applied to all combinations of nodes, where possible. We do not add redundant operations into this list. For example, if an edge already exists from node A to node B, then we will not add the operation `addEdge(A,B)` to the list.

```

Procedure Greedy(G_init, Data, ScoringMetric)
{
  G_best <- G_init
  do
  {
    G <- G_best
    Progress <- false
    List_of_Operators <- getListOfAllOperators(G)
    for each operator O in List_of_Operators
    {
      // G remains unmodified while G_temp is the graph with O applied on G
      G_temp <- applyOperator(O, G)

      if(G_temp is Legal Structure)
      {
        if(ScoringMetric(G_temp, data) > ScoringMetric(G_best, data))
        {
          G_best <- G_temp
          Progress <- true
          BREAK OUT OF FOR LOOP
        }
      }
    }
  } while (Progress)

  return G_best
}

```

Figure 7: Pseudocode of our implementation of the Greedy Local Structural Search

### 5.1.1 Optimizations in Implementation

Each time an operation is applied to the graph, the Greedy algorithm needs to learn the best parameters for the modified graph given the data. Since the Maximum Likelihood estimator is decomposable, we can maximize each local likelihood separately for each node. Therefore instead of calculating all the parameters of the graph over again, our implementation of Greedy just runs the Maximum Likelihood Estimator on the nodes that were affected by the operation. The CPT's of other nodes would not have changed. For example, the `addEdge(fromNode, toNode)` operation will only change the CPT of the `toNode`. Therefore we only recalculate the parameters of the `toNode`. For `removeEdge(fromNode, toNode)`, only the `toNode`'s CPT is recalculated, and for `reverseEdge(fromNode, toNode)`, the CPTs of both `fromNode` and `toNode` are calculated.

Each time an operation is applied to the graph, the new graph needs to be scored. Just like we efficiently calculated the parameters of the new graph, we also efficiently calculated the score of the new graph. Our Greedy implementation only recalculates the *FamScore* of the nodes affected by the operation. The *FamScore* of the other nodes remain unchanged. Equation 3 shows how we decomposed the BICScore.

When comparing the score of the new graph to the score of the current best graph, our Greedy implementation just compares the new *FamScores* of the affected nodes to their corresponding *FamScores* in the best structure. Since we know that the *FamScores* of all the other nodes did not change, there is no point in comparing them.

### 5.1.2 Checking for Legality of Graph

As described in Section 4, our algorithms for generating initial networks guarantee that the initial networks are legal (without any directed cycles). If our initial networks are legal, and if we only perform legal operations (operations that produce legal graphs), then our graphs will always be legal. Based on this logic, our implementation will only permanently apply an operation onto the graph if the resulting graph is legal. Therefore the graph will always be legal before a new operation is applied. Note that `removeEdge` will always produce a legal graph given that the original graph was legal. Therefore we only need to check the legality of `addEdge` and `reverseEdge` operations.

The following statements describe the main intuition behind our method of checking for legal operations.

- If the graph is legal before a legal operation is applied, then the resulting graph is legal.
- If the graph is legal before an illegal operation is applied (produces directed cycle(s)), then the resulting graph is not legal AND the nodes affected by the illegal operation must be part of the cycle(s).

Since we only permanently apply legal operations, the graph before any operation is applied will always be legal.

Our checking algorithm first temporarily applies the new operation onto the graph. Based on the logic behind the statements above, it only needs to traverse all the paths starting from the child node that was produced by the operation (i.e. `addEdge`, `reverseEdge`). This is because if there is a directed cycle(s), the child node of the operation must be part of the cycle(s). While traversing from the child node, if the child node is encountered again, then we have detected a cycle, so the operation is illegal. If we reach all the root nodes of the subgraph without encountering the child

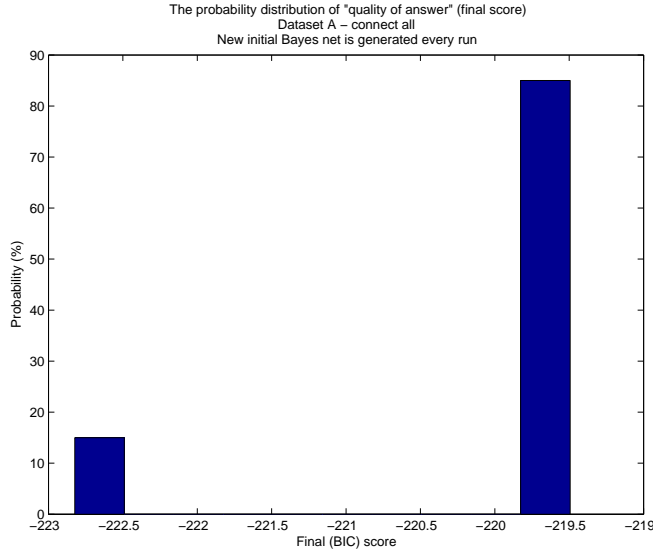


Figure 8: Histogram of scores of the Greedy runs for the 100 initial networks generated by connectAll for net A

node again, then the operation is legal. (If the network is connected then there is only one subgraph that is the entire graph). Note that we traverse the paths of parents since the parents of each node are stored in the node structure. If the operation is legal, then we just return the resulting graph. If the operation is illegal then we undo the operation and return the original graph (unmodified).

## 5.2 Experiments using Greedy Search Algorithm

In this section we compare the performance of the two initialization methods (connectAll, and connectRandom) for the two datasets.

For each dataset we generated 100 initial networks using the connectAll method and 100 initial networks using the connectRandom method. For each dataset, we fed both sets of 100 initial networks generated by connectAll and ConnectRandom into the Greedy algorithm to obtain the 200 corresponding output nets for each dataset.

We measure the performance of each initial network generation method by two criteria:

- the final scores of the resulting graphs obtained using Greedy search
- the number of iterations the Greedy search needed to reach the final score. One iteration represents the testing of one operation onto the graph to see if it improves the graph's score.

Figure 8 and Figure 9 show histograms of the final scores and number of iterations for the initial networks generated by connectAll using network A. As can be seen in Figure 8, about 85% of the time, the Greedy algorithm, using an initial net generated from connectAll, will find the higher of the BICScores (-219.5). The fact that only two distinct local maxima scores are detected (without any spread) suggests that these local maxima are not part of a cluster of maxima that main similar scores. Figure 9 shows that on average, the Greedy algorithm runs for about 31 iterations before reaching a local maximum (given an initial network generated by connectAll).

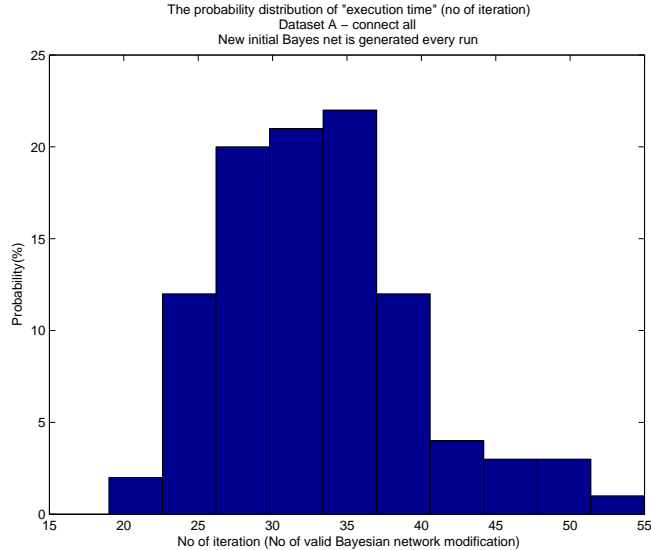


Figure 9: Histogram of number of iterations of the Greedy runs for the 100 initial networks generated by connectAll for net A

Figure 10 and Figure 11 show histograms of the final scores and number of iterations for the initial networks generated by connectRandom using network A. As can be seen in Figure 10, about 82% of the time, the Greedy algorithm, using an initial net generated from connectRandom, will find the higher BICScore (-219.5). Figure 11 shows that on average, the Greedy algorithm runs for about 27 iterations before reaching a local maximum (given an initial network generated by connectRandom).

Since 85% of the initial networks produced by connectAll obtained the better BICScore for net A, while 82% of the initial nets produced by connectRandom obtained the same better BICScore for net A, we conclude that connectAll is a slightly better initial network generation algorithm than connectRandom for dataset A. This may be due to the fact that the true structure representing the data is quite connected, so it is easier to reach the true structure in the search space when starting at a connectAll init net than one of connectRandom. Sometimes, connectRandom may generate initial nets that are quite unconnected so they may not get past a potential neighboring local maximum (score = -222.5) that is not as connected as the true structure.

The fact that the runs for connectRandom converge in less iterations than connectAll may suggest that when starting at a somewhat unconnected structure, there is a local maximum nearby that is not as good as the maximum that is farther away (near the connected end of the search space). The distance between the lower local maximum and the connectRandom initial networks is less than the distance between the higher local maximum and the very connected init structures generated by connectAll.

Figure 12 and Figure 13 show histograms of the final scores and number of iterations for the initial networks generated by connectAll using network B. As can be seen in Figure 12, about 90% of the time, the Greedy algorithm, using an initial net generated from connectAll, will find the higher of the local maxima( around -31560 to -31550). Unlike for netA, the greedy algorithm obtains a larger spread of scores near the two local maxima. Therefore there must be two clusters of local maxima, where each cluster of maxima contain maxima with similar scores. Figure 13 shows that

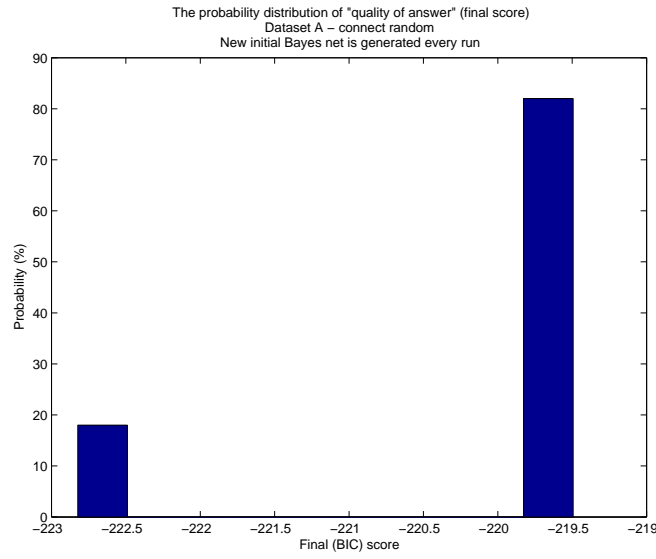


Figure 10: Histogram of scores of the Greedy runs for the 100 initial networks generated by connectRandom for net A

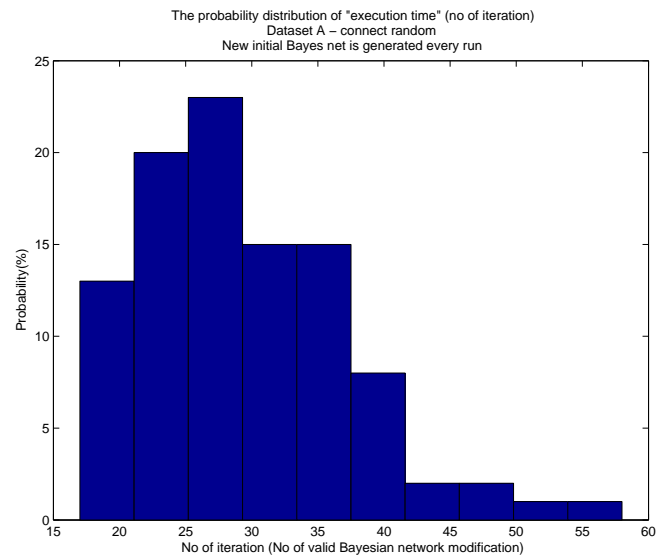


Figure 11: Histogram of number of iterations of the Greedy runs for the 100 initial networks generated by connectRandom for net A

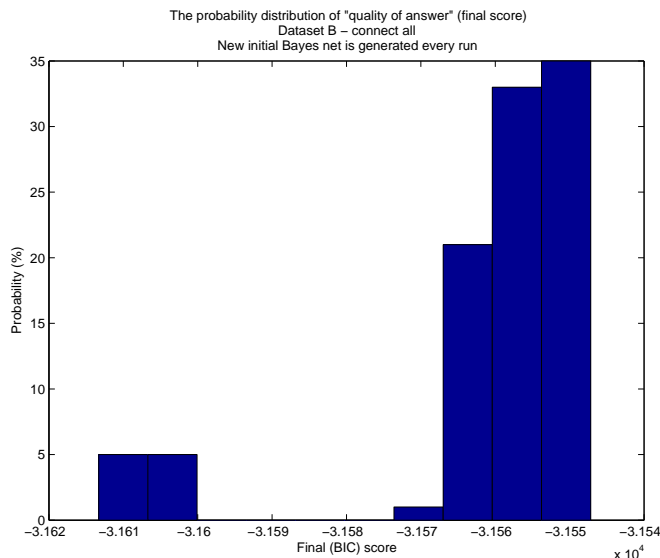


Figure 12: Histogram of scores of the Greedy runs for the 100 initial networks generated by connectAll for net B

on average, the Greedy algorithm runs for about 260 iterations before reaching a local maximum (given an initial network generated by connectAll).

Figure 14 and Figure 15 show histograms of the final scores and number of iterations for the initial networks generated by connectRandom using network B. As can be seen in Figure 14, about 93% of the time, the Greedy algorithm, using an initial net generated from connectRandom, will find the higher of the local maxima (around -31560 to -31550). Again, the larger spread of scores near the two local maxima reinforces our hypothesis that there must be two clusters of local maxima, where each cluster of maxima contain maxima with similar scores. Figure 15 shows that on average, the Greedy algorithm runs for about 250 iterations before reaching a local maximum (given an initial network generated by connectAll).

Because the greedy runs using connectRandom init nets found the higher local maxima (-31560) 3% more often than the greedy runs using connectAll, we conclude that connectRandom is a better initialization algorithm for dataset B. Since we saw that connectAll was better for dataset A, this suggests that the best initialization method varies depending on the dataset. Therefore we cannot expect one initialization algorithm to be the best for all datasets.

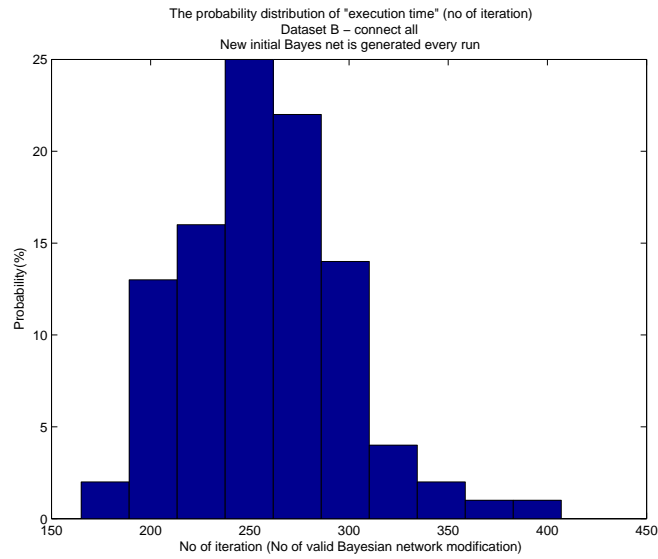


Figure 13: Histogram of number of iterations of the Greedy runs for the 100 initial networks generated by connectAll for net B

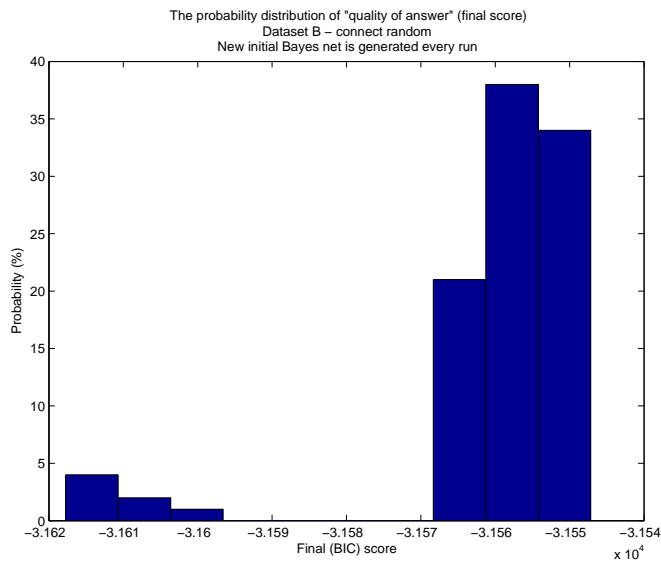


Figure 14: Histogram of scores of the Greedy runs for the 100 initial networks generated by connectRandom for net B

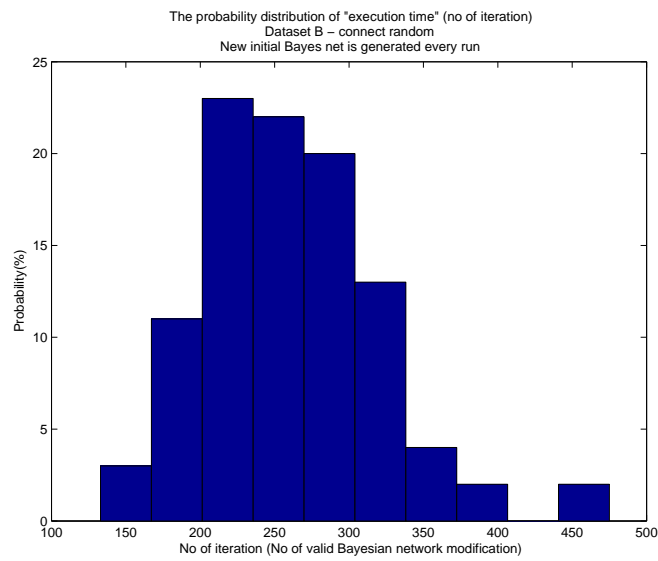


Figure 15: Histogram of number of iterations of the Greedy runs for the 100 initial networks generated by connectRandom for net B

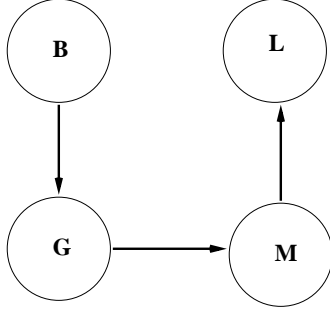


Figure 16: Output Network for dataA generated by ConnectAll

B	With Correction
false	0.06862745098039216
true	0.9313725490196079

Table 15: CPT for node B of output network in Figure 16

### 5.2.1 Sample Output Networks

The following Figures and Tables show three output networks and their CPTs for both datasets A and B. These output networks were generated using the Greedy search algorithm with different initial networks generated by the ConnectAll method. Since our Greedy search algorithm is first-ascent hill climbing and the initial networks are different for each case, we do get different results (i.e. final networks) each time we run the algorithm.

Figure 16 and Tables 15 - 18 show the first output network for dataset A and its CPTs. The BICScore with Bayesian Correction for this network is -222.82337647865944.

Figure 17 and Tables 19 - 22 show the second output network for dataset A and its CPTs. The BICScore with Bayesian Correction for this network is -219.49308741207318.

Figure 18 and Tables 23 - 26 show the third output network for dataset A and its CPTs. The BICScore with Bayesian Correction for this network is -219.70695025162132.

B	G	With Correction
false	false	0.875
false	true	0.125
true	false	0.0625
true	true	0.9375

Table 16: CPT for node G of output network in Figure 16

L	M	With Correction
false	false	0.7272727272727273
false	true	0.03333333333333333
true	false	0.2727272727272727
true	true	0.9666666666666667

Table 17: CPT for node L of output network in Figure 16

G	M	With Correction
false	false	0.6923076923076923
false	true	0.3076923076923077
true	false	0.38461538461538464
true	true	0.6153846153846154

Table 18: CPT for node L of output network in Figure 16

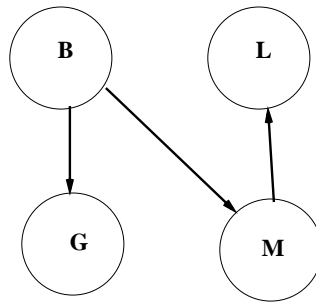


Figure 17: Output Network for dataA generated by ConnectAll

B	With Correction
false	0.06862745098039216
true	0.9313725490196079

Table 19: CPT for node B of output network in Figure 17

B	G	With Correction
false	false	0.875
false	true	0.125
true	false	0.0625
true	true	0.9375

Table 20: CPT for node G of output network in Figure 17

L	M	With Correction
false	false	0.7272727272727273
false	true	0.03333333333333333
true	false	0.2727272727272727
true	true	0.9666666666666667

Table 21: CPT for node L of output network in Figure 17

B	M	With Correction
false	false	0.875
false	true	0.125
true	false	0.3854166666666667
true	true	0.6145833333333334

Table 22: CPT for node L of output network 17

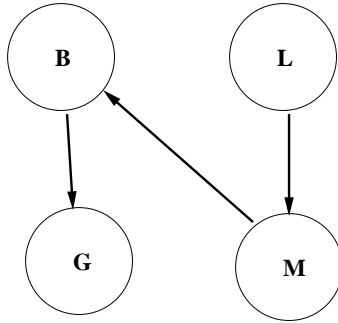


Figure 18: Output Network for dataA generated by ConnectAll

B	M	With Correction
false	false	0.1590909090909091
false	true	0.016666666666666666
true	false	0.8409090909090909
true	true	0.9833333333333333

Table 23: CPT for node B of output network in Figure 18

B	G	With Correction
false	false	0.875
false	true	0.125
true	false	0.0625
true	true	0.9375

Table 24: CPT for node G of output network in Figure 18

L	With Correction
false	0.3235294117647059
true	0.6764705882352942

Table 25: CPT for node L of output network in Figure 18

L	M	With Correction
false	false	0.9411764705882353
false	true	0.058823529411764705
true	false	0.17142857142857143
true	true	0.8285714285714286

Table 26: CPT for node L of output network in Figure 18

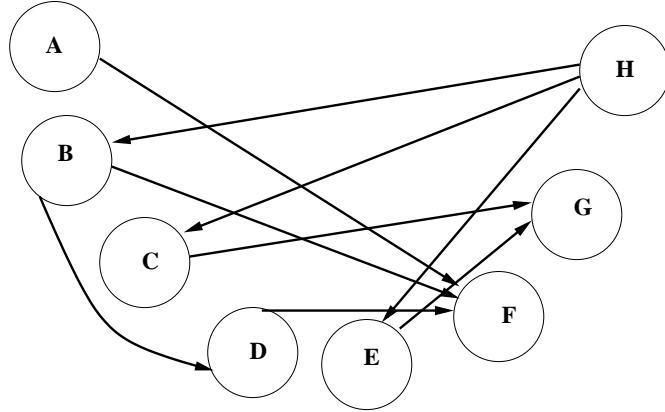


Figure 19: Output Network for dataB generated by ConnectAll

A	With Correction
false	0.493202718912435
true	0.5067972810875649

Table 27: CPT for node A of output network in Figure 19

Figure 19 and Tables 27 - 34 show the first output network for dataset B and its CPTs. The BICScore with Bayesian Correction for this network is -31603.003268718967.

B	H	With Correction
false	false	0.8522427440633246
false	true	0.16886481394253414
true	false	0.14775725593667546
true	true	0.8311351860574658

Table 28: CPT for node B of output network in Figure 19

C	H	With Correction
false	false	0.7031662269129287
false	true	0.302637776731041
true	false	0.29683377308707126
true	true	0.697362223268959

Table 29: CPT for node C of output network in Figure 19

B	D	With Correction
false	false	0.16214233308877476
false	true	0.8378576669112252
true	false	0.21367756110958527
true	true	0.7863224388904148

Table 30: CPT for node D of output network in Figure 19

E	H	With Correction
false	false	0.2519788918205805
false	true	0.7633066415449835
true	false	0.7480211081794196
true	true	0.2366933584550165

Table 31: CPT for node E of output network in Figure 19

A	B	D	F	With Correction
false	false	false	false	0.923728813559322
false	false	false	true	0.07627118644067797
false	false	true	false	0.4517857142857143
false	false	true	true	0.5482142857142858
false	true	false	false	0.9715762273901809
false	true	false	true	0.028423772609819122
false	true	true	false	0.6394606103619588
false	true	true	true	0.36053938963804116
true	false	false	false	0.6415094339622641
true	false	false	true	0.3584905660377358
true	false	true	false	0.20512820512820512
true	false	true	true	0.7948717948717948
true	true	false	false	0.7944162436548223
true	true	false	true	0.20558375634517767
true	true	true	false	0.3994509265614276
true	true	true	true	0.6005490734385724

Table 32: CPT for node F of output network in Figure 19

C	E	G	With Correction
false	false	false	0.4467323187108326
false	false	true	0.5532676812891674
false	true	false	0.3058321479374111
false	true	true	0.6941678520625889
true	false	false	0.24298662063012516
true	false	true	0.7570133793698748
true	true	false	0.1010332950631458
true	true	true	0.8989667049368542

Table 33: CPT for node G of output network in Figure 19

H	With Correction
false	0.15133946421431427
true	0.8486605357856857

Table 34: CPT for node H of output network in Figure 19

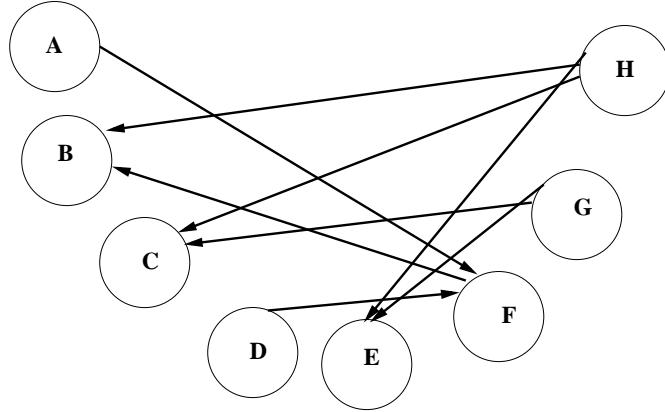


Figure 20: Output Network for dataB generated by ConnectAll

A	With Correction
false	0.493202718912435
true	0.5067972810875649

Table 35: CPT for node A of output network in Figure 20

Figure 20 and Tables 35 - 42 show the first output network for dataset B and its CPTs. The BICScore with Bayesian Correction for this network is -31551.472898512424.

B	F	H	With Correction
false	false	false	0.7474747474747475
false	false	true	0.10852713178294573
false	true	false	0.9642857142857143
false	true	true	0.24195223260643822
true	false	false	0.25252525252525254
true	false	true	0.8914728682170543
true	true	false	0.03571428571428571
true	true	true	0.7580477673935618

Table 36: CPT for node B of output network in Figure 20

C	G	H	With Correction
false	false	false	0.892018779342723
false	false	true	0.454861111111111
false	true	false	0.6288848263254113
false	true	true	0.24612403100775193
true	false	false	0.107981220657277
true	false	true	0.545138888888888
true	true	false	0.3711151736745887
true	true	true	0.7538759689922481

Table 37: CPT for node C of output network in Figure 20

D	With Correction
false	0.1995201919232307
true	0.8004798080767693

Table 38: CPT for node D of output network in Figure 20

E	G	H	With Correction
false	false	false	0.3051643192488263
false	false	true	0.865451388888888
false	true	false	0.23217550274223034
false	true	true	0.7251291989664083
true	false	false	0.6948356807511737
true	false	true	0.134548611111111
true	true	false	0.7678244972577697
true	true	true	0.27487080103359174

Table 39: CPT for node E of output network in Figure 20

A	D	F	With Correction
false	false	false	0.9622266401590457
false	false	true	0.03777335984095427
false	true	false	0.5861718352821556
false	true	true	0.41382816471784445
true	false	false	0.7630522088353414
true	false	true	0.23694779116465864
true	true	false	0.34362745098039216
true	true	true	0.6563725490196078

Table 40: CPT for node F of output network in Figure 20

G	With Correction
false	0.2722910835665734
true	0.7277089164334266

Table 41: CPT for node G of output network in Figure 20

H	With Correction
false	0.15133946421431427
true	0.8486605357856857

Table 42: CPT for node H of output network in Figure 20

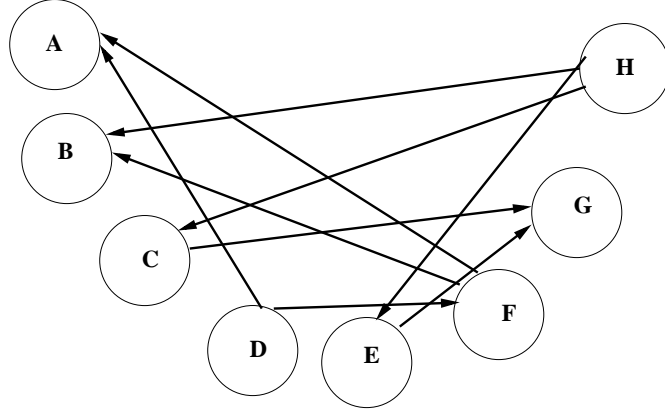


Figure 21: Output Network for dataB generated by ConnectAll

A	D	F	With Correction
false	false	false	0.5601851851851852
false	false	true	0.1386861313868613
false	true	false	0.6218985976267529
false	true	true	0.37807710171853226
true	false	false	0.4398148148148148
true	false	true	0.8613138686131386
true	true	false	0.378101402373247
true	true	true	0.6219228982814677

Table 43: CPT for node A of output network in Figure 21

Figure 21 and Tables 43 - 50 show the first output network for dataset B and its CPTs. The BICScore with Bayesian Correction for this network is -31552.981721842243.

## 6 Greedy Structural Search Algorithm with Random Restarts

We implemented a modified version of the Search with Restarts described in Figure 14.9 in [1]. Two interesting input parameters in this method are the restart length and the number of random restarts. The restart length is the number of random operations to apply onto a local maximum graph before running Greedy on the resulting graph. The number of random restarts is the number of times we apply the restart algorithm before returning the best graph.

Our implementation of Greedy with Random Restarts applies random restarts of growing magnitudes. Instead of fixing a random restart length, we multiply the restart length at each restart by *restartLengthFactor*. The idea behind random restarts with growing magnitudes is that we want to first search for a better maximum in the search space that is close to the current maximum. But, after a few restarts, we may not find any better maximum close by, so we want to search farther away, to see if we can find an even better maximum at a more distant location in the search space.

B	F	H	With Correction
false	false	false	0.7474747474747475
false	false	true	0.10852713178294573
false	true	false	0.9642857142857143
false	true	true	0.24195223260643822
true	false	false	0.25252525252525254
true	false	true	0.8914728682170543
true	true	false	0.03571428571428571
true	true	true	0.7580477673935618

Table 44: CPT for node B of output network in Figure 21

C	H	With Correction
false	false	0.7031662269129287
false	true	0.302637776731041
true	false	0.29683377308707126
true	true	0.697362223268959

Table 45: CPT for node C of output network in Figure 21

D	With Correction
false	0.1995201919232307
true	0.8004798080767693

Table 46: CPT for node D of output network in Figure 21

E	H	With Correction
false	false	0.2519788918205805
false	true	0.7633066415449835
true	false	0.7480211081794196
true	true	0.2366933584550165

Table 47: CPT for node E of output network in Figure 21

D	F	With Correction
false	false	0.8638638638638638
false	true	0.13613613613613615
true	false	0.4626716604244694
true	true	0.5373283395755306

Table 48: CPT for node F of output network in Figure 21

C	E	G	With Correction
false	false	false	0.4467323187108326
false	false	true	0.5532676812891674
false	true	false	0.3058321479374111
false	true	true	0.6941678520625889
true	false	false	0.24298662063012516
true	false	true	0.7570133793698748
true	true	false	0.1010332950631458
true	true	true	0.8989667049368542

Table 49: CPT for node G of output network in Figure 21

H	With Correction
false	0.15133946421431427
true	0.8486605357856857

Table 50: CPT for node H of output network in Figure 21

## 6.1 Comparison of Greedy vs. Greedy with Random Restarts

We made 100 runs of Greedy and Greedy with Random Restarts using the BICScore ( $w_l = 0.8$  and  $w_s = 0.2$ ) for both cases. For both cases the initial networks were generated using connectAll. The initial restart length used was 3, the *restartLengthFactor* used was 2, and the number of restarts was fixed at 4. After obtaining the resulting graphs and BIC scores, we averaged the the 100 resulting graphs and BIC scores for each type of greedy run to get the following average scores and iterations:

- A w/o restart:  $\text{avg}(\text{finalscore}) = -160.2111$ ,  $\text{avg}(\text{iter}) = 32.22$
- A w/ restart:  $\text{avg}(\text{finalscore}) = -160.0936$ ,  $\text{avg}(\text{iter}) = 95.14$
- B w/o restart:  $\text{avg}(\text{finalscore}) = -25166$ ,  $\text{avg}(\text{iter}) = 251.54$
- B w/ restart:  $\text{avg}(\text{finalscore}) = -25166$ ,  $\text{avg}(\text{iter}) = 728.60$

These results were the ones we expected. We expected that the Greedy with restarts would return on average a higher or equal score to the Greedy without resrarts. This is because each run of Greedy with restarts explores a larger search space and therefore is more likely to escape local maxima. We also expected that the Greedy with restarts would run for a larger number of iterations since we fix the number of restarts that we perform at the beginning of the run to 4.

Figures 22 and ?? show histograms of the final scores obtained from 100 runs of Greedy with Restarts for dataset A and B respectively. In contrast, Figures ?? and ?? show the histograms of the final scores from 100 runs of Greedy for datasets A and B.

The Greedy Restart and Greedy Histograms on the score for data A suggests strongly that the Greedy Restart algorithm will on average find the better score. However, looking at the score histograms for data B, it is less obvious that Greedy Restart will on average provide a graph with

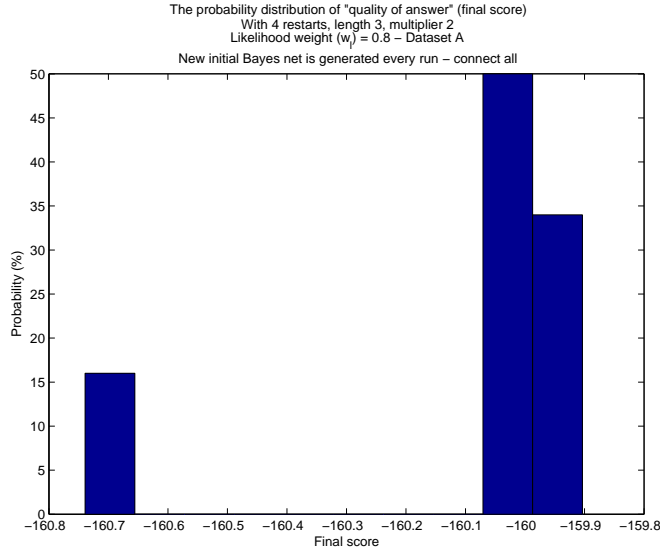


Figure 22: Histogram of the scores of 100 runs of Greedy with Random Restarts for data A

a better score. The fact that restarts does not help much in data B may be due to the fact that dataB's search space has two large clusters of local maxima (one better than another) while data A either does not have the clusters or has smaller clusters of 2 local maxima (see cluster hypothesis in previous section). Since B has larger clusters of maxima, it is easier for the Greedy search to find one of the many local maxima in the better score cluster. Therefore the fact that Greedy with Restarts can explore a larger search space does not really help. The search space is already easy enough for the Greedy algorithm to find one of the maxima in the better cluster.

As seen in Figures 26 to 29 The iterations for Greedy with Restarts is larger than the iterations for Greedy. Again this is because we fix the number of restarts. Therefore we run Greedy multiple times. Therefore it is expected that Greedy with Restarts should have a larger number of iterations. Since in general, Greedy with restarts seem to give an equal or better graph than Greedy there is a tradeoff between getting a better graph (higher score) vs. the time it takes to search.

## 7 Comparison of Different Scoring Metrics

In our study of different scoring metrics, we varied the weights of the log likelihood and structural penalty components of the BIC Score. Therefore we modified the BICScore to look like Equation 4.

$$S(G : D) = w_l * l(\theta_G : D) - w_s * \frac{\log_2 M}{2} Dim[G] \quad (4)$$

$w_l$  is the log likelihood weight, and  $w_s$  is the structural penalty weight.

In our experiments, we divide each dataset (A and B) into two datasets (Training and Testing datasets). We divide the datasets by randomly picking 0.7 of the datapoints to be in the Training dataset and 0.3 of the dataset to be in the Testing dataset. We repeat this operation 100 times on both datasets to generate 100 training and testing dataset pairs for each of dataset A and

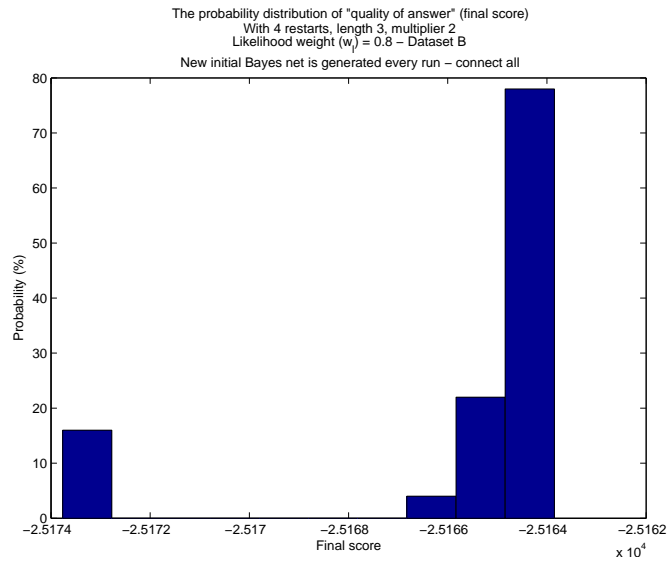


Figure 23: Histogram of the scores of 100 runs of Greedy with Random Restarts for data B

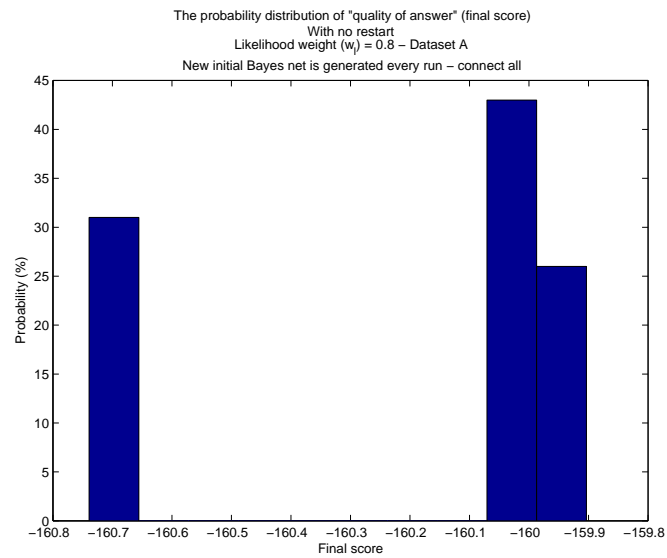


Figure 24: Histogram of the scores of 100 runs of Greedy for data A

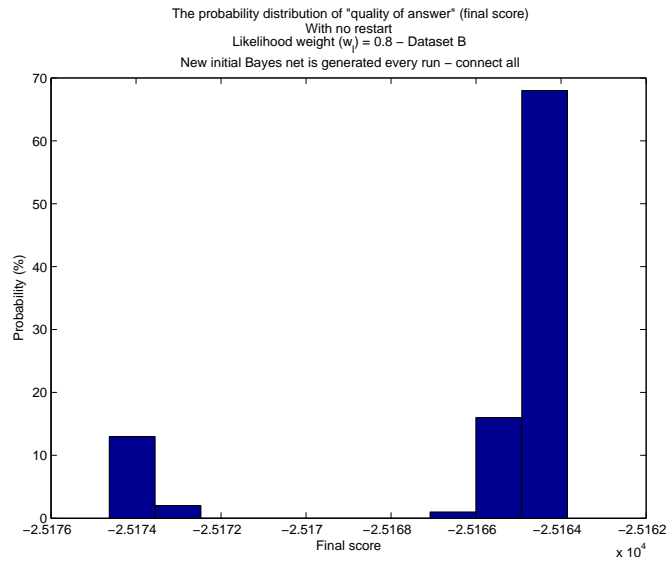


Figure 25: Histogram of the scores of 100 runs of Greedy for data B

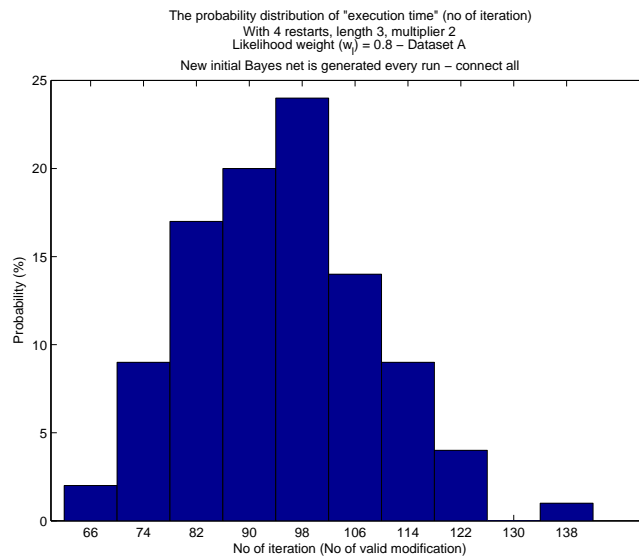


Figure 26: Histogram of the number of iterations of 100 runs of Greedy with Restarts for data A

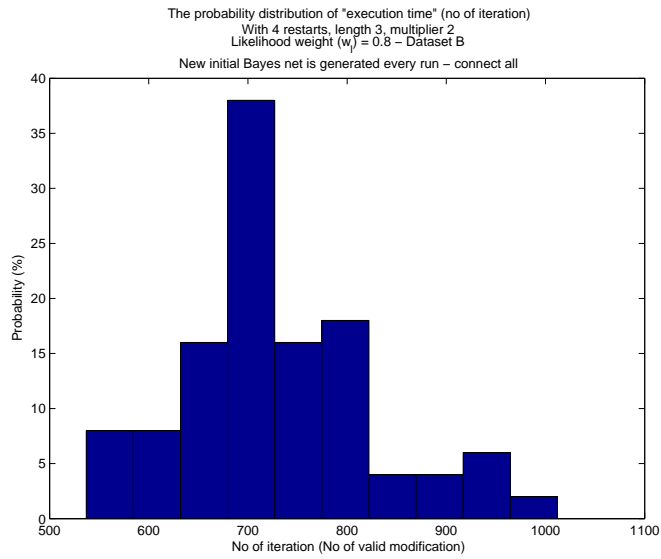


Figure 27: Histogram of the number of iterations of 100 runs of Greedy with Restarts for data B

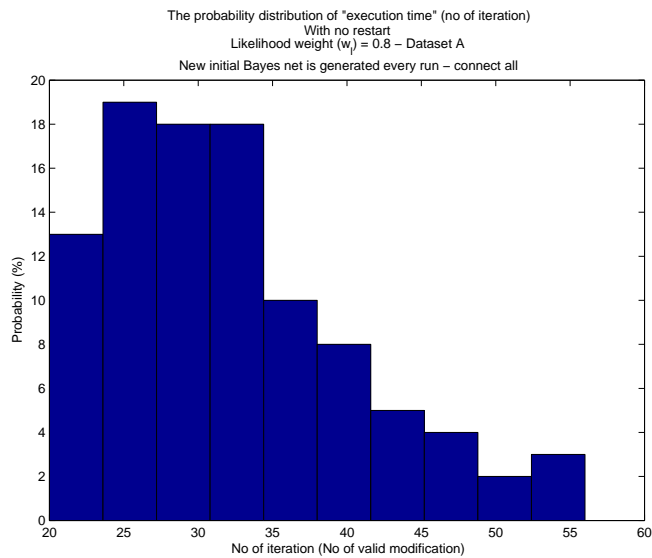


Figure 28: Histogram of the number of iterations of 100 runs of Greedy for data A

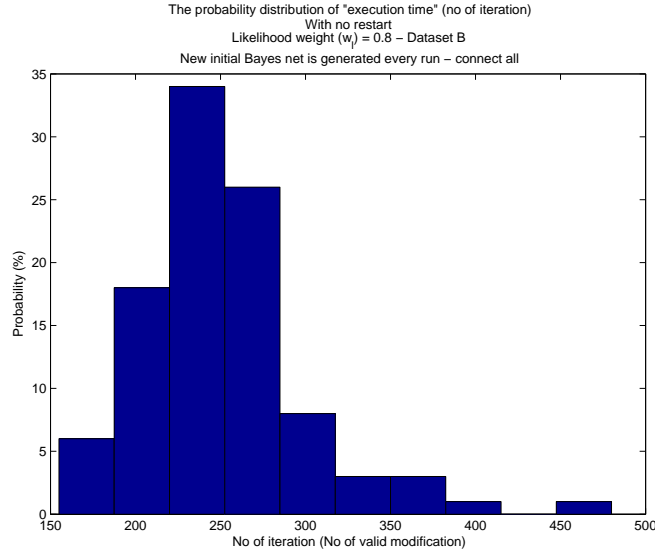


Figure 29: Histogram of the number of iterations of 100 runs of Greedy for data B

dataset B. We then ran our Greedy first-ascent hill climbing algorithm with each training dataset using initial networks generated from the connectAll method, while varying the weights of the BIC scoring function. Since we are just interested in the relative weighting of the two components of the BIC score, our experiments varied  $w_l$  in 0.1 increments from 0 to 1 while  $w_s$  varied as  $(1 - w_l)$ . Therefore we tried a total of 11  $w_l$ 's. In summary, we ran the Greedy algorithm 100 times for each variation of the scoring function, totalling 1100 runs.

After obtaining all 1100 output networks, we evaluated each output network on their respective Testing datasets using the Maximum Likelihood Score. We then averaged the Maximum Likelihood scores of the graphs produced with the same BIC score weighting, to obtain an average Likelihood Score for the graphs produced with a particular Score weighting. These average Likelihood Scores are plotted against their respective weighting  $w_l = 1 - w_s$  in Figures 30 and 31 for datasets A and B respectively. We chose to evaluate the graphs and the Testing datasets with the Likelihood score because we no longer care about the structural penalty. We already have a given network and we would like to evaluate how well it fits the data. The likelihood score calculates exactly this.

As can be seen in 30, for dataset A, as  $w_l$  increases, the Average Likelihood scores of the resulting graphs increase. The  $w_l$  that produces the highest average Likelihood score is  $w_l = 1$ . Therefore for dataset A, the best graph would be generated using a weighting of  $w_l = 0$  and  $w_s = 1$  which is exactly the Maximum Likelihood Score! Therefore it seems that dataset A requires a graph that is highly connected.

As shown in 31, for dataset B, the Average Likelihood score also increases as  $w_l$  increases. Therefore it also seems that the best graph is produced by using the greedy algorithm with the Maximum Likelihood Score.

We are skeptical about the results given in 30 and 31 because we would expect that graphs generated using the Maximum Likelihood Score would tend to overfit the Training data and therefore would not fit the Testing data very well. And we would also expect the graphs generated the some weighting of the BIC Score that does include a nonzero structural penalty to produce a graph that is more general so it will fit the Testing data better.

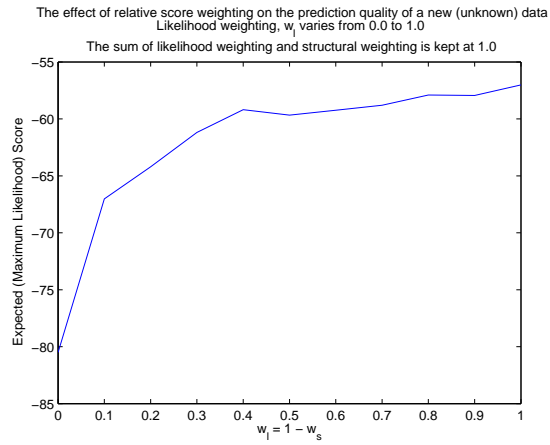


Figure 30: Data A: Average final scores obtained from the Greedy runs using different weights for the score.

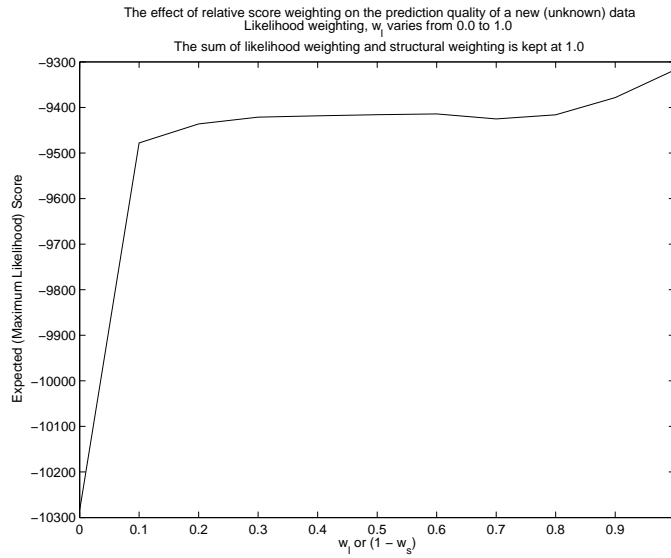


Figure 31: Data B: Average final scores obtained from the Greedy runs using different weights for the score.

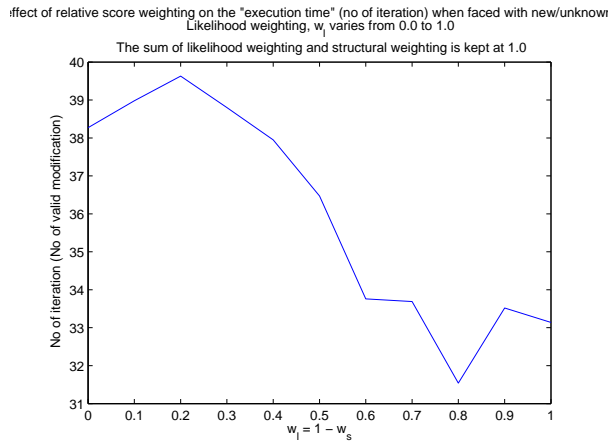


Figure 32: Average number of iterations for the Greedy runs against the weighting type for data A

Figures 32 and 33 plots the average number of iterations for the greedy runs against the type of weighting, for dataset A and B respectively. For Figure 32 and dataset A, it seems that the lower the  $w_l$ , the smaller the number of iterations it takes for the greedy algorithm to converge to a local maximum. This trend is also shown for data B in 33. We find this phenomena puzzling but very interesting. We are not sure why this occurs.

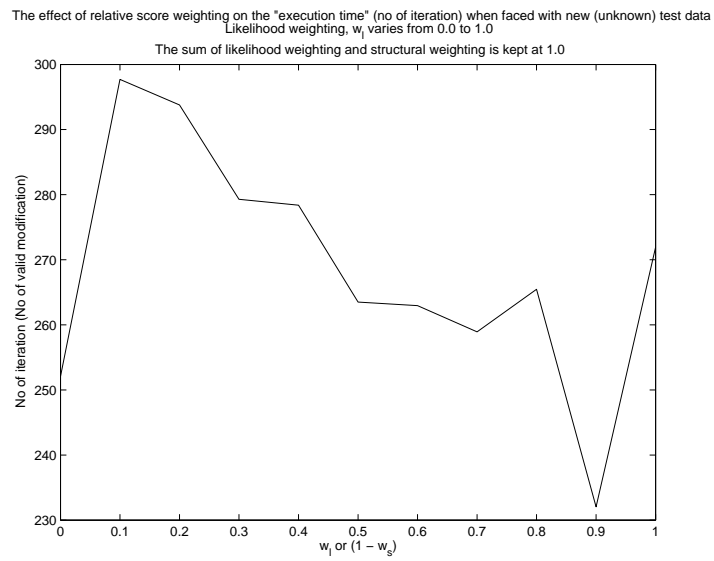


Figure 33: Average number of iterations for the Greedy runs against the weighting type for data B

## References

- [1] Koller and Friedman. Draft: Bayesian Networks and Beyond. Ch 14.