Finding Consensus DAG using Genetic Algorithm

Priyank Tiwari
priti063@student.liu.se

1 Introduction

Whenever a single expert is consulted for modeling a problem using Bayesian network, there is always a risk that the obtained Bayesian network structure (represented as DAG) is not accurate enough, either due to the expert bias or due to some details overlooked by the expert. Thus, to avoid such risks, multiple domain experts (or machine learning algorithms) are consulted and the multiple DAGs obtained from these experts are combined into a single consensus DAG. The consensus DAG is constructed in such a way that it represents independencies that are present in all the different DAGs obtained from different experts and also have least parameters associated with it. It has been proved that there exists multiple consensus DAGs and finding one of them belongs to NP-hard category of problems [Peña 2011]. In this project, we try to find the approximated consensus DAG using Genetic Algorithm and also study the effect of Genetic Algorithm parameters like initial population and number of generations on the best found Consensus DAG.

1.1 *Bayesian Network*

Bayesian Network or Belief Network is a graphical modeling technique used to represent knowledge about an uncertain domain. For example, a Bayesian network can be used to determine the probability of occurrence of illnesses by computing a relationship between the diseases and their symptoms.

The structure of a Bayesian network is represented via directed acyclic graph (DAG) where each node represents a random variable and edges between the nodes represent the probabilistic dependencies among the corresponding random variables. Thus, a directed edge from node X_i to X_j represents that the value taken by node X_j depends on the value of node X_i. Node X_i is referred as parent of node X_j and, similarly, node X_j is referred to as child of X_i.

1.2 *Node ordering*

Node ordering is defined as a linear ordering of vertices in a DAG such that, for every edge uv in DAG, u comes before v in the ordering. A DAG can have one or more valid node orderings. For example, the DAG shown in the figure below has following valid node orderings,

- 7, 5, 3, 11, 8, 2, 9, 10

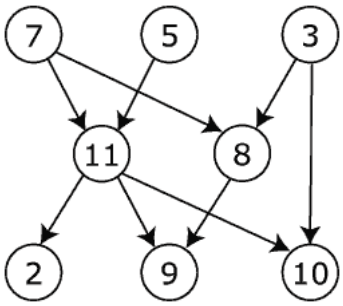

FIGURE 1: DIRECTED ACYCLIC GRAPH (DAG)

- 3, 5, 7, 8, 11, 2, 9, 10
- 3, 7, 8, 5, 11, 10, 2, 9
- 5, 7, 3, 8, 11, 10, 9, 2
- 7, 5, 11, 3, 10, 8, 9, 2
- 7, 5, 11, 2, 3, 8, 9, 10

1.3 *Independence Property of Bayesian Networks*

The independence that is encoded in a Bayesian network is that each variable is independent of its non-descendents given its parents. Thus, independence expressed in the example Bayesian Network shown in the Figure 2 below are as follows,

- Node C is independent of node B given node A (node A is parent of node C).
- Node D is independent of node C given node A and B.
- E is independent of A, B, and D given C.

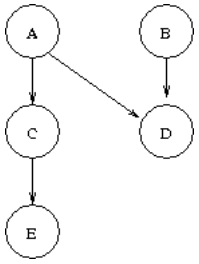

FIGURE 2: BAYESIAN NETWORK

The complete specification of probability distribution for a DAG with n random variables requires 2^n-1 joint probabilities. With the help of independence

property of Bayesian network, the number of joint probabilities required are reduced significantly.

2 Genetic Algorithms

According to Wikipedia description, A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution and generally used to generate approximate solutions to optimization and search problems.

In GA, population of individuals competes for survival. Each individual is designated by a set of genes that define its behavior. Individuals that perform better (as defined by the fitness function) have a higher chance of mating with other individuals. When two individuals mate, they swap some of their genes, resulting in an individual that has properties from both of its parents. Every now and then, a mutation occurs where some gene randomly changes value, resulting in a different individual. If all is well defined, after a few generations, the population converges on a "good-enough" solution to the problem being tackled.

A genetic algorithm implementation typically runs for a fixed number of iterations known as generations. During each generation,

- Performance of all individuals is calculated using fitness function and a fitness score is assigned to each individual. Higher the fitness score, bigger are the chances that its genes will be passed on to future generations using crossover.
- Individuals selected are paired up and new individuals are generated using crossover. The new individuals are then injected into population and take part in evolution during next generation.

3 Consensus DAG problem as a Genetic Algorithm optimization problem

As described in research paper by Peña, J. M., every DAG (G) has an associated node ordering, and one node ordering (say S_α) can be converted to another node ordering (say S_β) using method "Construct S_β" (refer table 2). The method "Construct S_β" returns the ordering S_β which is consistent with G and as close to S_α as possible. To convert a given graph G to another graph G_α which is consistent with given node ordering S_α, "Method B2" (refer table 3) is used which adds few more edges in the graph to achieve this.

The goal is to find an ordering 'O' of nodes, such that all the graphs are consistent with the ordering 'O' and total number of edges added by "Method B2", while converting the graphs to make them consistent with *'O'*, is minimum. Finding such an ordering *'O'* is NP-Hard in nature (refer Peña). With graphs of 10 nodes each, total number of orderings to be tested is 10! and grows exponentially with number of nodes. Thus, we resort to search heuristics like Genetic Algorithm.

The following steps outline the genetic algorithm approach that is used to obtain a consensus DAG from N different DAGs with V vertices each.

1. Randomly select a small predefined number of node orderings (O) out of $V!$ possible node orderings.

2. For each node ordering i in O,

   (a) Call Method B2(G_n, O_i), for n from 1 to N (number of DAGs)

   (b) Calculate the total number of edges (e) added by "Method B2" in step a). The inverse of number of edges added (e^{-1}) is the fitness cost for ordering O_i.

3. Select a set of node orderings from step 2 using Roulette selection operator.

4. Generate new node orderings using order crossover procedure (OX) and add them to the set of node orderings O.

5. Repeat step 2 unless fitness cost == 0 or predefined number of iterations is reached.

3.1 Roulette Selection

In Roulette selection method the chance of a node ordering getting selected to generate new node orderings using crossover is proportional to its fitness cost. Thus, higher the fitness cost, higher the chances of node ordering getting selected.

3.2 Order Crossover (OX)

Parent1:	1 2 *3 4 5 6* 7 8 9
Child:	7 9 *3 4 5 6* 1 2 8
Parent2:	5 7 4 9 1 3 6 2 8

Table 1: Order Crossover (OX)

The procedure of generating a new child ordering from parent orderings using Order Crossover is as follows,

1. Select a substring of nodes from a parent at random. Assume "*3 4 5 6*" (shown in blue) is selected at random from Parent1 in Table 1.

2. Produce a Child by copying the substring obtained in step 1 from Parent1 into corresponding position of Child (shown in blue).

3. Delete the nodes from Parent2 which are already in substring obtained in step 1. Resulting sequence in Parent2 contains the nodes that are required by Child (Nodes *3, 4, 5, 6* gets deleted from Parent2 as shown in red).

4. Insert the remaining nodes from Parent2 into empty nodes in Child from left to right (Nodes 7, 9, 1, 2, 8 are inserted from Parent2 into empty places in Child from left to right).

Construct S_β (G, S_α)

/* Given a DAG G and a node ordering S_α , the algorithm returns a node ordering S_β that is consistent with G and as close to S_α as possible */

1 $S_\beta = \phi$
2 G' = G
3 Let A denote the rightmost node in S_α that is a sink node in G'
4 Add A as the leftmost node in S_β
5 Let B denote the right neighbor of A in S_β
6 If $B \neq \phi$ and $A \notin \mathrm{Pa}_G(\mathrm{B})$ and A is to the right of B in S_α then
7 Interchange A and B in S_β
8 Go to line 5
9 Remove A and all its incoming arcs from G
10 If $G = \phi$ then go to line 3
11 Return S_β

TABLE 2: CONSTRUCT S_β

Method B2(G, S_α)

/* Given a DAG G and a node ordering S_α , the algorithm returns G_α */

1 S_β=Construct S_β (G, S_α)
2 Let Y denote the rightmost node in S_α that has not been considered before
3 Let Z denote the right neighbor of Y in S_β
4 If $Z = \phi$ and Z is to the left of Y in S_α then
5 Interchange Y and Z in S_β
6 If $Y \rightarrow Z$ is in G then cover and reverse $Y \rightarrow Z$ in G
7 Go to line 3
8 If $S_\beta = S_\alpha$ then go to line 2
9 Return G

TABLE 3: METHOD B2

4 Method of Evaluation

To find out how well the genetic algorithm performs in finding the best Consensus DAG from 'N' number of DAGs, we compare the GA results to 2 other trivial methods of obtaining Consensus DAG which are as follows.

1. With N different DAGs, we have total N different orderings(one ordering associated with each DAG). We take an ordering 'X' of a DAG and convert all other N-1 orderings of remaining DAGs to 'X' using "Method B2" and

calculate the total number of edges added in the entire process (recall that Method B2 adds edges in the graph while conversion). The process is repeated for X = 1 to N and the ordering 'X' in which least number of edges are added is chosen as the best ordering for Consensus DAG.

2. If GA while looking for best ordering calculates cost for 'M' different orderings, then we select M different orderings at random from all the possible orderings, calculate edges added for each of the 'M' orderings and select the ordering which adds the least number of edges.

5 Technical Details

The Genetic Algorithm code to solve the Consensus DAG problem is written in Perl using modules Graph and AI::Genetic::Pro from cpan.org. Module Graph is used to perform all the general operations on graphs like adding/removing an edge, checking for cycles in graph, adding/removing vertices in graph, etc. Module AI::Genetic::Pro is used to perform genetic algorithm operations like adding initial population, mutating orderings, evolving orderings, etc. Caching mechanism provided by module AI::Genetic::Pro is used to speed up the fitness score calculations.

6 Results

To evaluate the results, we designed a simulation with 10 iterations. In each iteration,

1. We generate 3 random DAGs with 10 nodes each.
2. Calculate the best ordering using Genetic Algorithm by selecting appropriate values for parameters like number of generations and initial population.
3. Calculate best ordering using other two methods described in section 4 above
4. Plot the scores on graph and repeat unless iterations == 10

In charts displayed below,

1. X-axis represents the iteration
2. Y-axis represents the best score which is the inverse of least number of edges added. This is because the genetic algorithm Perl library that we use for simulation *(AI::Genetic::Pro)* tries to obtain an ordering which maximizes the score and we want the library to select the ordering which minimizes the number of edges added. Thus, if three orderings O_1, O_2, O_3 add 2, 3 and 4 edges respectively, then their scores will be 2^{-1}, 3^{-1} and 4^{-1} respectively. The best score selected by the genetic algorithm library will be 2^{-1} as it is greater than 3^{-1} and 4^{-1}. Inverse of best score gives us the number of edges added.

3. GA Best Score represents the best score obtained by Genetic Algorithm library.

4. Random Best Score represents the best score obtained using random sampling method described in section 4 above.

5. G_0 Score represents the score obtained after converting the ordering of DAGs G_1 and G_2 to ordering of DAG G_0

6. G_1Score represents the score obtained after converting the ordering of DAGs G_0 and G_2 to ordering of DAG G_1

7. G_2 Score represents the score obtained after converting the ordering of DAGs G_0 and G_1 to ordering of DAG G_2

6.1 Genetic algorithm simulation run 1

To start with, we run the genetic algorithm simulation with initial population of *200* and evolve them for *30* generations. On an average, the total number of different node orderings evaluated by GA is *1700*. Thus, we select *1700* node orderings at random and evaluate them to compare against GA. Below is the graph obtained for simulation run 1.

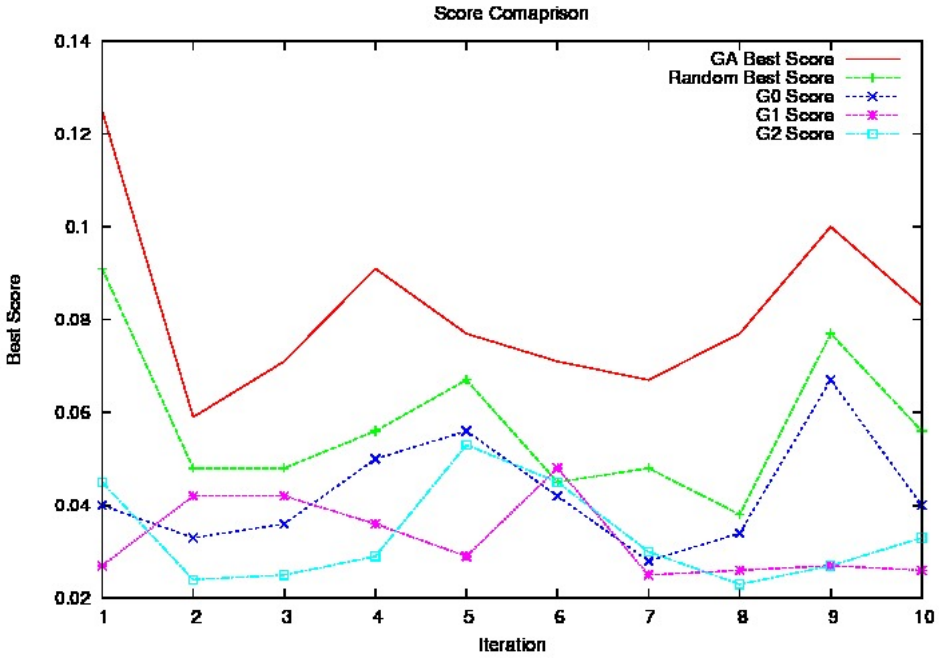

FIGURE 3: GENERATIONS = 30, POPULATION = 200, RANDOM = 1700

By looking at the graph above, we observe that GA finds better node ordering for consensus dag as compared to random selection method while evaluating the same number of total node orderings.

6.2 Genetic algorithm simulation run 2

For second simulation run, we select *300* initial population and evolve them for *7* generations which again evaluates on an average *1700* total orderings as in simulation run 1. We also evaluate *1700* node orderings at random for random selection method. Below is the graph obtained for simulation run 2.

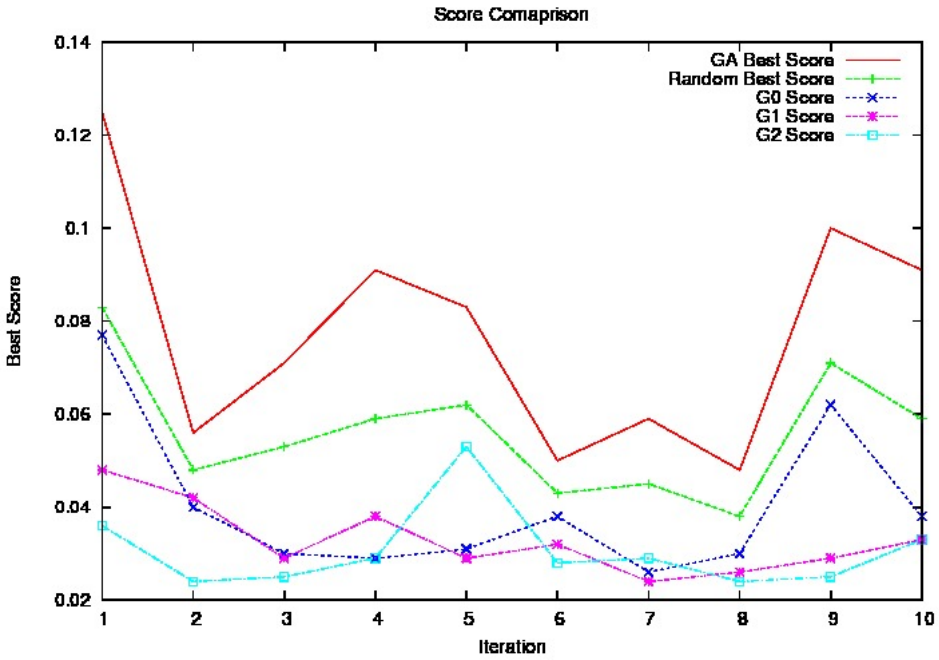

FIGURE 4: GENERATIONS = 7, POPULATION = 300, RANDOM = 1700

By looking at the graph above, we observe that GA finds better node ordering for consensus dag even if relatively large population is evolved for relatively small number of generations.

6.3 Genetic algorithm simulation run 3

For third simulation run, we take *500* initial population and evolve them for *20* generations. On an average, the total number of orderings evaluated by GA in this run is *4500*. To observe how well GA performs against random selection method, in this case we choose *10000* orderings at random i.e. double the amount of orderings evaluated by GA. Below is the graph obtained for simulation run 3.

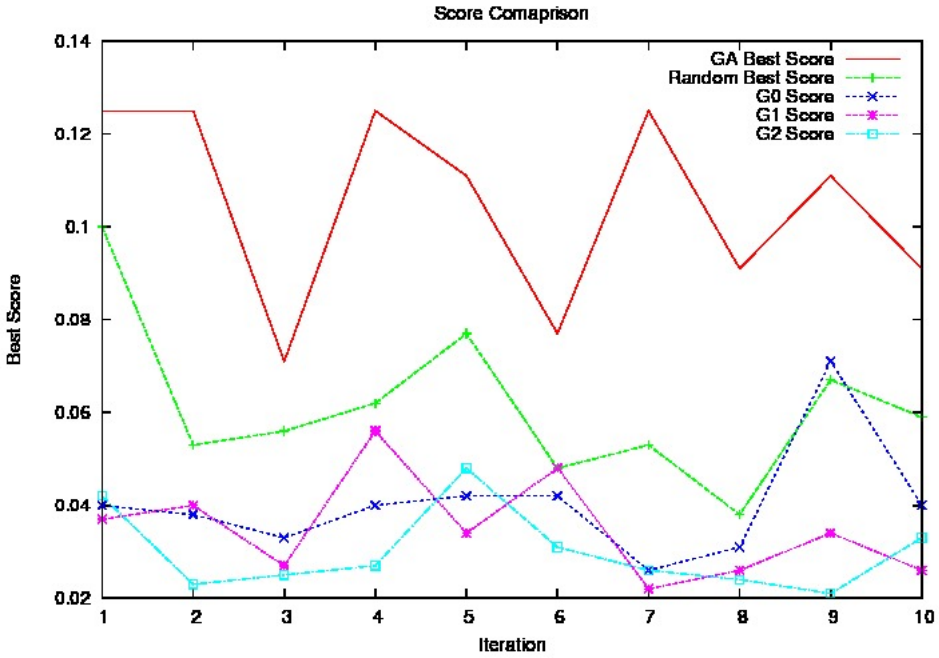

FIGURE 5: GENERATIONS = 20, POPULATION = 500, RANDOM = 10000

By looking at the graph above, we observe that even if we double the search space of random selection method, GA still finds better node ordering for consensus dag while evaluating only half the number of total orderings as compared to random selection method.

7 Conclusion

In all the simulations that we have run during the project, we observed that the genetic algorithm is a better choice when finding a consensus dag as compared to naive methods described in section 4 above. GA finds better consensus dag even when it evaluates less number of total orderings as compared to random selection method, which means that a better node ordering can be found in lesser amount of time.

8 Source Code

The Perl source code used for simulations can be found at `https://github.com/priyankt/Consensus-DAG`

9 Acknowledgement

Sincere thanks to Prof. Jose M. Peña for his continuous guidance and feedback during the entire course of the project.

10 References

1. Peña, J. M., *FINDING CONSENSUS DAG*, 2011

2. Nasir Hayat, Dr. Andrew Wirth, *Genetic Algorithms and Machine Scheduling With Class Setups,* IJCEM, May 1997

3. Wikipedia, *Genetic Algorithm,* *http://en.wikipedia.org/wiki/Genetic_algorithm*

4. Cpan, *Perl Graph Module*, `http://search.cpan.org/~jhi/Graph-0.94/lib/Graph.pod`

5. Cpan, *AI::Genetic::Pro Module,* `http://search.cpan.org/~strzelec/AI-Genetic-Pro-0.4/lib/AI/Genetic/Pro.pm`