Finding the Consensus DAG using Genetic Algorithms

Priyank Tiwari
priti063@student.liu.se

1 Introduction

Whenever a single expert is consulted for modeling a problem using a Bayesian network (BN), there is always a risk that the obtained BN structure (represented as directed acyclic graph) 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 directed acyclic graphs (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 the 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 the NP-hard category of problems [1]. Thus, in this project, we try to find an approximation to the consensus DAG using genetic algorithms (GA). We also study the effect of GA 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, suppose that there are two events which could cause grass to be wet: either the sprinkler is on or it's raining. Also, suppose that the rain has a direct effect on the use of the sprinkler (when it rains, the sprinkler is usually not turned on). Then the situation can be modeled with a BN as shown in figure (1). All three variables have two possible values, T (for true) and F (for false).

The joint probability distribution is given as:

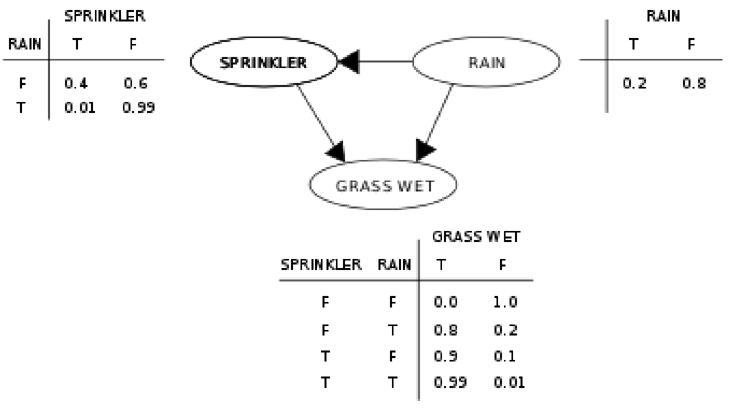
$$P(G, S, R) = P(G|S, R)P(S|R)P(R)$$

where G = Wet grass, S = Sprinkler, and R = Rain.

The model can be used to answer questions like "Given that the grass is wet, what is the probability that it is raining?" by using the conditional probability definition as follows:

$$P(R = T|G = T) = \frac{P(G = T, R = T)}{P(G = T)} = \frac{\sum\limits_{S\epsilon\{T,F\}} P(G = T, S, R = T)}{\sum\limits_{S,R\epsilon\{T,F\}} P(G = T, S, R)}$$

$$= \frac{(0.99 \times 0.01 \times 0.2 = 0.00198_{TTT}) + (0.8 \times 0.99 \times 0.2 = 0.1584_{TFT})}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0_{TFF}} \approx 35.77\%$$



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Figure 1: Example Bayesian Network [2]

The structure of a BN is represented as a DAG where each node represents a random variable and edges between the nodes represent the causal relationship among the corresponding random variables. A set of nodes that can be reached through a direct path from the node are called descendents.

1.2 *Node ordering*

A node ordering is defined as a linear ordering of vertices in a DAG such that, for every edge $u \rightarrow v$ 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 the following valid node orderings,

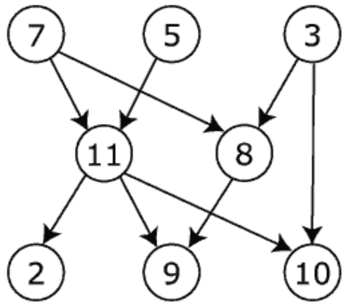


Figure 2: Directed Acyclic Graph (DAG)

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

Independencies present in the domain can be obtained from the DAG of a BN using a technique known as *"d-separation"* which is a graphical test of independence between variables in a DAG.

In *"d-separation"*, given two sets of variables A and B, we test if they are independent conditioned on a set Z of variables by checking all paths between each variable in A and each variable in B. We say that A is independent of B given Z (A⊥B|Z) if all paths between each variable in A and B are closed when we condition on (or in other words observe) Z. If any path is open, we cannot claim independence but also cannot claim dependence. We have to examine the conditional probability tables to verify the independence claims if there is no d-separation. A path is closed if there is any sequence of vertices and edges on it that are closed according to the rules mentioned in Table (1):

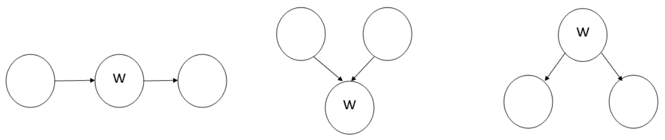


Figure 3: Example Sequential, Convergent and Divergent paths

	sequential	convergent	divergent
path is open	W∉Z	W∈Z or descendent(W)∈Z	W∉Z
path is closed	W∈Z	W∉Z or descendents(W)∉Z	W∈Z

Table 1: d-separation rules

Consider the example graph in Figure (4). On this graph, some of the independencies are as follows:

- $(Q \perp X, Y, Z, P \mid W)$: $Q \!\rightarrow\! W \!\rightarrow\! X$ is a divergent path and this is closed since we condition on W.
- $(Z \perp X, W, Q \mid \emptyset)$: $Z \!\rightarrow\! Y \!\leftarrow\! X$ is a closed convergent path since we do not condition on Y or its descendents P.
- $(Z, Y, P \perp W, Q \mid X)$: $W \!\rightarrow\! X \!\rightarrow\! Y$ is a closed sequential path since we condition on X.

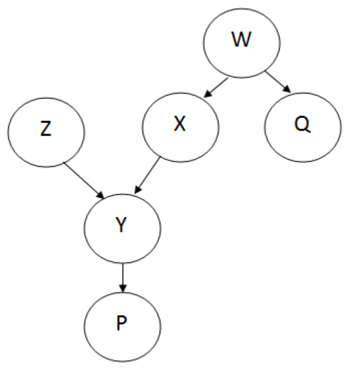


FIGURE 4: D-SEPARATION EXAMPLE GRAPH

1.4 *Independence Model and Minimal Directed Independence (MDI) Map*

- Independence model (M) is a set of variables from which we can determine whether the conditional independence relation $(X \perp Y \mid Z)$ is true for all possible triplets of disjoint subsets X, Y and Z derived from M.
- A graph G is said to be an I-map of the independence model M, if all conditional independence relations derived from G hold in M.
- A MDI map is a graph G_α which is an I-map of M with minimum edges. Thus, if we remove any edge from graph G_α, then it won't be an I-map of M.

2 Consensus DAG problem

Whenever a single expert is consulted for creating a BN for a domain, there is always a risk that the expert may miss out on some details or have bias over some parts of the domain. Thus, the obtained BN will not be an accurate representation of the domain. To minimize such risks, multiple experts are consulted to create a BN of the same domain and then all the generated BNs are combined into a single consensus BN. For the purpose of this project, we consider the definition of consensus DAG as the "DAG that represents the most independencies among all the MDI maps of the intersection of the independence models induced by the given DAGs" [1].

The consensus BN is generally obtained in the following two steps,

1. First, obtain the consensus BN structure from the BNs provided by multiple experts.
2. Second, obtain the consensus parameters for the consensus BN structure.

In this project, we restrict ourselves to the problem of obtaining a consensus BN structure from the given multiple BNs.

Let $G^1, .., G^m$ be the DAGs obtained from the multiple experts and α be any ordering of nodes that are present in the DAGs. Then, the consensus DAG can be obtained using following steps.

1. For each DAG G^i, find the MDI map G^i_α of DAG G^i relative to node ordering α.

2. Once the MDI map of all the DAGs are obtained, the consensus DAG is obtained by constructing the DAG whose arcs are exactly the union of arcs in $G^1_\alpha, .., G^m_\alpha$.

To obtain the MDI map of G^i relative to node ordering α, "Method B2" (3) is used . "Method B2" uses method "Construct S_α" (2) which takes a DAG G and a node ordering S_α as input and returns a node ordering S_β that is consistent with DAG G and as close to ordering S_α as possible. Refer to the work of J. M. Peña [1] for the detailed explanation of these methods.

It has been proved that the multiple non-equivalent consensus DAGs may exists for a given set of consensus DAGs and finding one of them belongs to the NP-hard category of problem [1]. The goal of this project is to find an approximated consensus DAG using GA search heuristic technique.

```
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 $S_\alpha$ that is a sink node in $G'$
4   Add $A$ as the leftmost node in $S_\beta$
5   Let $B$ denote the right neighbor of $A$ in $S_\beta$
6   If $B \neq \phi$ and $A \notin \mathrm{Pa}_G(B)$ and $A$ is to the right of $B$ in $S_\alpha$ then
7   Interchange $A$ and $B$ in $S_\beta$
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 $S_\beta$
```

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  $S_\beta$=Construct $S_\beta$ ($G$, $S_\alpha$)
2  Let $Y$ denote the rightmost node in $S_\alpha$ that has not been considered before
3  Let $Z$ denote the right neighbor of $Y$ in $S_\beta$
4  If $Z = \phi$ and $Z$ is to the left of $Y$ in $S_\alpha$ then
5      Interchange $Y$ and $Z$ in $S_\beta$
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

3 Genetic Algorithms

According to Wikipedia description [3], 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 GA 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.

4 Genetic Algorithm approach to find an approximate Consensus DAG

The algorithm mentioned in section (2) to obtain the consensus DAG is sensitive to node ordering S_α and will obtain the consensus DAG that respects the given node ordering. Thus, it is important to find the best node ordering S_α which minimizes the total number of edges added by the algorithm. This leads to an optimization problem where GA can help.

The following steps outline the GA approach that is used to obtain an approximate 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.

4.1 *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.

4.2 *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 4: 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 (4).
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).

5 Method of Evaluation

To find out how well the GA 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.

6 Technical Details

The GA code to solve the Consensus DAG problem is written in Perl using modules Graph [4] and AI::Genetic::Pro [5] 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 GA 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.

7 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 GA by selecting appropriate values for parameters like number of generations and initial population.

3. Calculate best ordering using other two methods described in section 5 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 GA Perl library [5] that we use for simulation 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 GA 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 GA library.

4. Random Best Score represents the best score obtained using random sampling method described in section (5) 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_1 Score 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

7.1 Genetic algorithm simulation run 1

To start with, we run the GA 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.

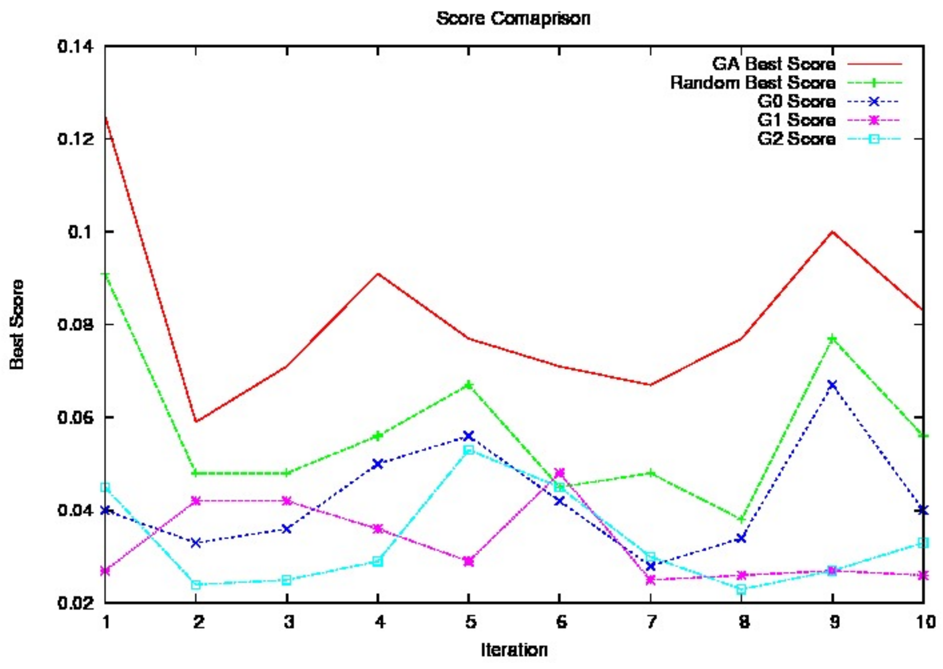


FIGURE 5: GENERATIONS = 30, POPULATION = 200, RANDOM = 1700

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.

7.2 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.

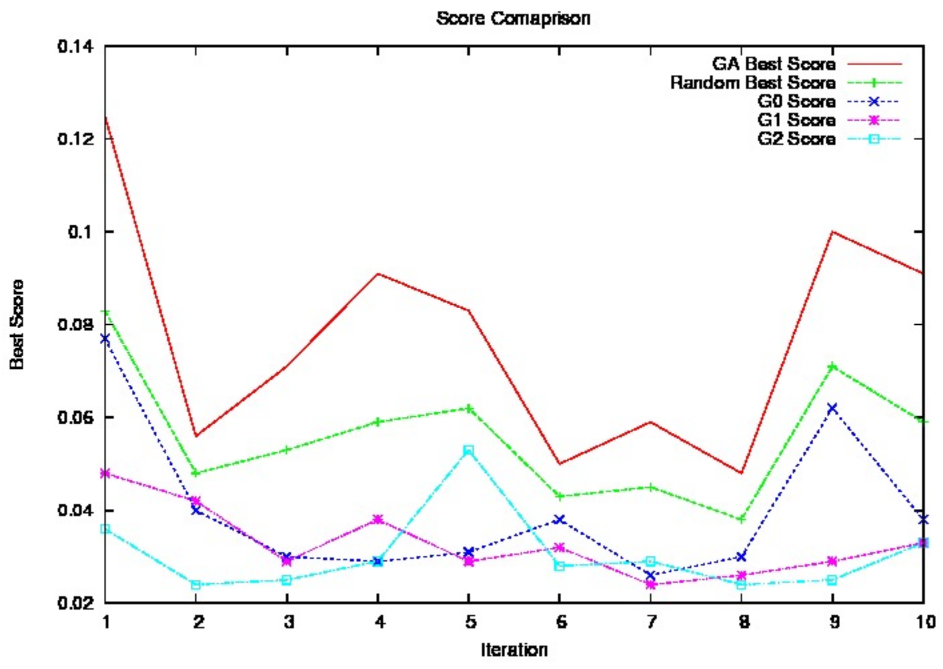


FIGURE 6: GENERATIONS = 7, POPULATION = 300, RANDOM = 1700

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.

7.3 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.

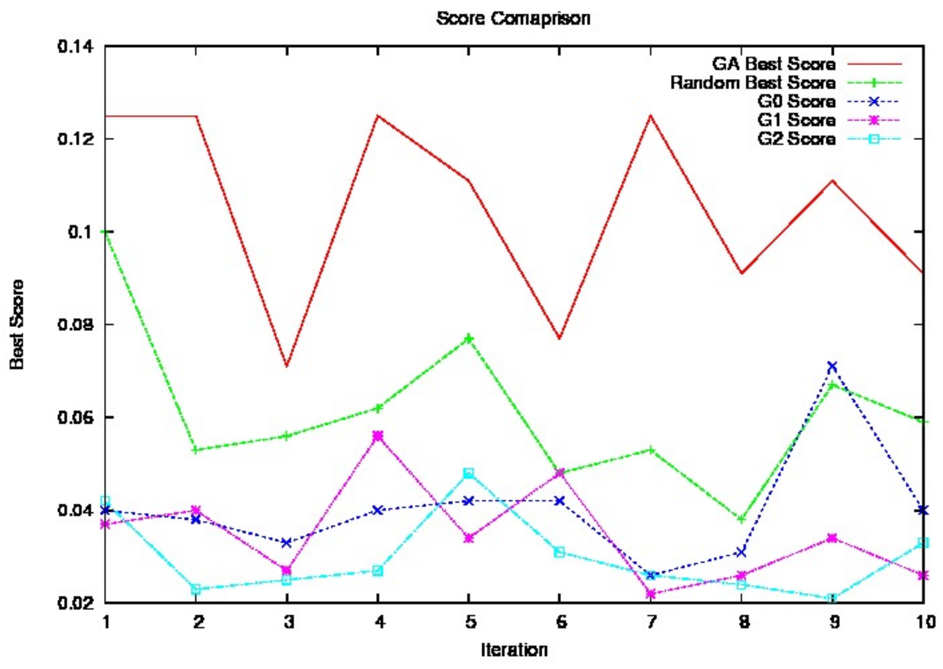


FIGURE 7: GENERATIONS = 20, POPULATION = 500, RANDOM = 10000

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.

8 Conclusion

In all the simulations that we have run during the project, we observed that the GA is a better choice when finding a consensus DAG as compared to naive methods described in section (5) 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.

9 Source Code

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

10 Acknowledgement

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

References

[1] J. M. Peña. Finding consensus bayesian network structures. *Journal of Artificial Intelligence Research*, 42(661-687), 2011.

[2] Bayesian network. `http://en.wikipedia.org/wiki/Bayesian_network`.

[3] Genetic algorithm. `http://en.wikipedia.org/wiki/Genetic_algorithm`.

[4] J. Hietaniemi. Perl graph module. `http://search.cpan.org/~jhi/Graph-0.94/lib/Graph.pod`.

[5] S. Lukasz. Perl genetic algorithm module. `http://search.cpan.org/~strzelec/AI-Genetic-Pro-0.401/lib/AI/Genetic/Pro.pm`.

[6] N. Hayat and A. Wirth. Genetic algorithms and machine scheduling with class setups. *International Journal of Computer and Engineering Management*, May 1997.

[7] E. Schreiber. d-separation. `www.cs.ucla.edu/~ethan/documents/dseparation.pdf`.

[8] J. Gutierrez. Probabilistic network models: Bayesian networks. `personales.unican.es/gutierjm/cursos/expertos/`.