CS5011 Assignment 3: Uncertainty

Student ID: 210034146
(Dated: April 1, 2022)

1: INTRODUCTION

1.1: Sections implemented

The following list describes the sections implemented in this assignment:

- Part 1 (Printing CPTs): Attempted and fully working.
- Part 2 (Making simple inferences): Attempted and fully working.
- Part 3 (Adding Evidence): Attempted and fully working.
- Part 4 (Deciding on an order): Attempted and fully working.

1.2: Running instructions

Steps to run the algorithm:

- Navigate to the src folder using CDM/terminal/command line.
- Type the following:
  java A3main $< part_{ID} > < Network_{ID} >$

1.4: Instructions: Stacscheck

The stacscheck is expanded from 21 test cases to a total of 51 test cases to validate part 4 of this assignment. The modified stacscheck can be run by following the steps below.

- Using CDM/terminal/command line navigate to the directory which contains the src and the Tests folders.
- Type the following to run the tests:
  stacscheck < test directory path >
- For example, in this case the tests folder is located in the starterCode directory (same directory as the src folder). Type the following:
  stacscheck /Tests

2: DESIGN AND IMPLEMENTATION

2.1: Brief introduction to Bayesian networks

A Bayesian network is a probability graph model which consists of nodes and directed edges connecting the said nodes. Where each node N_i describes the probability $P(N_i = z_i)$ or the conditional probability $P(N_i = z_j|N_k, N_e....))$ of the values (z_j) that node can take (z_j = True $\wedge$ False in this case). The direction of the edges describes the causal relations between the nodes; for example $Node_A \longrightarrow Node_B$ implies $P(B|A)$. The permutations of probabilities each node describes are known as a conditional probability table (CPTs for short). Bayesian networks take advantage of the emergent factoring property (see equation 1) of probability graph models /testbfi.e. the product of CPTs describes the joint probability distribution over all variables).

$$P(N_1, N_2, ...N_n) = \prod_{i=1}^{n} P(N_i|\text{parent}(N_i)) \qquad (1)$$

Given a joint distribution, a probabilistic inference to any query (within the faculty of the network) can be made by summing out irrelevant variables (**i.e.** every variable except the query variable).

2.2: Main Design & Implementation

The problem at hand involves creating different Bayesian networks and then making inferences to queries from the said network. As described above, an inference to any query can be made given the joint probability distribution. Hence the approach here involves creating nodes and their respective conditional probability tables and joining the said tables and summing over non-query variables in **order** (order being the most optimal joining sequence). The goal of obtaining an inference utilises five classes which are described in the following subsections.

2.2.1: Node

The $Node$ class plays a foundational role in implementing the Bayesian network. Each node contains five main attributes; Name, list of parent nodes, list of children nodes, neighbouring nodes (child + parent + co-parents), and a CPT object. The Node class has methods such as add child/parent, and remove child/parent, which is used to add/remove the edges between different nodes in a given BayesNet. Furthermore, the node class also has a getCPT method which returns the CPT. The addCPTvalues method feeds the CPT values to the CPT object (described later). Finally, the addNeighbor method adds an input node as a neighbour (used later for creating an undirected graph), and finally, the getNeighbor method returns a list of the undirected neighbour nodes (child + parent + co-parents) of the current node.

2.2.2: CPT

The CPT class has three attributes, the node associated with the CPT, a list of parents of the current node (for conditional dependence), and the conditional probability table. Given the current node and CPT values (from the addCPTvalues() method), the CPT class generates a 2d array describing the conditional probabilities of that node. Listing 1 describes the conditional probabilities of node M in the BNB network, which has two parents, K and L.

```
                  [K, L, M, Prob]
                  [T, T, T, 0.6]
                  [T, T, F, 0.4]
                  [T, F, T, 0.7]
                  [T, F, F, 0.3]
                  [F, T, T, 0.2]
                  [F, T, F, 0.8]
                  [F, F, T, 0.1]
                  [F, F, F, 0.9]
```

Listing 1: CPT: Node M.

2.2.3: Bayesian Network

The $BayesianNetwork$ class has one main attribute, which is a list of notes encompassing the network. This class has multiple methods, which are listed below.

- addNode(node): adds the input node to the network.
- addEdge(node1,node2): adds a directed edge from node1 to node2 (**i.e.** node1 is the parent of node2 and conversely node2 is the child of node1).
- removeEdge(node1,node2): removes the directed edge from node1 to node2

These methods, combined with the Node and CPT classes, form the backbone of the probabilistic graph model.

2.2.4: Order

The $Order$ class has one function called order which takes two inputs: a constructed Bayesian network object and the query (inference) variable. The order function outputs a list of nodes, which describes the order in which the nodes should be joined and summed out. Every order yields a valid algorithm, but finding the right (can be more than one) order improves the algorithm's efficiency (less time and space requirements) since an optimal order ensures the number of joining and elimination processes are kept to a minimum.

The order function uses the maximum cardinality search technique to find the right order. This technique works by converting the network into an induced graph (**i.e.** parents of a node are linked, see figure 1). Listing 2 describes how the directed network was converted into an induced graph, and listing 3 illustrates the maximum cardinality search technique.

```
for (Node node: Nodes){
    // get parents of current node
  ArrayList<Node> parents = node.getParents();
   // iterate over parents
  if (parents.size() > 1){
    for (Node parent_x:parents){
      for (Node parent_y:parents){
        if (!parent_x.equals(parent_y)){
            // if parents not already linked
          if (!parent_x.getNeighbor().contains(
    parent_y)){
          // link parent nodes
          parent_x.addNeighbor(parent_y);
          }
        }
      }
    }
  }
}
```

Listing 2: Generating an induced graph.

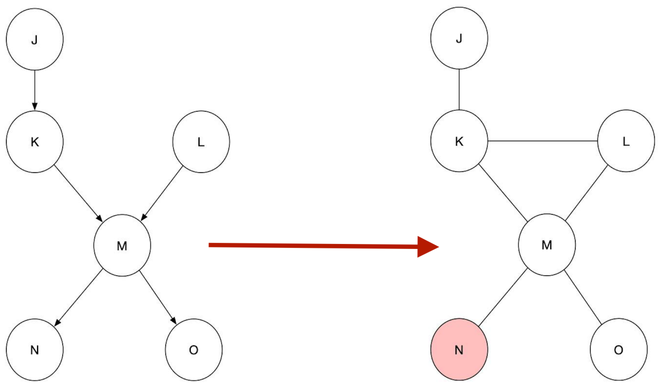


FIG. 1: Converting a regular directed network (on the left) into an induced graph network (on the right). taken from [1].

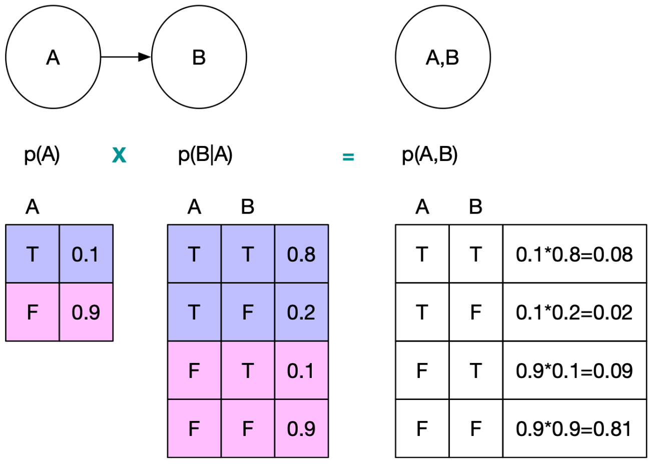
```
ArrayList<Node> unmarked = new ArrayList<Node>(
    Nodes);
ArrayList<Node> marked = new ArrayList<Node>();

// selecting starter node = query node
unmarked.remove(query_node);
marked.add(query_node);

// generating order, looping till unmarked list is
    empty
while (unmarked.size() > 0){
  int max_marked_neighborCount = 0;
  Node max_marked_node = null;
  // iterating over unmarked neighbours
  for (Node unmark : unmarked){
    ArrayList<Node> temp_neighbours = unmark.
    getNeighbor()
    // finding common nodes
    temp_neighbours.retainAll(marked);

    // selecting nodes with max marked neighbours
    if (temp_neighbours.size() >
    max_marked_neighborCount){
      max_marked_neighborCount = temp_neighbours.
    size();
      max_marked_node = unmark;
    }
  }
  // adding unmarked node with max marked
    neighbours to marked
  marked.add(max_marked_node);
  unmarked.remove(max_marked_node);
}

marked.remove(starter_node);
// reverse list
Collections.reverse(marked);
```

Listing 3: Maximum cardinality search.

2.2.5: Solver

The solver class has four methods that, when combined, can generate inferences to queries.

- joinCPTs(): This method takes in two CPTs and joins them into a single CPT. This is done by comparing the common labels between the two CPTs, and creating a third CPT, whose entries are the product of the entries of the two input CPTs where the values of the common labels are the same. Figure 2 describes the joining operation.



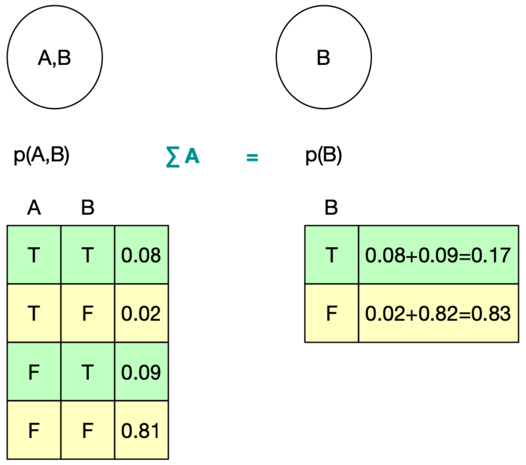
A	
T	0.1
F	0.9

A	B	
T	T	0.8
T	F	0.2
F	T	0.1
F	F	0.9

A	B	
T	T	0.1*0.8=0.08
T	F	0.1*0.2=0.02
F	T	0.9*0.1=0.09
F	F	0.9*0.9=0.81

FIG. 2: Join CPT operation. taken from [1].

- elimination(): This function takes in a single CPT and a label (node's name) and sums over the given label to produce a new smaller CPT, effectively eliminating the variable. Figure 3 describes the elimination operation.



A	B	
T	T	0.08
T	F	0.02
F	T	0.09
F	F	0.81

B	
T	0.08+0.09=0.17
F	0.02+0.82=0.83

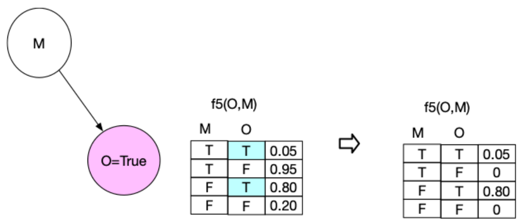
FIG. 3: Elimination operation. taken from [1].

- prune(): Given a user-defined or a generated order, the prune function takes the Bayesian Network object and a list of query nodes (query + evidence nodes) as input. The prune function removes every variable that is not an ancestor of a queried variable or an ancestor of the evidence nodes. Listing 4 describes the pruning process.

```
prune(BayesianNetwork, queries){

  HashMap Fmap = new HashMap();
  HashMap Bmap = new HashMap();

  for (Node node: bn.getNodes()){
    Fmap.put(node.getName(), node);
    Bmap.put(node, node.getName());
  }

  // adding query and evidence nodes to the stack
  Stack<> stk = new Stack<>();
  for (String query : queries){
    stk.push(Fmap.get(query));
  }
  // list of relavant nodes (currently empty)
  ArrayList<> relavant = new ArrayList<>();
  // iterate over the stack till stack is empty
  while (!stk.empty()){
    Node current = stk.peek();
    // parents of current node
    ArrayList<> parents = current.getParents();
    // add current node to relevant node list
    relavant.add(current);
    // remove current node from the stack
    stk.pop();
    // add parents of the current node to stack the
    for (Node parent:parents){
      stk.push(parent);
    }
  }
  Set<> set = new HashSet<>(relavant);
  relavant.clear();
  relavant.addAll(set);

  return relavant;
}
```

Listing 4: Prune nodes.

- solver(): The final function of the solver class takes in the four inputs; the network name (generates the network accordingly), the query variable, the order given (generates order if no order is given), the evidence. This function starts by pruning the order using the prune method described above. After pruning, CPTs from the relevant nodes are added to a list known as factors. If the conditional evidence (**i.e.** the conditional variables and their observed values) is provided, then all entries of all CPTs which contain the given conditional variables but do not match the given observation are fixed to 0 (see figure 4).



M	O	
T	T	0.05
T	F	0.95
F	T	0.80
F	F	0.20

M	O	
T	T	0.05
T	F	0
F	T	0.80
F	F	0

FIG. 4: Given evidence $O = T$, rows of all CPTs which contains the variable O but does not match the given observation ($O \neq T$) are fixed to 0. taken from [1].

Once the factors list is filled with the appropriate CPTs, a for loop is used to iterate over each label in the pruned order list. For each variable, CPTs containing the said variable are removed from the factors list and joined into a single CPT using the joinCPTs() method. The variable is eliminated using the elimination() method. This updated CPT is added back to the factors list, and the process repeats until all variables in pruned order list have been iterated over. The final CPT (which should only contain the query variable if implemented correctly) is then normalised, and the desired inference is then printed to the screen.

3: PARTS

3.1: Part 1

The first part of this practical involves creating different Bayesian Networks and printing out the CPTs (see listing 1). Figure 5 describes the Bayesian Network for the CNX system. The following subsections describe the assumptions that went into creating the relationships between the CNX nodes and their respective CPTs.

3.1.1: Outdated maintenance information

As described in the spec sheet, the probability that the maintenance information is outdated is $P = 0.02$.

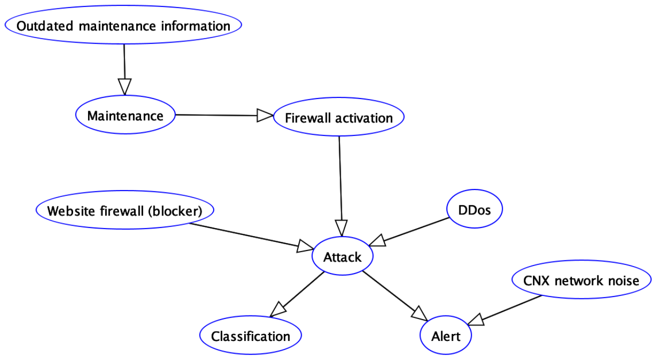


FIG. 5: Bayesian Network: CNX system.

3.1.2: Maintenance

Since the maintenance depends on the maintenance information, an assumption is made that the maintenance is scheduled to once month ($P = \frac{1}{12}$) if the maintenance information is not outdated, and the maintenance is scheduled to once a week ($P = \frac{4}{12}$) if the maintenance information is outdated.

3.1.3: Firewall activation

Given that the firewall depends on the maintenance, if the system is undergoing maintenance, the probability of the firewall being active is $P = 0.97$. An assumption is made that the firewall is always active $P = 1$, when no maintenance occurs.

3.1.4: Unsafe website blocker

The website blocker blocks malicious websites with a probability of $P = 0.85$.

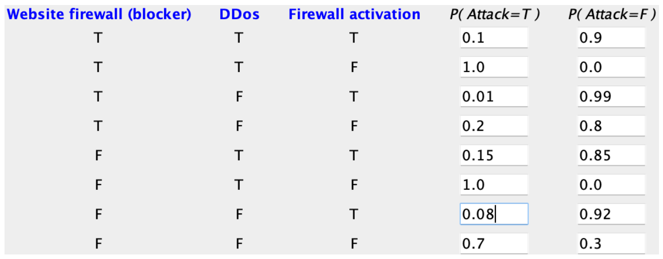
3.1.5: DDoS attack

DDoS attacks peak during holidays, and given that the system is located in a place with 45 holidays, the probability of a DDoS attack is $P = \frac{45}{365}$.

3.1.6: Attack

The system can be attacked from two places, by visiting malicious websites and from DDoS attacks. The presence or absence of a firewall makes a difference. For this reason, the attack node is causally connected to the firewall node, the DDoS node, and the website blocker node. The assumed CPT is described in figure 6.

The general idea is that the probability of an attack is low when the firewall is active, and conversely, the probability of an attack is high when the firewall is inactive. The DDoS



Website firewall (blocker)	DDos	Firewall activation	P(Attack=T)	P(Attack=F)
T	T	T	0.1	0.9
T	T	F	1.0	0.0
T	F	T	0.01	0.99
T	F	F	0.2	0.8
F	T	T	0.15	0.85
F	T	F	1.0	0.0
F	F	T	0.08	0.92
F	F	F	0.7	0.3

FIG. 6: CPT: attack node.

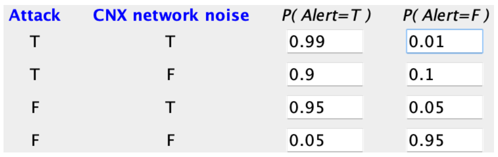
attack (DDoS = T) is given a larger weight than the website blocker (blocker = false). This is because DDoS attacks are more targeted than attacks from malicious websites (which are more generic).

3.1.7: CNX noise

The probability of noise being present in the CNX network is assumed to be $P = 0.01$

3.1.8: Alert

The alert system can be triggered in the presence of an attack or noise in the system, or both. The assumed CPT table for the alert node is shown below.



Attack	CNX network noise	P(Alert=T)	P(Alert=F)
T	T	0.99	0.01
T	F	0.9	0.1
F	T	0.95	0.05
F	F	0.05	0.95

FIG. 7: CPT: attack node.

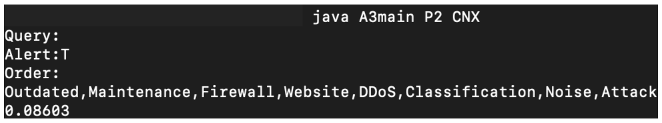
3.1.9: Classification

Given an attack on the system, the classification node differentiates logs by classifying whether an activity was normal or anomalous. This system accurately classifies logs with a probability of $P = 0.7$ (misclassification of $P = 0.3$)

3.2: Part 2: Making simple inferences

For part two, simple inferences are made by providing a query and an order. In this case, no evidence is provided (CPTs are not modified to fix certain entries to 0), and the system generates no order.

The following query is fed to the CNX network ($P(Alert = T)$). This returns the probability of the system being alerted. Belief and Decision Network Tool Version 5.1.10 was used to cross-validate the algorithm's output. For this query, the Decision Network Tool outputs the probability of 0.086; meanwhile, figure 8 describes the algorithm's output. The result does not change by changing the order; however, inefficient orders take longer to compute.



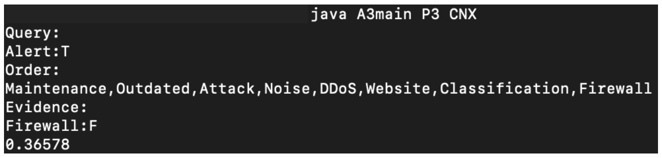
```
java A3main P2 CNX
Query:
Alert:T
Order:
Outdated,Maintenance,Firewall,Website,DDoS,Classification,Noise,Attack
0.08603
```

FIG. 8: CNX: $P(Alert = T)$

3.3: Part 3: Adding Evidence

Part 2 of the algorithm deals with evidence. The user provides the query, the order, and the evidence. As discussed before, when the user provides the evidence, the CPTs are modified such that rows in CPTs containing the evidence variables which do not match the observation are set to zero.

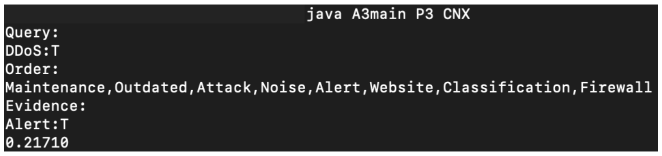
Two queries are fed into the CNX network. The first one is a predictive query (direction of reasoning moves from the parents to the children); here, the following query is asked: $P(Alert = T|Firewall = F)$. The decision network tool outputs a probability of 0.37 to this query, and the algorithm's output can be seen in figure 9.



```
java A3main P3 CNX
Query:
Alert:T
Order:
Maintenance,Outdated,Attack,Noise,DDoS,Website,Classification,Firewall
Evidence:
Firewall:F
0.36578
```

FIG. 9: CNX: $P(Alert = T|Firewall = F)$

The second query to the network is a diagnostic query (direction of reasoning moves from the children to the parents); here, the following query is asked: $P(DDos = T|Alert = T)$. The decision network tool outputs a probability of 0.22 to this query, and the algorithm's output can be seen in figure 10.

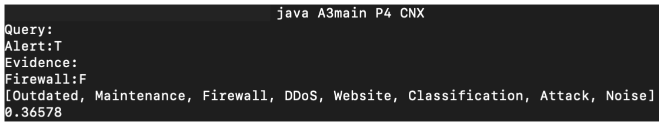


```
java A3main P3 CNX
Query:
DDoS:T
Order:
Maintenance,Outdated,Attack,Noise,Alert,Website,Classification,Firewall
Evidence:
Alert:T
0.21710
```

FIG. 10: CNX: $P(DDos = T|Alert = T)$

3.4: Part 4: Deciding on an order

As discussed before, if no order is given, the algorithm generates the order by using the maximum cardinality search technique described in section 2.2.4. The order does not affect the result for the CNX network but affects the computational time to respond to the said queries. For the CNX network, non-optimal orders tend to have more factors to join until all non-query variables have been factored out. The p4 part of the Algorithm outputs the order and the inference to the query. For example, parsing the same query as in part 3 $P(Alert = T|Firewall = F)$ yields the following output.



```
                    java A3main P4 CNX
Query:
Alert:T
Evidence:
Firewall:F
[Outdated, Maintenance, Firewall, DDoS, Website, Classification, Attack, Noise]
0.36578
```

FIG. 11: CNX: $P(DDos = T|Alert = T)$

4: TEST SUMMARY

The system passes 21 out of 21 stacscheck test cases provided on studres. However, the stacscheck only validates the P2 and P3 parts of the algorithm for networks BNA, BNB, and BNC. For this reason, the stacscheck is expanded to validate P4 and the queries from the CNX network on P1, P2, P3, and P4. Ultimately resulting in a total of 51 test cases. The system passes 51 out of 51 stacscheck test cases.

5: CONCLUSION

The implementation of the Bayesian Network algorithm here works for all networks, as it has been shown to pass all test cases. Part 1 prints the outputs of the CPT, and part 2 works with simple inferences given an order. Part 3 takes into account the given evidence and order. Part 4 only considers the given evidence and generates an order using the maximum cardinality search technique described in section 2.2.4.

REFERENCES

[1] *CS5011: A3 - Uncertainty*, School of Computer Science, University of St Andrews, (2022)