Search, Planning, and Machine Learning
Block 3: Bayesian Network

Chantal van Duin (s1004516) & Nayeong Kim (s1006313)

December 2018

1 Specification

This report will look into an implementation of how inference works in Bayesian networks. This is done using an implementation of the Variable Elimination algorithm and applying it onto different inference queries in various Bayesian networks using the programming language Java and the programming program Eclipse. The performance and evaluation of the Variable Elimination algorithm is highly influenced by the approach of how the order - in which the variables are eliminated in the algorithm - is decided. This report however will not focus on this aspect of efficiency of implementation of the algorithm but rather focuses on the inner workings and results of the algorithm.

2 Design

2.1 Basic design and idea of VE - algorithm

The goal of Variable Elimination algorithm is to calculate the inference by using the joint distribution where variables of the network has been summed out to create the marginal joint distribution of network. This summing out is made possible by using Factors. Factors can be used to represent a probability distribution. There are two operations needed to manipulate the factors to be able to sum out variables out of distributions : factor product and factor marginalization. Factor product is used to create the factor of the union of two other factors. Factor marginalization is used to sum out a variable from a factor. Factor reduction is used to change the factor to only have the observed values of the observed variables in the factor. See the pseudo code for factor operations in subsections 2.2 till 2.4.
Variable Elimination algorithm sums out the variables of the joint distribution is by eliminating out every variable in a specified elimination order and reducing the factors for the observed values. See pseudo code for variable elimination algorithm and deciding elimination order in subsections 2.5 and 2.6. For reading in the Bayesian network and user input, the provided code attached to assignment was used throughout the implementation of the VE algorithm.

2.2 Pseudo code Marginalization

```
function MARGINALIZATION(v) returns a factor F'
   input:     v, the variable which will be deleted
```

```
newVariables <- list of new variables, initially empty
newRows <- list of new probability rows with values, initially empty
deletedVar <- the index number of the variable that is equal to given v in variable list
    of the factor / f.V, initially -1

if size(f.V) = 1 then :
   add new row r([], SUM-ALL-PROB()) to newRows
   and return new factor F'(V, newRows)

for each row r1 in Table of F :
   if r1 is not visited yet then :
      for each row r2 in T but not r1 :
         if CAN-SUM(r1, r2) / check if r1 and r2 are equal then :
             prob = r1.prob + r2.prob
             add r1 and r2 to visited
    newValues <- MAKE-NEW-VALUES(r1, deletedVar)
    add new row r(newValues, prob) to newRows

for each variable var in V :
   if index(var) != deletedVar then :
       add var to newVariables

return new factor F'(newVariables, newRows)
```

2.3 Pseudo code Product

```
function PRODUCT(F) returns a factor F'
   input:     F, the factor to be multiplied

   newVariables <- list of new variables, initially empty
   newProb <- list of new probability rows with values, initially empty
   idx1, idx2 <- integer list of the index number of variable shared by two factors

   for each variables v1 in V and v2 in F.V :
      if v1 = v2 then :
          add each indices to idx1 and idx2 respectively

   for each rows r1 in T and r2 in F.T :
      if canProduct(r1, r2, idx1, idx2) /check if the shared variable has the same value,
          then :
         newValues = makeNewValue(r1, r2, idx2)
         prob = r1.prob * r2.prob
         add new row r(newValues, prob) to newProb

   newVariables <- addAll(V)

   for each variable var1 in V and var2 in F.V :
      if var1 != var2 then :
          add var2 to newVariables
```

```
  return new factor F'(newVariables, newProb)
```

2.4 Pseudo code Reduction

```
function REDUCTION (ob) returns a factor F'
    input:     ob, observed variable to be reduced

  newVariables <- list of new variables, initially containing V / list of variables of
      original factor
  newRows <- list of new probability rows with values, initially emptty
  index <- index number of variable that is equal to the observed variable, initially -1

  for each variable v1 in newVariables :
     if v1 = ob then :
          index = i;

     for each row r1 in Table T:
        if value of row at shared index = observed value of ob :
           add r1 to newRows

  return new factor F'(newVariables, newRows)
```

2.5 Pseudo code VE - algorithm

```
function ALGORITHM ()

    order <- DECIDE-ORDERING() decide order of variables to eliminate
    factorList <- VARIABLES-INTO-FACTOR () turn needed variables into factors

    for each factor f in factorList :
        for each observed variable ob :
            if factor contains ob then :
                facor newF = f.REDUCE(ob)
                remove f from factorList
                add newF to factorList

    for each variable v in order :
        factorListProduct <- list of factors needed to be multiplied, initially empty
        toRemove <- list of factors needed to be removed, initially empty

        for each factor f in factorList:
            if factor f contains variable v then :
                add f to factorListProduct and toRemove

        while size(factorListProduct) is not 1:
            factor newF = PRODUCT factors in factorListProduct
            remove f in factorListProduct
            add newF to factorListProduct
        factor newF = MARGINALIZE(factor in factorListProduct)
        add newF to factorList

        for each factor f in toRemove :
            remove f in factorList

        if query is earlier in network than observed value / ORDER-METHOD-OBV-FIRST() then :
            while factorList is not empty :
                factor newF = PRODUCT factors in factorList
                remove f in factorList
                add newF to factorList

    normalisedTable = NORMALISE () table of only factor in factorList
    print normalisedTable
```

2.6 Basic Concept and Pseudo code Elimination order

The idea for the elimination order was to first eliminate the variables that have the smallest number of parent nodes and the variables with highest number of parent nodes last.

```
function DECIDE-ORDERRING()
    order <- priority queue prioritizing smallest number of parents
   parentsOfQuery <- PARENTS-OF-QUERY () arrayList of variables containing parent nodes of
        the query
   parentsOfObs <- OBSERVED-VALUES () arrayList of variables containing parent nodes of the
        the observed values

   for each variable v in basian network :
        if query is earlier in network than observed value / CHECK-ORDER-METHOD() then :
          if v is not query and parentsOfQuery contains v then:
              add v to order
      else then :
         for each observed variable ob
            if v equals ob then :
                add v to order
            if v is not query and parentsOfObs contains v then :
                add v to order

   return order
```

3 Implementation

3.1 class: Factor

3.1.1 Factor Class and Constructor

A factor is considered as a multi-dimensional table assigning a value to each assignment of a set of variables. The variables that a factor must have in order to execute a VE algorithm are:

```
private ArrayList<Variable> variables;
private ArrayList<ProbRow> table;

public Factor(ArrayList<Variable> variables, ArrayList<ProbRow> table) {
   this.variables = variables;
   this.table = table;
}

public Factor(Variable var, ArrayList<ProbRow> table) {
   ArrayList<Variable> variables = new ArrayList<Variable>();
   variables.add(var);
   if (var.getNrOfParents() != 0)
      variables.addAll(var.getParents());
   this.variables = variables;
   this.table = table;
}
```

3.1.2 Factor Marginalization

Marginalization can be done in two cases depending on the size of the variable in factor.

1. Case 1: when the factor has only one variable

```
 if (variables.size() == 1) {
   ArrayList<String> newValues = new ArrayList<String>();
   newRows.add(new ProbRow(newValues, sumAllProb())); //-------> F.2.1.1

   return new Factor(variables, newRows);
}
```

F.2.1.1 Let v be a variable when there is only variable in factor. In this case, there two rows in the table and each row would represent the probabilities when the value of v is True and False, respectively. Then, the sum for v would be 1 and there are not any values in table after marginalization. To make a new factor, add a new probability row with empty value list and the sum of all probabilities in table. The sum can be get by applying sumAllProb(), which adds all probabilities for each row in table.

2. Case 2: there are more than one variables in the factor

```
int deletedVar = -1;
for (int i = 0; i < variables.size(); i++)
   if (variables.get(i).equals(v)) //-------> F.2.2.1
      deletedVar = i;
```

F.2.2.1 Check if the factor contains the variable given or not. If so, put the index of variable to be deleted into deletedVar.

```java
ArrayList<Integer> checkedIdx = new ArrayList<Integer>();
for (int i = 0; i < table.size(); i++) { //-------> F.2.2.2
   if (!checkedIdx.contains(i)) {
      checkedIdx.add(i); //-------> F.2.2.3
      Double prob = table.get(i).getProb();
      for (int j = i + 1; j < table.size(); j++) {
           if ( canSum(table.get(i), table.get(j), deletedVar) ) {
           //-------> F.2.2.4
             checkedIdx.add(j);
             prob += table.get(j).getProb();
         }
      }
   ArrayList<String> newValues = new ArrayList<String>();
   newValues = makeNewValues(table.get(i), deletedVar); //-------> F.2.2.5
   newRows.add(new ProbRow(newValues, prob));
   }
}
```

F.2.2.2 Compare each two rows in factor.

F.2.2.3 First, check if the given row was visited already. If the value is not visited yet, add the index number to list.

F.2.2.4 Check if two probability rows can be sum by comparing two rows' values are same. If so, add the index number of another row also to the list for avoiding row duplicated and calculate new probability by adding their probabilities.

F.2.2.5 Make a new value list containing all values in factor without the value in deletedVar.

```java
for (int i = 0; i < variables.size(); i++)
   if (i != deletedVar)
      newVariables.add(variables.get(i)); //-------> F.2.2.6
return new Factor(newVariables, newRows);
```

```java
public ArrayList<String> makeNewValues(ProbRow p, int deletedVar) {
   ArrayList<String> newValues = new ArrayList<String>();
   for (int i = 0; i < p.getValues().size(); i++)
      if (!(i == deletedVar))
         newValues.add(p.getValues().get(i));

   return newValues;
}
```

F.2.2.6 Then, delete the variable given in the list of variables of factor. The function, makeNewValues, is to make new string list of values without the value in the position of deleted variable

3.1.3 Factor Product

1. Make a list for common variables

```java
ArrayList<Integer> idx1 = new ArrayList<Integer>();
ArrayList<Integer> idx2 = new ArrayList<Integer>();

for (int i = 0; i < variables.size(); i++) {
   for (int j = 0; j < variables2.size(); j++) {
      if (variables.get(i).equals(variables2.get(j))) { //-------> F.3.1
         idx1.add(i);
         idx2.add(j);
      }
   }
}
```

F.3.1 For each variables in two factors, check if two factors share same variable. Every time common variable is represented, add the index numbers where the variable is stored to idx1 and idx2, respectively.

2. Compare each row in two factors to product

```java
for (ProbRow p1 : this.getTable()) {
   for (ProbRow p2 : table2) {
      if (canProduct(p1, p2, idx1, idx2)) { //-------> F.3.2.1
         ArrayList<String> newValues = new ArrayList<String>();
         newValues = makeNewValues(p1, p2, idx2); //-------> F.3.2.2
         Double prob = p1.getProb() * p2.getProb(); //-------> F.3.2.3
         newProb.add(new ProbRow(newValues, prob));
      }
   }
}
```

F.3.2.1 Check if two rows(p1, p2) can be product by using canProduct function. The function returns true only when every common variables in two factors have same values. When comparing the values of common variables in both factors, the index list is used. The index list stores the index numbers for each variable in the same order.

```java
public boolean canProduct(ProbRow p1, ProbRow p2, ArrayList<Integer> idx1,
     ArrayList<Integer> idx2) {
    for (int i = 0; i < idx1.size(); i++) {
       int commonVal1 = idx1.get(i);
        int commonVal2 = idx2.get(i);
        String valOfF1 = p1.getValues().get(commonVal1);
        String valOfF2 = p2.getValues().get(commonVal2);
       if (!(valOfF1.equals(valOfF2)))
          return false;
    }
 return true;
 }
```

F.3.2.2 If it is possible to multiply two rows, the new list of values would be needed. At this time, this list stores the values of all the variables included in the two factors, and is stored only once for the duplicated variables. In other words, if the index number of the value list is an element in idx2, it means that it is the value of the common variable. Therefore, in this case this value would not be stored in the new value list. We can see this at F.3.2.2-1 in following code.

```
public ArrayList<String> makeNewValues(ProbRow p1, ProbRow p2,
    ArrayList<Integer> idx2) {
   ArrayList<String> values = new ArrayList<String>();
   values.addAll(p1.getValues());

   for (int i = 0; i < p2.getValues().size(); i++)
      if (!(idx2.contains(i))) //-------> F.3.2.2-1
         values.add(p2.getValues().get(i));

   return values;
 }
```

F.3.2.3 Now there is no doubt that it is possible to multiply two rows given. The probability of a new row is defined as the product of the probability of two rows.

3.1.4 Factor Reduction

1. Find observed variable

```
ArrayList<Variable> newVariables = variables;
int observedVar = -1;
for (int i = 0; i < newVariables.size(); i++)
   if (newVariables.get(i).getName().equals(ob.getName())) //-------> F.4.1
      observedVar = i;
```

F.4.1 To find the variable observed, check if the name of variable is same with observed one for each list of variable. If so, store the index number of list to observedVar.

2. Make new probability row with observed value

```
ArrayList<ProbRow> newRows = new ArrayList<ProbRow>();
for (ProbRow r : table)
   if (r.getValues().get(observedVar).equals(ob.getObservedValue()))
      newRows.add(r); //-------> F.4.2
```

F.4.2 To make a new row, check the observedVar-th element in the list of values in each row first. If it is equal to the value of variable observed, add the row to newRows. Then, a new factor reduced can be made.

3.2 Variable Elimination Algorithm Implementation

3.2.1 VE-Algorithm Class and Constructor

To be able to make use of frequently used ArrayLists and variables in the Variable Elimination algorithm and to get a more clean code, a separate class was created for the VE-algorithm. The class VEAlgorithm and the actual call of computation of the algorithm is initialized in the Main class of provided code, as below stated code.

```
//PUT YOUR CALL TO THE VARIABLE ELIMINATION ALGORITHM HERE
    VEAlgorithm ve = new VEAlgorithm (vs, ps, query, observed);
    ve.Algorithm();
```

In the constructor of the VE-Algorithm class, the ArrayList vs and ps are initialized with the lists of variables and probabilities nodes created from using the Main and Networkreader class of the provided code while query and ob Variable query, ArrayList obs are initialized using the given input of the user and the provided Main and UserInterface classes. In the further implementation of the VE-algorithm and Factor, the classes Variable, Table and ProbRow and its methods are used from the provided code attached to the assignment, these classes have not been further adjusted or expanded.

```
 ArrayList<Variable> vs;
ArrayList<Table> ps;
Variable query;
ArrayList<Variable> obs;
PriorityQueue<Variable> order;
ArrayList<Factor> factorList ;

public VEAlgorithm(ArrayList<Variable> variables, ArrayList<Table> probabilities,
     Variable query, ArrayList<Variable> observed) {
    this.vs = variables;
   this.ps = probabilities;
   this.query = query;
   this.obs = observed;
   }
```

3.2.2 Deciding order of elimination

The order in which the variables will be eliminated in the VE-algorithm is initialized with the function decideOrdering(), E.1.

```
 this.order = this.decideOrdering(); //-------> E.1
```

```
private PriorityQueue<Variable> decideOrdering() { //-------> E.1
     Comparator<Variable> comparator = new VariableComparator(); //-------> E.2
    PriorityQueue<Variable> order = new PriorityQueue<Variable>(vs.size(), comparator);
        //-------> E.2
    ArrayList<Variable> parentsOfQuery = ParentsOfQuery(); //-------> E.3
    ArrayList<Variable> parentsOfObs = ParentsOfObservedValue();//-------> E.4
    for (Variable v : vs) {
       if (orderMethodObvFirst()) { //-------> E.5.1
          if (!(v.equals(query)) && parentsOfQuery.contains(v)) //-------> E.5.2
            order.add(v);
       } else { //-------> E.5.3
          for (Variable ob : obs) {
             if (v.equals(ob))
```

```
                order.add(v);
             if (!(v.equals(query)) && parentsOfObs.contains(v))
                order.add(v);
            }
         }
      }
   return order;
   }
```

E.2 By using a PriorityQueue to store the variables in, the algorithm ensures that the variables have an actual order in which they will be eliminated. This order is determined by the VariableComparator class, which sorts the variables contained in the PriorityQueue order from Variable with smallest number of parents to variable with the highest number.

```
public class VariableComparator implements Comparator <Variable>{
   @Override
   public int compare(Variable o1, Variable o2) {
       if (o1.getNrOfParents() < o2.getNrOfParents())
         return -1 ;
      else if (o1.getNrOfParents() > o2.getNrOfParents())
         return 1 ;
      else
         return 0;
   }
}
```

E.3 The function ParentsOfQuery returns a list of every parent variable and their parents respectively of the given query.

```
private ArrayList<Variable> ParentsOfQuery() {
      ArrayList<Variable> parents = new ArrayList<Variable>();
      ArrayList<Variable> temp = new ArrayList<Variable>();
      temp.add(query); //-------> E.4.1
      while (!(temp.isEmpty())) {
         if (temp.get(0).hasParents()) {
            temp.addAll(temp.get(0).getParents());
            parents.addAll(temp.get(0).getParents());
         }
         temp.remove(0);
      }
      return parents;
   }
```

E.4 The function ParentsOfObservedValue returns a list of every parent variable and their parents respectively of the given observations variables. It is the same as the ParentsOfQuery function however at place E.4.1 in ParentsOfQueryinstead of temp.add(query); it is temp.addAll(obs);. See Appendix for ParentsfObservedValue function.

E.5.1 The function orderMethodObvFirst checks whether the observed value(s) is reached earlier than query or whether the query is earlier is reached earlier than the observed value(s). This has an influence on which variables will be added to the elimination order and on how the probability of query will be calculated. The boolean forward returns True when the observed value(s) are reached earlier than the query variable.

```java
 private boolean orderMethodObvFirst() {
    boolean forward = false;
    ArrayList<Variable> parents = new ArrayList<Variable>();
    ArrayList<Variable> temp = new ArrayList<Variable>();
    temp.add(query);
    while (!(temp.isEmpty())) {
      if (temp.get(0).hasParents()) {
        temp.addAll(temp.get(0).getParents());
        parents.addAll(temp.get(0).getParents());
      }
      temp.remove(0);
    }
    if (parents.containsAll(obs))
      forward = true;
    return forward;
  }
```

E.5.2 When orderMethodObvFirst() returns True, every variable that is needed to reach the query is added to the elimination order.

E.5.3 When orderMethodObvFirst() returns False, every variable in between observed value(s) and the query variable including the observed value(s) is added to the elimination order

3.2.3 Turning Variables into Factors

In order to implement the VE algorithm, the variables used in the Algorithm needs to be turned into their Factor counterparts and stored in the factorList. This is done by the function variablesIntoFactors, VF.1.

```java
 this.factorList = this.variablesIntoFactors(); //-------> VF.1
```

```java
private ArrayList<Factor> variablesIntoFactor() { //-------> VF.1
ArrayList<Factor> factorList = new ArrayList<Factor>();
for (Variable v : this.vs) {
  ArrayList<Variable> var = new ArrayList<Variable>();
  if (this.order.contains(v) || v.equals(query)) { //-------> VF.2
    var.add(v);
    if (v.getNrOfParents() >= 1) //-------> VF.3
      var.addAll(v.getParents());

    for (Table t : this.ps) {//-------> VF.4
      if (t.getVariable().equals(v))
        factorList.add(new Factor(var, t.getTable())); //-------> VF.5
    }
  }
}
```

```
        return factorList;
    }
```

VF.2 Every variable that is in the elimination order or which is the query is turned into a Factor and stored in the factorList.

VF.3 If a variable that is dependant on other Variables, then these variables will be added to the variableList in the Factor. This is to ensure that the variables of conditional probabilities are included in factors.

VF.4 Find the probability distribution corresponding to the variable.

VF.5 Create the new factor of the variable that needs to be used in the VE algorithm and store it in the factorList.

3.2.4 VE-Algorithm

The function Algorithm computes the probability of the query using the VE algorithm.

```
  public void Algorithm() {
     this.order = this.decideOrdering(); //-------> E.1
     this.factorList = this.variablesIntoFactors(); //-------> VF.1

     for (int k = 0; k < factorList.size(); k++) {
        for (int j = 0; j < obs.size(); j++) {
           for (int i = 0; i < factorList.get(k).getVariables().size(); i++) { //------->
               VE.1
              if (factorList.get(k).getVariables().get(i).equals(obs.get(j))) {
                 Factor newF = factorList.get(k).reduce(obs.get(j));
                 factorList.remove(factorList.get(k));
                 factorList.add(newF);
              }
           }
        }
     }

     while (!order.isEmpty()) { //-------> VE.2
        ArrayList<Factor> factorListProduct = new ArrayList<Factor>();
        ArrayList<Factor> toRemove = new ArrayList<Factor>();
        Variable v = order.poll();

        for (int i = 0; i < factorList.size(); i++) { //-------> VE.3
           Factor tempFac = factorList.get(i);
           if (tempFac.getVariables().contains(v)) {
              factorListProduct.add(tempFac);
              toRemove.add(tempFac);
           }
        }

        while (factorListProduct.size() != 1) { //-------> VE.4
           Factor newF = factorListProduct.get(0).product(factorListProduct.get(1));
           factorListProduct.remove(0);
```

```
            factorListProduct.remove(0);
            factorListProduct.add(0, newF);
         }

          //-------> VE.5
         Factor newF = factorListProduct.get(0).marginalization(v);
         factorList.add(newF);

         for (Factor r : toRemove) //-------> VE.6
            factorList.remove(r);
      }

      if (!(orderMethodObvFirst())) { //-------> VE.7
         while (factorList.size() != 1) {
            Factor newF = factorList.get(0).product(factorList.get(1));
            factorList.remove(0);
            factorList.remove(0);
            factorList.add(0, newF);
         }
      }

     //-------> VE.8
      Table resultTable = normalise();
      System.out.println(resultTable.toString());
   }
```

VE.1 For every factor in factorList, see if the variables of that factor includes one of the observed variables. If this is the case, use Factor Reduction to change the factor so that it only includes the observed values of the observed variable. Remove the old factor from the factorList and add the changed factor with the observed values to the factorList.

VE.2 For each variable in the elimination order :

- Create the factorListProduct ArrayList which stores every factor that needs to be multiplied and the toRemove ArrayList which stores every factor that needs to be removed from the factorList later on.

VE.3 Code to find all factors containing the variable that needs to be eliminated and stores those factors to factorListProduct and to toRemove.

VE.4 Code that computes the product of every factor in factorListProduct. It computes the product of the first two factors in the factorListProduct using the Factor Product function and removes these two factors from the factorListProduct while adding the new computed product to the factorListProduct. This repeats till only one factor remains thus when every factor containing the variable to eliminate has been multiplied.

VE.5 Sum the variable to eliminate out of the factor - the factor that has multiplied in VE.4 - using the Factor Marginalization function and add it to the factorList.

VE.6 Remove every factor that contained the eliminated variable in factorList.

VE.7 If the query is earlier in network than observed values, compute the product of the remaining factors in the factorList till only factor remains. This final factor is the probability of the query given the observed values after normalization.

VE.8 Turn the factor back into a variable and give the probability of that variable as output to the query. The function normalise is used to turn the factor back into a variable and compute the table of that variable. The function turns the table of the factor into a probability distribution table of the output variable of the query, by dividing the probability of every row in the table of the factor with the total sum of probabilities of each row in the table. The normalized probability value of each row is saved in the table that is given as output along with the query variable.

```
    private Table normalise() { //-------> VE.8
    ArrayList<ProbRow> resultProbRow = new ArrayList<ProbRow>();
    double sum = 0;
    for (int i = 0; i < factorList.get(0).getTable().size(); i++) {
       sum += factorList.get(0).getTable().get(i).getProb();
    }
    for (int i = 0; i < factorList.get(0).getTable().size(); i++) {
       double prob = factorList.get(0).getTable().get(i).getProb() / sum;
       resultProbRow.add(new
             ProbRow(factorList.get(0).getTable().get(i).getValues(), prob));
    }

    return new Table(query, resultProbRow);
 }
```

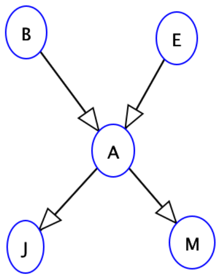
4 Testing

4.1 Simple Discrete Bayesian Network - Earthquake

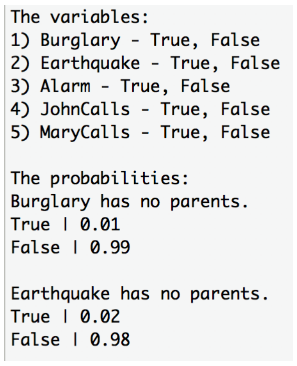
To test whether the implementation of the Variable Elimination algorithm returns the correct values, the discrete Bayesian Network Earthquake and the program AISpace was used. See Figure 1 for the representation and probabilities of variables of the Earthquake network. To test with earthquake file, the input stated below was entered in Main class :

```
public class Main {
   private final static String networkName = "earthquake.bif";
   public static void main(String[] args) {
      // Read in the network
      Networkreader reader = new Networkreader(networkName);
```

The results of testing whether the values are the correct when the query is earlier in network than observed value(s0, can be found in Figure 2 and Figure 3. The results of testing whether the values are the correct when the observed value(s) is earlier in network than query, can be found in Figure 4 and 5. The results of when wrong input was given can be found in Figure 6 and 7.



(a) representation

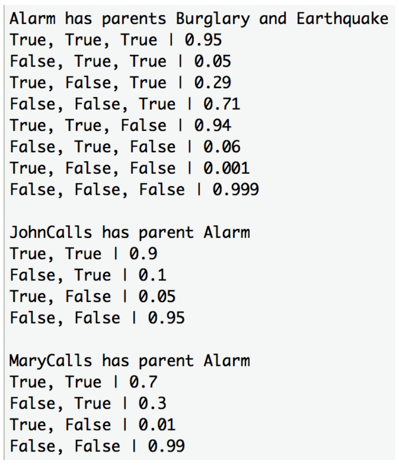


```
The variables:
1) Burglary - True, False
2) Earthquake - True, False
3) Alarm - True, False
4) JohnCalls - True, False
5) MaryCalls - True, False

The probabilities:
Burglary has no parents.
True | 0.01
False | 0.99

Earthquake has no parents.
True | 0.02
False | 0.98
```

(b) probabilities



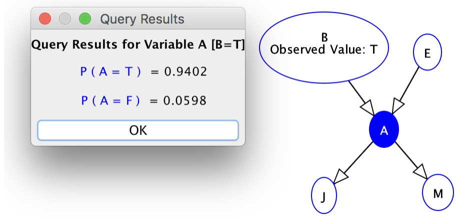
```
Alarm has parents Burglary and Earthquake
True, True, True | 0.95
False, True, True | 0.05
True, False, True | 0.29
False, False, True | 0.71
True, True, False | 0.94
False, True, False | 0.06
True, False, False | 0.001
False, False, False | 0.999

JohnCalls has parent Alarm
True, True | 0.9
False, True | 0.1
True, False | 0.05
False, False | 0.95

MaryCalls has parent Alarm
True, True | 0.7
False, True | 0.3
True, False | 0.01
False, False | 0.99
```

(c) probabilities

Figure 1: Bayesian network Earthquake representation and probabilities of variables

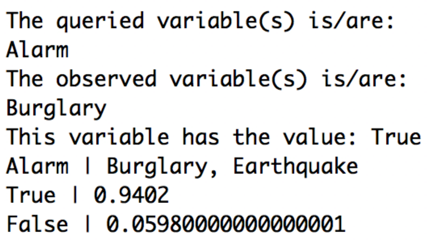


Query Results for Variable A [B=T]

P (A = T) = 0.9402

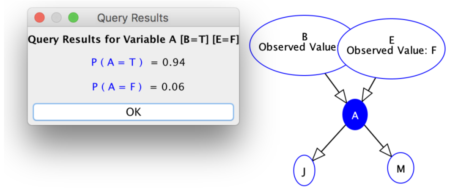
P (A = F) = 0.0598

(a) AI space



```
The queried variable(s) is/are:
Alarm
The observed variable(s) is/are:
Burglary
This variable has the value: True
Alarm | Burglary, Earthquake
True | 0.9402
False | 0.05980000000000001
```

(b) Output algorithm

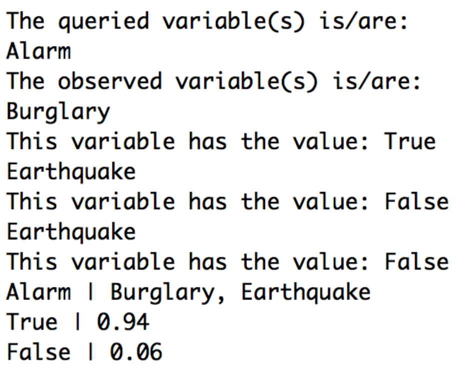


Query Results for Variable A [B=T] [E=F]

P (A = T) = 0.94

P (A = F) = 0.06

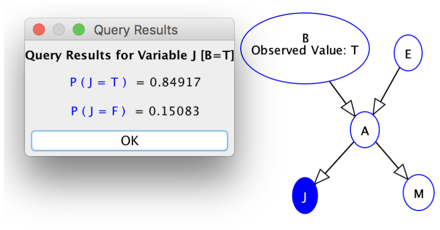
(c) AI space



```
The queried variable(s) is/are:
Alarm
The observed variable(s) is/are:
Burglary
This variable has the value: True
Earthquake
This variable has the value: False
Earthquake
This variable has the value: False
Alarm | Burglary, Earthquake
True | 0.94
False | 0.06
```

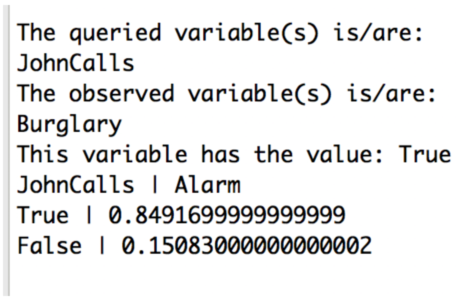
(d) output algorithm

Figure 2: Results of various inputs in Earthquake network where query is earlier in network than observed value(s)



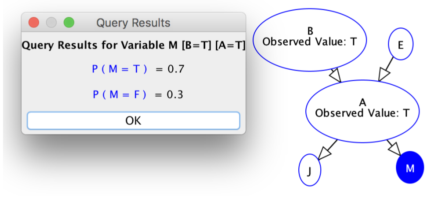
```
P ( J = T )  = 0.84917
P ( J = F )  = 0.15083
```

(a) AI space



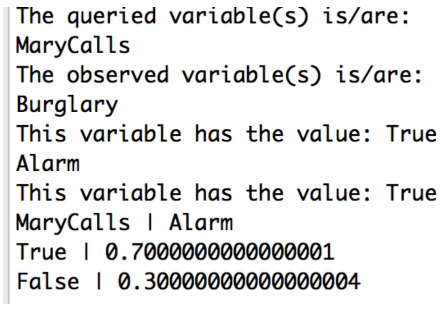
```
The queried variable(s) is/are:
JohnCalls
The observed variable(s) is/are:
Burglary
This variable has the value: True
JohnCalls | Alarm
True | 0.8491699999999999
False | 0.15083000000000002
```

(b) Output algorithm



```
P ( M = T )  = 0.7
P ( M = F )  = 0.3
```

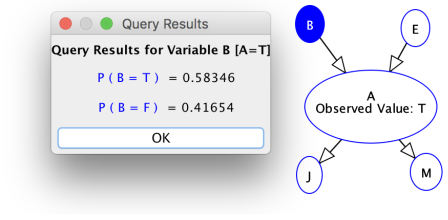
(c) AI space



```
The queried variable(s) is/are:
MaryCalls
The observed variable(s) is/are:
Burglary
This variable has the value: True
Alarm
This variable has the value: True
MaryCalls | Alarm
True | 0.7000000000000001
False | 0.30000000000000004
```

(d) output algorithm

Figure 3: Results of various inputs in Earthquake network where query is earlier in network than observed value(s)

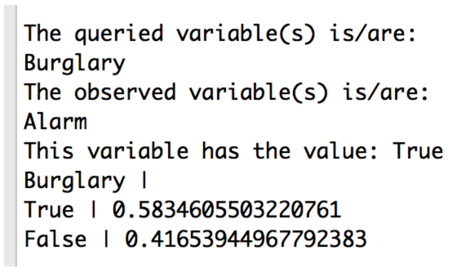

Query Results

Query Results for Variable B [A=T]

P (B = T) = 0.58346

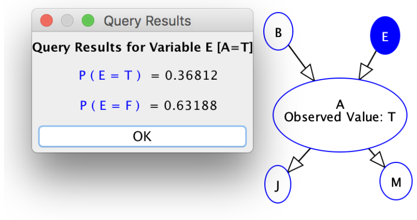
P (B = F) = 0.41654

(a) AI space



```
The queried variable(s) is/are:
Burglary
The observed variable(s) is/are:
Alarm
This variable has the value: True
Burglary |
True | 0.5834605503220761
False | 0.41653944967792383
```

(b) Output algorithm

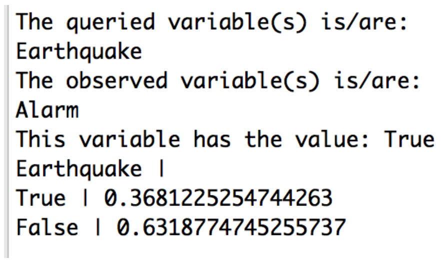

Query Results

Query Results for Variable E [A=T]

P (E = T) = 0.36812

P (E = F) = 0.63188

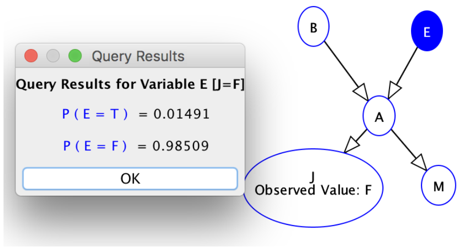
(c) AI space



```
The queried variable(s) is/are:
Earthquake
The observed variable(s) is/are:
Alarm
This variable has the value: True
Earthquake |
True | 0.3681225254744263
False | 0.6318774745255737
```

(d) output algorithm

Figure 4: Results of various inputs in Earthquake network where observed value(s) is earlier in network than query



Query Results

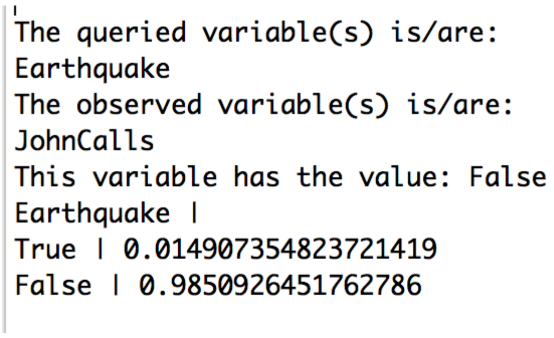
Query Results for Variable E [J=F]

P (E = T) = 0.01491

P (E = F) = 0.98509

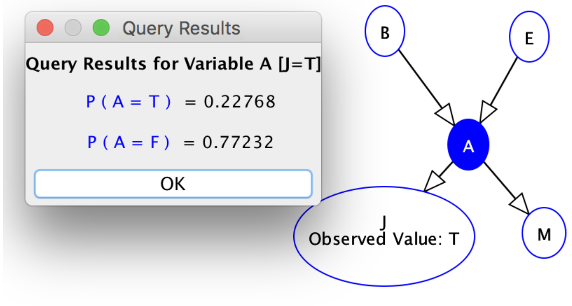
OK

(a) AI space



```
The queried variable(s) is/are:
Earthquake
The observed variable(s) is/are:
JohnCalls
This variable has the value: False
Earthquake |
True | 0.014907354823721419
False | 0.9850926451762786
```

(b) Output algorithm



Query Results

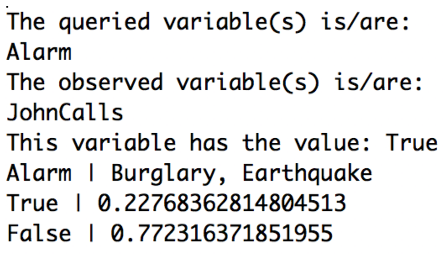
Query Results for Variable A [J=T]

P (A = T) = 0.22768

P (A = F) = 0.77232

OK

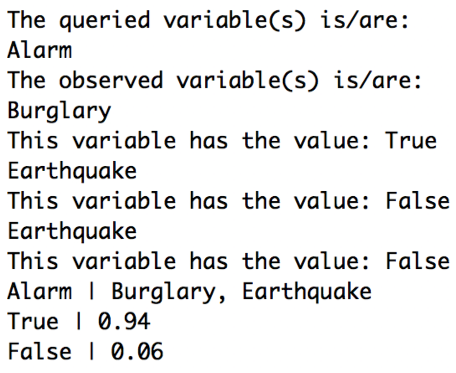
(c) AI space



```
The queried variable(s) is/are:
Alarm
The observed variable(s) is/are:
JohnCalls
This variable has the value: True
Alarm | Burglary, Earthquake
True | 0.22768362814804513
False | 0.772316371851955
```

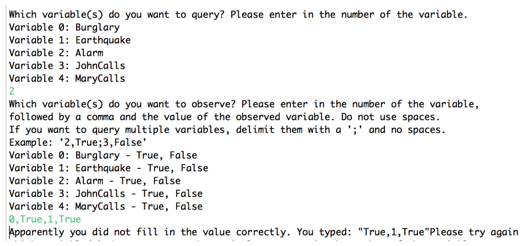
(d) output algorithm

Figure 5: Results of various inputs in Earthquake network where observed value(s) is earlier in network than query)



```
The queried variable(s) is/are:
Alarm
The observed variable(s) is/are:
Burglary
This variable has the value: True
Earthquake
This variable has the value: False
Earthquake
This variable has the value: False
Alarm | Burglary, Earthquake
True | 0.94
False | 0.06
```

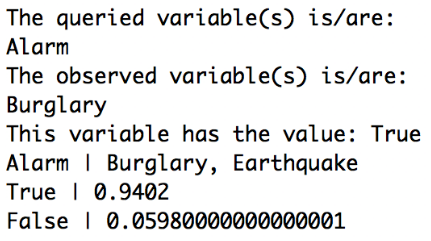
(a) Expected Output for 6.b



```
Which variable(s) do you want to query? Please enter in the number of the variable.
Variable 0: Burglary
Variable 1: Earthquake
Variable 2: Alarm
Variable 3: JohnCalls
Variable 4: MaryCalls
2
Which variable(s) do you want to observe? Please enter in the number of the variable,
followed by a comma and the value of the observed variable. Do not use spaces.
If you want to query multiple variables, delimit them with a ';' and no spaces.
Example: '2,True;3,False'
Variable 0: Burglary - True, False
Variable 1: Earthquake - True, False
Variable 2: Alarm - True, False
Variable 3: JohnCalls - True, False
Variable 4: MaryCalls - True, False
0,True,1,True
Apparently you did not fill in the value correctly. You typed: "True,1,True"Please try again
```

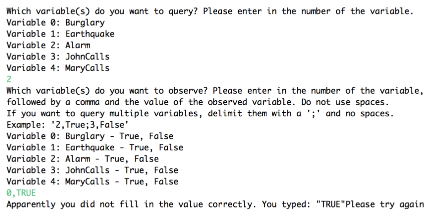
(b) Actual output

Figure 6: Results of various wrong inputs in Earthquake network



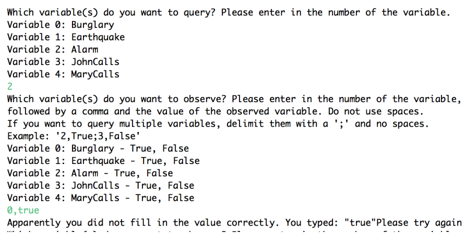
```
The queried variable(s) is/are:
Alarm
The observed variable(s) is/are:
Burglary
This variable has the value: True
Alarm | Burglary, Earthquake
True | 0.9402
False | 0.05980000000000001
```

(a) Expected Output for 7.b and 7.c, 7.d



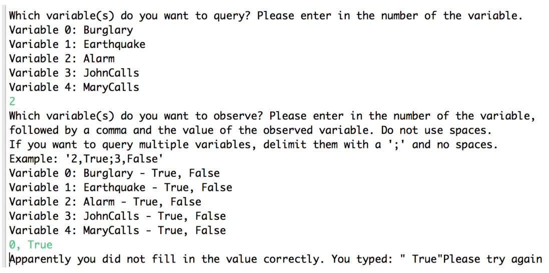
```
Which variable(s) do you want to query? Please enter in the number of the variable.
Variable 0: Burglary
Variable 1: Earthquake
Variable 2: Alarm
Variable 3: JohnCalls
Variable 4: MaryCalls
2
Which variable(s) do you want to observe? Please enter in the number of the variable,
followed by a comma and the value of the observed variable. Do not use spaces.
If you want to query multiple variables, delimit them with a ';' and no spaces.
Example: '2,True;3,False'
Variable 0: Burglary - True, False
Variable 1: Earthquake - True, False
Variable 2: Alarm - True, False
Variable 3: JohnCalls - True, False
Variable 4: MaryCalls - True, False
0,TRUE
Apparently you did not fill in the value correctly. You typed: "TRUE"Please try again
```

(b) Actual Output



```
Which variable(s) do you want to query? Please enter in the number of the variable.
Variable 0: Burglary
Variable 1: Earthquake
Variable 2: Alarm
Variable 3: JohnCalls
Variable 4: MaryCalls
2
Which variable(s) do you want to observe? Please enter in the number of the variable,
followed by a comma and the value of the observed variable. Do not use spaces.
If you want to query multiple variables, delimit them with a ';' and no spaces.
Example: '2,True;3,False'
Variable 0: Burglary - True, False
Variable 1: Earthquake - True, False
Variable 2: Alarm - True, False
Variable 3: JohnCalls - True, False
Variable 4: MaryCalls - True, False
0,true
Apparently you did not fill in the value correctly. You typed: "true"Please try again
```

(c) Actual Output



```
Which variable(s) do you want to query? Please enter in the number of the variable.
Variable 0: Burglary
Variable 1: Earthquake
Variable 2: Alarm
Variable 3: JohnCalls
Variable 4: MaryCalls
2
Which variable(s) do you want to observe? Please enter in the number of the variable,
followed by a comma and the value of the observed variable. Do not use spaces.
If you want to query multiple variables, delimit them with a ';' and no spaces.
Example: '2,True;3,False'
Variable 0: Burglary - True, False
Variable 1: Earthquake - True, False
Variable 2: Alarm - True, False
Variable 3: JohnCalls - True, False
Variable 4: MaryCalls - True, False
0, True
Apparently you did not fill in the value correctly. You typed: " True"Please try again
```

(d) Actual output

Figure 7: Results of various wrong inputs for Earthquake network

4.2 Medium Discrete Bayesian Network - Alarm

To test whether the implementation of the Variable Elimination algorithm on medium network, the discrete Bayesian Network Alarm was used. See Figure 8 for the representation Alarm network and the source http://www.bnlearn.com/bnrepository/discrete-medium.htmlalarm to get the probabilities of the variables of the network. The list of probabilities distributions is too large to be included in this report. To test with alarm file, the input stated below was entered in Main class :

```java
public class Main {
   private final static String networkName = "alarm.bif";
   public static void main(String[] args) {
      // Read in the network
      Networkreader reader = new Networkreader(networkName);
```

It was not possible to test whether the values were accurate for the Bayesian Network Alarm, as the AISpace program gave a lot of errors when trying to draw or import network file of Alarm. See Figure 9 and 10 for various outputs in Alarm network. See Figure 11 for various wrong inputs in

Alarm network.

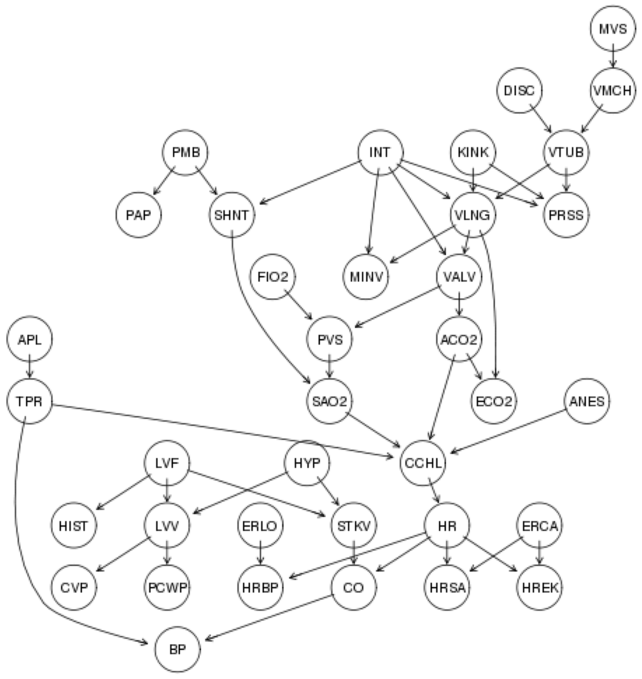
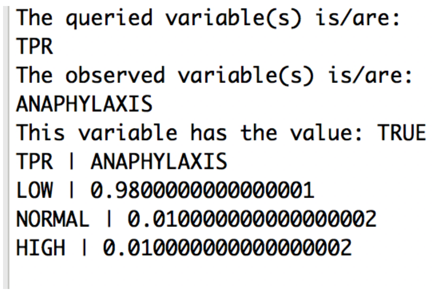
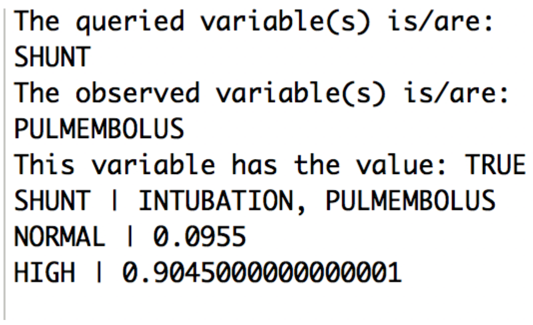


Figure 8: Representation Alarm Bayesian network



```
The queried variable(s) is/are:
TPR
The observed variable(s) is/are:
ANAPHYLAXIS
This variable has the value: TRUE
TPR | ANAPHYLAXIS
LOW | 0.9800000000000001
NORMAL | 0.010000000000000002
HIGH | 0.010000000000000002
```

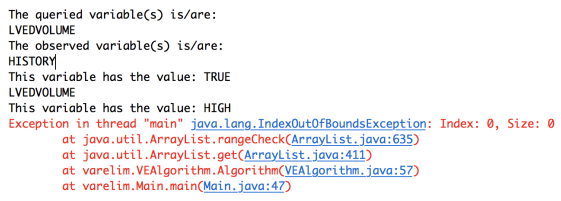
(a) Output for 1 observed value



```
The queried variable(s) is/are:
SHUNT
The observed variable(s) is/are:
PULMEMBOLUS
This variable has the value: TRUE
SHUNT | INTUBATION, PULMEMBOLUS
NORMAL | 0.0955
HIGH | 0.9045000000000001
```

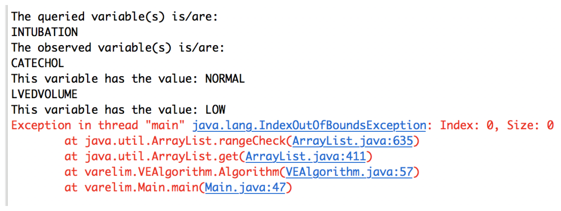
(b) Output for 1 observed value

Figure 9: Results of various inputs in Alarm network



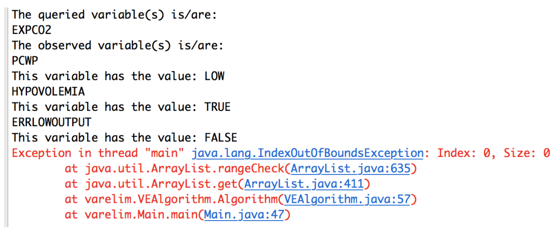
```
The queried variable(s) is/are:
LVEDVOLUME
The observed variable(s) is/are:
HISTORY
This variable has the value: TRUE
LVEDVOLUME
This variable has the value: HIGH
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
	at java.util.ArrayList.rangeCheck(ArrayList.java:635)
	at java.util.ArrayList.get(ArrayList.java:411)
	at varelim.VEAlgorithm.Algorithm(VEAlgorithm.java:57)
	at varelim.Main.main(Main.java:47)
```

(a) Output for 2 observed values



```
The queried variable(s) is/are:
INTUBATION
The observed variable(s) is/are:
CATECHOL
This variable has the value: NORMAL
LVEDVOLUME
This variable has the value: LOW
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
	at java.util.ArrayList.rangeCheck(ArrayList.java:635)
	at java.util.ArrayList.get(ArrayList.java:411)
	at varelim.VEAlgorithm.Algorithm(VEAlgorithm.java:57)
	at varelim.Main.main(Main.java:47)
```

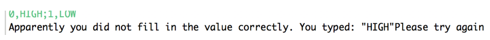
(b) Output for 2 observed values



```
The queried variable(s) is/are:
EXPCO2
The observed variable(s) is/are:
PCWP
This variable has the value: LOW
HYPOVOLEMIA
This variable has the value: TRUE
ERRLOWOUTPUT
This variable has the value: FALSE
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
	at java.util.ArrayList.rangeCheck(ArrayList.java:635)
	at java.util.ArrayList.get(ArrayList.java:411)
	at varelim.VEAlgorithm.Algorithm(VEAlgorithm.java:57)
	at varelim.Main.main(Main.java:47)
```

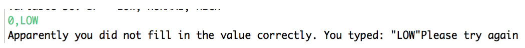
(c) Output for 3 observed values

Figure 10: Results of various inputs in Alarm network



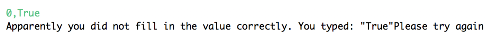
```
0,HIGH;1,LOW
Apparently you did not fill in the value correctly. You typed: "HIGH"Please try again
```

(a) Wrong Output



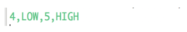
```
0,LOW
Apparently you did not fill in the value correctly. You typed: "LOW"Please try again
```

(b) Wrong Output



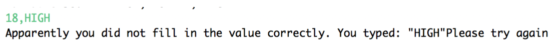
```
0,True
Apparently you did not fill in the value correctly. You typed: "True"Please try again
```

(c) Wrong Output



```
4,LOW,5,HIGH
```

(d) Wrong Output



```
18,HIGH
Apparently you did not fill in the value correctly. You typed: "HIGH"Please try again
```

(e) Wrong output

Figure 11: Results of various wrong inputs for Alarm network

4.3 Big Discrete Bayesian Network - Alarm

To test whether the implementation of the Variable Elimination algorithm on big network, the discrete Bayesian Network HEPAR2 was used. See Figure 12 for the representation HEPAR2 network and the source http://www.bnlearn.com/bnrepository/discrete-large.html to get the probabilities of the variables of the network. The list of probabilities distributions is too large to be included in this report. To test with HEPAR2 file, the input stated below was entered in Main class :

```
public class Main {
   private final static String networkName = "hepar2.bif"l
   public static void main(String[] args) {
      // Read in the network
      Networkreader reader = new Networkreader(networkName);
```

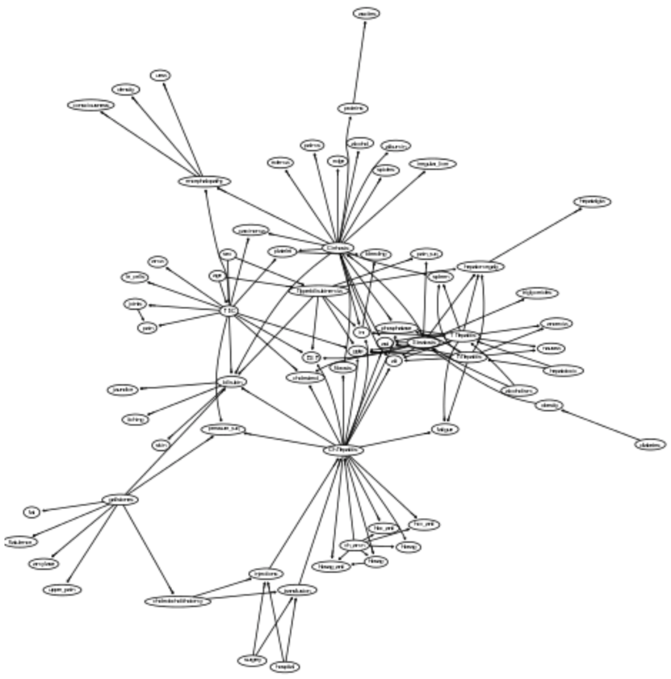
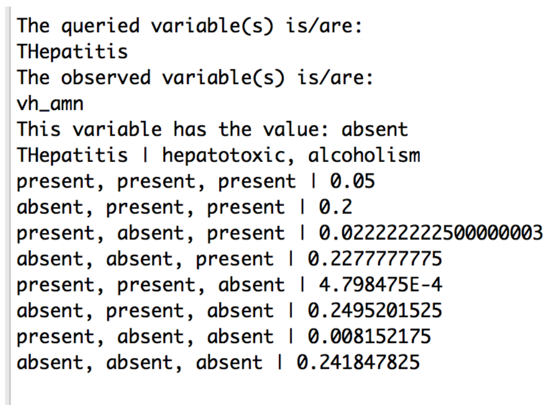


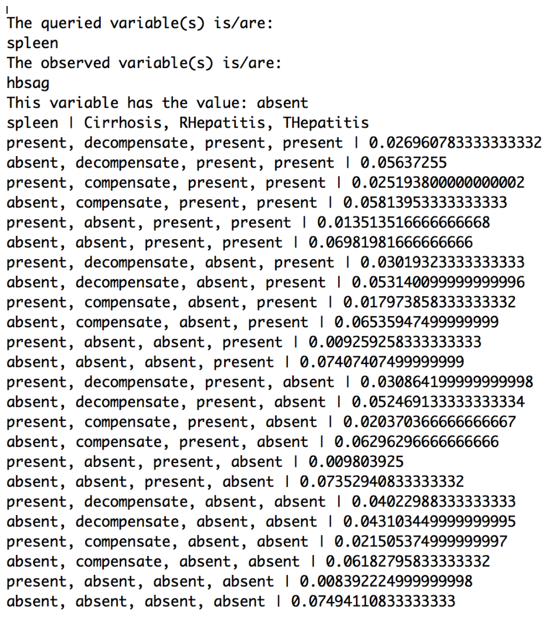
Figure 12: Representation HEPAR2 Bayesian network

It was not possible to test whether the values were accurate for the Bayesian Network HEPAR2, as the AISpace program once again gave a lot of errors when trying to draw or import network file of HEPAR2. See Figure 13 and 14 for various outputs of HEPAR2 network.



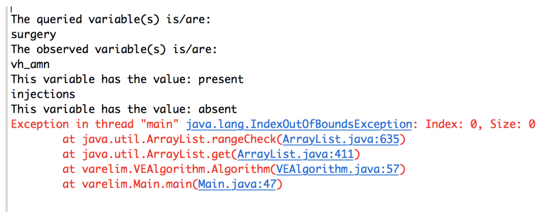
```
The queried variable(s) is/are:
THepatitis
The observed variable(s) is/are:
vh_amn
This variable has the value: absent
THepatitis | hepatotoxic, alcoholism
present, present, present | 0.05
absent, present, present | 0.2
present, absent, present | 0.022222222500000003
absent, absent, present | 0.2277777775
present, present, absent | 4.798475E-4
absent, present, absent | 0.2495201525
present, absent, absent | 0.008152175
absent, absent, absent | 0.241847825
```

(a) Output for 1 observed values



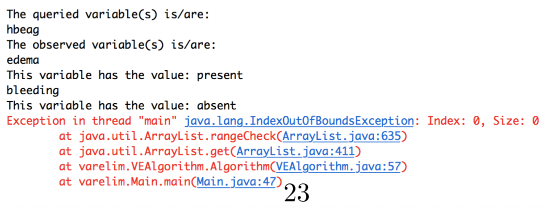
```
The queried variable(s) is/are:
spleen
The observed variable(s) is/are:
hbsag
This variable has the value: absent
spleen | Cirrhosis, RHepatitis, THepatitis
present, decompensate, present, present | 0.026960783333333332
absent, decompensate, present, present | 0.05637255
present, compensate, present, present | 0.025193800000000002
absent, compensate, present, present | 0.058139533333333333
present, absent, present, present | 0.013513516666666668
absent, absent, present, present | 0.06981981666666666
present, decompensate, absent, present | 0.030193233333333333
absent, decompensate, absent, present | 0.053140099999999996
present, compensate, absent, present | 0.017973858333333332
absent, compensate, absent, present | 0.06535947499999999
present, absent, absent, present | 0.009259258333333333
absent, absent, absent, present | 0.07407407499999999
present, decompensate, present, absent | 0.030864199999999998
absent, decompensate, present, absent | 0.052469133333333334
present, compensate, present, absent | 0.020370366666666667
absent, compensate, present, absent | 0.06296296666666666
present, absent, present, absent | 0.009803925
absent, absent, present, absent | 0.07352940833333332
present, decompensate, absent, absent | 0.040229883333333333
absent, decompensate, absent, absent | 0.043103449999999995
present, compensate, absent, absent | 0.021505374999999997
absent, compensate, absent, absent | 0.06182795833333332
present, absent, absent, absent | 0.008392224999999998
absent, absent, absent, absent | 0.07494110833333333
```

(b) Output for 1 observed values



```
The queried variable(s) is/are:
surgery
The observed variable(s) is/are:
vh_amn
This variable has the value: present
injections
This variable has the value: absent
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
	at java.util.ArrayList.rangeCheck(ArrayList.java:635)
	at java.util.ArrayList.get(ArrayList.java:411)
	at varelim.VEAlgorithm.Algorithm(VEAlgorithm.java:57)
	at varelim.Main.main(Main.java:47)
```

(c) Output for 2 observed values



```
The queried variable(s) is/are:
hbeag
The observed variable(s) is/are:
edema
This variable has the value: present
bleeding
This variable has the value: absent
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
	at java.util.ArrayList.rangeCheck(ArrayList.java:635)
	at java.util.ArrayList.get(ArrayList.java:411)
	at varelim.VEAlgorithm.Algorithm(VEAlgorithm.java:57)
	at varelim.Main.main(Main.java:47)
```

(d) Output for 2 observed values

Figure 13: Results of various inputs in HEPAR2 network

4.4 Runtime

4.4.1 Measurement

For the runtime measurement implementation, multiple methods were looked into. The methods were chosen to calculate the running time using an available system timer in Java because the priority queue is used in the code. There were 2 feasible options: 1) System.nanoTime() and 2) System.currentTimeMillis().
As a result, System.nanoTime() would measure the runtime the most accurately as it measures a time in nanoseconds relative to an arbitrary point.

```
VEAlgorithm ve = new VEAlgorithm (vs, ps, query, observed);
long startTime = System.nanoTime();
ve.Algorithm();
long endTime = System.nanoTime();

long runTime = endTime - startTime;
System.out.println("[RUNTIME] ---> " + runTime + " nanosec.");
```

4.4.2 Result

1. Small Network: earthquake.bif

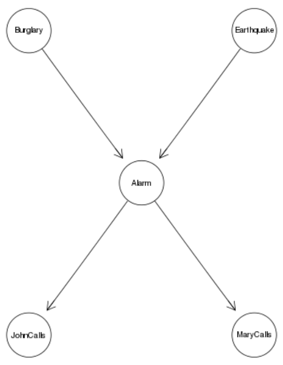


Figure 14: Alarm Network

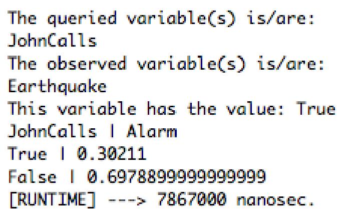
FORWARD

- input: 3(JOHNCALLS) & 1(EARTHQUAKE),TRUE
- output & run-time:

BACKWARD

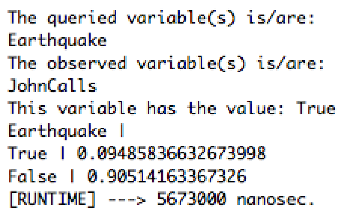
- input: 1(EARTHQUAKE) & 3(JOHNCALLS),TRUE
- output & run-time:

2. Midium Network: alarm.bif
FORWARD



```
The queried variable(s) is/are:
JohnCalls
The observed variable(s) is/are:
Earthquake
This variable has the value: True
JohnCalls | Alarm
True | 0.30211
False | 0.6978899999999999
[RUNTIME] ---> 7867000 nanosec.
```

Figure 15: Small Network - forward



```
The queried variable(s) is/are:
Earthquake
The observed variable(s) is/are:
JohnCalls
This variable has the value: True
Earthquake |
True | 0.09485836632673998
False | 0.90514163367326
[RUNTIME] ---> 5673000 nanosec.
```

Figure 16: Small Network - backward

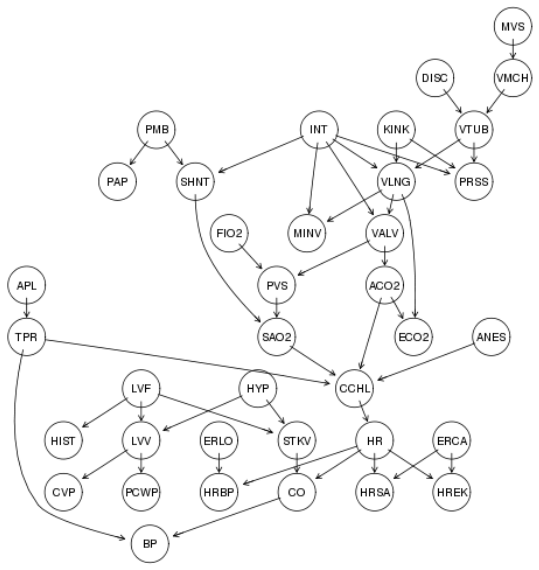


Figure 17: Alarm Network

- input: 20(SAO2) & 16(KINK),TRUE
- output & run-time:

BACKWARD

- input: 16(KINK) & 20(SAO2),LOW
- output & run-time:

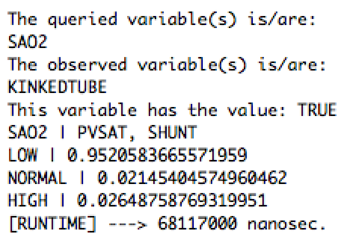

```
The queried variable(s) is/are:
SAO2
The observed variable(s) is/are:
KINKEDTUBE
This variable has the value: TRUE
SAO2 | PVSAT, SHUNT
LOW | 0.9520583665571959
NORMAL | 0.02145404574960462
HIGH | 0.02648758769319951
[RUNTIME] ---> 68117000 nanosec.
```

Figure 18: Midium Network - forward

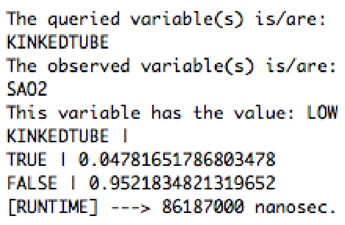

```
The queried variable(s) is/are:
KINKEDTUBE
The observed variable(s) is/are:
SAO2
This variable has the value: LOW
KINKEDTUBE |
TRUE | 0.04781651786803478
FALSE | 0.9521834821319652
[RUNTIME] ---> 86187000 nanosec.
```

Figure 19: Midium Network - backward

5 Conclusion

5.1 Simple Discrete Bayesian Network - Earthquake

From the results of Figure 2,3,4 and 5 can be concluded that the implementation of the Variable Elimination algorithm works as the output correspond to calculated probabilities values. From the results of Figure 6 and 7, it can be concluded that algorithm is very sensitive to mistakes of the user input. This is not favourable and would be a good point to improve algorithm on. One thing that could be implemented to improve this, is to make the user input is not sensitive to capital letters or spaces.

5.2 Medium Discrete Bayesian Network - Alarm

From the results of Figure 9,10 it can be concluded that the implementation of the Variable Elimination algorithm works for outputs where the input consist of only one observed value, but not for multiple observed values. The error is given in as an index out of bounds exception in below stated code :

```
while (factorListProduct.size() != 1) {
        Factor newF = factorListProduct.get(0).product(factorListProduct.get(1));
      }
```

The origin of error can be in the Product function of the Factor class. Since the error is not encountered when giving multiple observed values in the simpler Earthquake Bayesian network, it could be that Alarm network variables can have more than 2 discrete values and that the implementation of the product function is not equipped to handle more than 2 discrete values. For further research and improvement of algorithm, this should be looked into and solved.
From the results of Figure 11, it can be concluded that the algorithm continuous to be very sensitive

to user input mistakes. In addition, it can be noted that the user must know exactly what the input values can be for each variables in network. This notably harder to do, when the number of variables and the values of the variable of that network increase. A point to improve besides making the algorithm less sensitive to user input mistakes, is to make a more clear overview of which value the variable can take after giving the observed value as input.

5.3 Big Discrete Bayesian Network - Alarm

From Figure 13 it can be can concluded that the algorithm experiences the same error as the error in Medium Discrete Bayesian Network Alarm when the input is given as more than 2 observed values.

5.4 Runtime

From the results of running time observed, it can be concluded that the following about the relationship between the runtime and the size of network:

- The runtime is almost proportional to the size of network. For medium networks, it was observed that in all cases regardless of whether forward or backward, the running time is longer than that of the small network. That is, the larger the amount of variables and relationship between them are, the larger the runtime will be.
- In most networks on the same network, forward is faster than backward.

However, these conclusions are not always established in all networks. For example, in out previous small network test, earthquake, it was found that the runtime when running backward as an algorithm and the runtime when running as forward do not have a noticeable difference. In a simple network, the complexity is so low that it seems that the computer can be processed quickly in almost same time, no matter which direction the algorithm is run. Also, even if there are the same number of variables, the execution time may vary depending on the network connectivity and complexity.
For future improvement and testing of the Variable Elimination it would be interesting to use a different method to choose the variable elimination order and see how it compares in runtime with the current heuristic of the elimination order.

5.5 Degree Fulfillment requirement project

The implementation fulfills the requirement of having a working accurate algorithm for simple discrete Bayesian Networks. There is an error when the are multiple observed values given as input when using bigger discrete Bayesian networks. This error results from an implementation error in algorithm or product, which would be recommend to be improved and fixed. In the future, it is also recommendable to test multiple heuristics for choosing the order and have the algorithm working not only discrete Bayesian Networks.

6 Appendix

6.1 Sources

- Discrete Bayesian Network Earthquake - K. B. Korb, A. E. Nicholson. Bayesian Artificial Intelligence, 2nd edition, Section 2.5.1. CRC Press, 2010.
- AI Space program - http://www.aispace.org

- Discrete Bayesian Network Alarm - I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine, pages 247-256. Springer-Verlag, 1989.
- bnlearn - http://www.bnlearn.com/bnrepository/discrete-medium.htmlalarm Discrete Bayesian Netowork HEPARN2 - A. Onisko. Probabilistic Causal Models in Medicine: Application to Diagnosis of Liver Disorders. Ph.D. Dissertation, Institute of Biocybernetics and Biomedical Engineering, Polish Academy of Science, Warsaw, March 2003.
- bnlearn - http://www.bnlearn.com/bnrepository/discrete-large.html

6.2 Whole code provided Main class

```java
package varelim;
import java.util.ArrayList;

/**
 * Main class to read in a network, add queries and observed variables, and run variable
     elimination.
 *
 * @author Marcel de Korte, Moira Berens, Djamari Oetringer, Abdullahi Ali, Leonieke van
     den Bulk
 */

public class Main {
   private final static String networkName = "earthquake.bif"; // The network to be read in
       (format and other networks can be found on http://www.bnlearn.com/bnrepository/)

   public static void main(String[] args) {

      // Read in the network
      Networkreader reader = new Networkreader(networkName);

      // Get the variables and probabilities of the network
      ArrayList<Variable> vs = reader.getVs();
      ArrayList<Table> ps = reader.getPs();

      // Make user interface
      UserInterface ui = new UserInterface(vs, ps);

      // Print variables and probabilities
      ui.printNetwork();

      // Ask user for query and heuristic
      ui.askForQuery();
      Variable query = ui.getQueriedVariable();

      // Turn this on if you want to experiment with different heuristics for bonus points
          (you need to implement the heuristics yourself)
      //reader.askForHeuristic();
      //String heuristic = Ui.getHeuristic();

      // Ask user for observed variables
      ui.askForObservedVariables();
      ArrayList<Variable> observed = ui.getObservedVariables();

      // Print the query and observed variables
      ui.printQueryAndObserved(query, observed);

      //PUT YOUR CALL TO THE VARIABLE ELIMINATION ALGORITHM HERE
      VEAlgorithm ve = new VEAlgorithm (vs, ps, query, observed);
      long startTime = System.nanoTime();
      ve.Algorithm();
      long endTime = System.nanoTime();

      long runTime = endTime - startTime;
      System.out.println("[RUNTIME] ---> " + runTime + " nanosec.");
   }
}
```

6.3 Whole Code VEAlgorithm class

```
package varelim;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.PriorityQueue;

public class VEAlgorithm {
   ArrayList<Variable> vs;
   ArrayList<Table> ps;
   Variable query;
   ArrayList<Variable> obs;
   PriorityQueue<Variable> order;
   ArrayList<Factor> factorList ;

   public VEAlgorithm(ArrayList<Variable> variables, ArrayList<Table> probabilities,
       Variable query,
        ArrayList<Variable> observed) {
      this.vs = variables;
      this.ps = probabilities;
      this.query = query;
      this.obs = observed;
   }

   public void Algorithm() {
      this.order = this.decideOrdering();
      this.factorList = this.variablesIntoFactors();

      for (int k = 0; k < factorList.size(); k++) {
         for (int j = 0; j < obs.size(); j++) {
```

```
         for (int i = 0; i < factorList.get(k).getVariables().size(); i++) {
            if (factorList.get(k).getVariables().get(i).equals(obs.get(j))) {
               Factor newF = factorList.get(k).reduce(obs.get(j));
               factorList.remove(factorList.get(k));
               factorList.add(newF);
            }
         }
      }
   }

   while (!order.isEmpty()) {
      ArrayList<Factor> factorListProduct = new ArrayList<Factor>();
      ArrayList<Factor> toRemove = new ArrayList<Factor>();
      Variable v = order.poll();

      for (int i = 0; i < factorList.size(); i++) {
         Factor tempFac = factorList.get(i);
         if (tempFac.getVariables().contains(v)) {
            factorListProduct.add(tempFac);
            toRemove.add(tempFac);
         }
      }

      while (factorListProduct.size() != 1) {
         Factor newF = factorListProduct.get(0).product(factorListProduct.get(1));
         factorListProduct.remove(0);
         factorListProduct.remove(0);
         factorListProduct.add(0, newF);
      }

      Factor newF = factorListProduct.get(0).marginalization(v);
      factorList.add(newF);
      for (Factor r : toRemove)
         factorList.remove(r);
   }

   if (!(orderMethodObvFirst())) {
      while (factorList.size() != 1) {
         Factor newF = factorList.get(0).product(factorList.get(1));
         factorList.remove(0);
         factorList.remove(0);
         factorList.add(0, newF);
      }
   }

   Table resultTable = normalise();
   System.out.println(resultTable.toString());
}

private Table normalise() {
   ArrayList<ProbRow> resultProbRow = new ArrayList<ProbRow>();
   double sum = 0;
```

```
    for (int i = 0; i < factorList.get(0).getTable().size(); i++) {
      sum += factorList.get(0).getTable().get(i).getProb();
    }
    for (int i = 0; i < factorList.get(0).getTable().size(); i++) {
      double prob = factorList.get(0).getTable().get(i).getProb() / sum;
      resultProbRow.add(new ProbRow(factorList.get(0).getTable().get(i).getValues(),
          prob));
    }

    return new Table(query, resultProbRow);
  }

  private ArrayList<Factor> variablesIntoFactors() {
    ArrayList<Factor> factorList = new ArrayList<Factor>();
    for (Variable v : this.vs) {
      ArrayList<Variable> var = new ArrayList<Variable>();
      if (this.order.contains(v) || v.equals(query)) {
        var.add(v);
        if (v.getNrOfParents() >= 1)
          var.addAll(v.getParents());

        for (Table t : this.ps) {
          if (t.getVariable().equals(v))
            factorList.add(new Factor(var, t.getTable()));
        }
      }
    }
    return factorList;
  }

  private PriorityQueue<Variable> decideOrdering() {
    Comparator<Variable> comparator = new VariableComparator();
    PriorityQueue<Variable> order = new PriorityQueue<Variable>(vs.size(), comparator);
    ArrayList<Variable> parentsOfQuery = ParentsOfQuery();
    ArrayList<Variable> parentsOfObs = ParentsOfObservedValue();
    for (Variable v : vs) {
      if (orderMethodObvFirst()) {
        if (!(v.equals(query)) && parentsOfQuery.contains(v))
          order.add(v);
      } else {
        for (Variable ob : obs) {
          if (v.equals(ob))
            order.add(v);
          if (!(v.equals(query)) && parentsOfObs.contains(v))
            order.add(v);
        }
      }
    }
    return order;

  }
```

```java
   private ArrayList<Variable> ParentsOfQuery() {
      ArrayList<Variable> parents = new ArrayList<Variable>();
      ArrayList<Variable> temp = new ArrayList<Variable>();
      temp.add(query);
      while (!(temp.isEmpty())) {
         if (temp.get(0).hasParents()) {
            temp.addAll(temp.get(0).getParents());
            parents.addAll(temp.get(0).getParents());
         }
         temp.remove(0);
      }
      return parents;
   }

   private ArrayList<Variable> ParentsOfObservedValue() {
      ArrayList<Variable> parents = new ArrayList<Variable>();
      ArrayList<Variable> temp = new ArrayList<Variable>();
      temp.addAll(obs);
      while (!(temp.isEmpty())) {
         if (temp.get(0).hasParents()) {
            temp.addAll(temp.get(0).getParents());
            parents.addAll(temp.get(0).getParents());
         }
         temp.remove(0);
      }
      return parents;
   }

   private boolean orderMethodObvFirst() {
      boolean forward = false;
      ArrayList<Variable> parents = new ArrayList<Variable>();
      ArrayList<Variable> temp = new ArrayList<Variable>();
      temp.add(query);
      while (!(temp.isEmpty())) {
         if (temp.get(0).hasParents()) {
            temp.addAll(temp.get(0).getParents());
            parents.addAll(temp.get(0).getParents());
         }
         temp.remove(0);
      }
      if (parents.containsAll(obs))
         forward = true;
      return forward;
   }
}
```

6.4 Whole code VariableComparator class

```java
package varelim;
```

```java
import java.util.Comparator;

public class VariableComparator implements Comparator <Variable>{

   @Override
   public int compare(Variable o1, Variable o2) {
      if (o1.getNrOfParents() < o2.getNrOfParents())
         return -1 ;
      else if (o1.getNrOfParents() > o2.getNrOfParents())
         return 1 ;
      else
         return 0;
   }

}
```

6.5 Whole code Factor class

```java
package varelim;

import java.util.ArrayList;

public class Factor {
   private ArrayList<Variable> variables;
   private ArrayList<ProbRow> table;

   public Factor(ArrayList<Variable> variables, ArrayList<ProbRow> table) {
      this.variables = variables;
      this.table = table;
   }

   public Factor(Variable var, ArrayList<ProbRow> table) {
      ArrayList<Variable> variables = new ArrayList<Variable>();
      variables.add(var);
      if (var.getNrOfParents() != 0)
         variables.addAll(var.getParents());
      this.variables = variables;
      this.table = table;
   }

   public ProbRow get(int i) {
      return table.get(i);
   }

   public ArrayList<Variable> getVariables() {
      return variables;
   }

   public ArrayList<ProbRow> getTable() {
      return table;
```

```java
}

public boolean contains(Variable var) {
   for (Variable v : this.variables)
      if (v.equals(var))
         return true;

   return false;
}

// Factor Marginalization
public Factor marginalization(Variable v) {
   ArrayList<Variable> newVariables = new ArrayList<Variable>();
   ArrayList<ProbRow> newRows = new ArrayList<ProbRow>(); // list of rows without V
   int deletedVar = -1;

   // For the case1: when the factor has only one variable

   if (variables.size() == 1) {
      ArrayList<String> newValues = new ArrayList<String>();
      newRows.add(new ProbRow(newValues, sumAllProb()));

      return new Factor(variables, newRows);
   }

   // For the case2: there are more than one variable in the factor

   // 1. check if the factor contains the variable given
   for (int i = 0; i < variables.size(); i++) {
      // if so, put the index of variable we need to delete into deletedVar
      if (variables.get(i).equals(v))
         deletedVar = i;
   }

   // 2. compare each two rows in factor
   // make integer list to check if this index's values was visited
   ArrayList<Integer> checkedIdx = new ArrayList<Integer>();
   for (int i = 0; i < table.size(); i++) {
      if (!checkedIdx.contains(i)) {
         // if it is not visited yet, add the index number to list
         checkedIdx.add(i);
         Double prob = table.get(i).getProb();

         for (int j = i + 1; j < table.size(); j++) {
            // compare two rows' values are same
            if ( canSum(table.get(i), table.get(j), deletedVar) ) {
               // if so, add the index number of another row also to the list for
                   avoiding row duplicated
               // and calculate new probability by adding their probabilities.
               checkedIdx.add(j);
               prob += table.get(j).getProb();
            }
         }

         // make a new value list containing all values in factor without the value in
             deletedVar
         ArrayList<String> newValues = new ArrayList<String>();
         newValues = makeNewValues(table.get(i), deletedVar);

         // add a new row with the list of new values and the probability we calculated
         newRows.add(new ProbRow(newValues, prob));
      }
   }

   // 3. delete the variable given in the list of variables of factor
   for (int i = 0; i < variables.size(); i++)
      if (i != deletedVar)
         newVariables.add(variables.get(i));

   return new Factor(newVariables, newRows);
}

/**
 * Sums the probabilities of all rows
 */
public double sumAllProb() {
   double prob = 0;
   for (ProbRow p : table)
      prob += p.getProb();

   return prob;
}

/**
 * Check if two probability rows can be sum
 */
public boolean canSum(ProbRow p1, ProbRow p2, int deletedVar) {
   assert p1.getValues().size() == p2.getValues().size() : "rows must be of equal
       length";
   for (int i = 0; i < p1.getValues().size(); i++)
      if (i != deletedVar && !p1.getValues().get(i).equals(p2.getValues().get(i)))
         return false;

   return true;
}

/**
 * Make new (string)list of values without the value in the position of deleted
 * variable
 */
public ArrayList<String> makeNewValues(ProbRow p, int deletedVar) {
   ArrayList<String> newValues = new ArrayList<String>();
   for (int i = 0; i < p.getValues().size(); i++)
      if (!(i == deletedVar))
```

```
            newValues.add(p.getValues().get(i));

      return newValues;
   }

   // Factor Product
   public Factor product(Factor f) {
      ArrayList<Variable> newVariables = new ArrayList<Variable>();
      ArrayList<ProbRow> newProb = new ArrayList<ProbRow>();

      ArrayList<Variable> variables2 = f.getVariables();
      ArrayList<ProbRow> table2 = f.getTable();

      ArrayList<Integer> idx1 = new ArrayList<Integer>();
      ArrayList<Integer> idx2 = new ArrayList<Integer>();

      // Check if two factors share same variable
      for (int i = 0; i < variables.size(); i++) {
         for (int j = 0; j < variables2.size(); j++) {
            if (variables.get(i).equals(variables2.get(j))) {
               idx1.add(i);
               idx2.add(j);
            }
         }
      }

      // Compare each row in two factors to product
      for (ProbRow p1 : this.getTable()) {
         for (ProbRow p2 : table2) {
            // check two rows(p1, p2) can be product
            if (canProduct(p1, p2, idx1, idx2)) {
               // if so, make a new (string)list of values in p1 and p2
               // and product their probability
               ArrayList<String> newValues = new ArrayList<String>();
               newValues = makeNewValues(p1, p2, idx2);
               Double prob = p1.getProb() * p2.getProb();
               newProb.add(new ProbRow(newValues, prob));
            }
         }
      }

      newVariables.addAll(getVariables());
      for (int i = 0; i < variables2.size(); i++)
         if (!(idx2.contains(i)))
            newVariables.add(variables2.get(i));

      return new Factor(newVariables, newProb);
   }

   /**
    * Check if two probability rows can be product
    */
```

```
  public boolean canProduct(ProbRow p1, ProbRow p2, ArrayList<Integer> idx1,
      ArrayList<Integer> idx2) {
    for (int i = 0; i < idx1.size(); i++) {
      int commonVal1 = idx1.get(i);
      int commonVal2 = idx2.get(i);
      String valOfF1 = p1.getValues().get(commonVal1);
      String valOfF2 = p2.getValues().get(commonVal2);

      if (!(valOfF1.equals(valOfF2)))
        return false;
    }
    return true;
  }

  /**
   * Make new (string)list of values in two probability rows
   */
  public ArrayList<String> makeNewValues(ProbRow p1, ProbRow p2, ArrayList<Integer> idx2) {
    ArrayList<String> values = new ArrayList<String>();
    values.addAll(p1.getValues());

    for (int i = 0; i < p2.getValues().size(); i++)
      if (!(idx2.contains(i)))
        values.add(p2.getValues().get(i));

    return values;
  }

  // Factor Reduction
  public Factor reduce(Variable ob) {
    ArrayList<Variable> newVariables = variables;
    ArrayList<ProbRow> newRows = new ArrayList<ProbRow>();
    int observedVar = -1;
    for (int i = 0; i < newVariables.size(); i++) {
      if (newVariables.get(i).getName().equals(ob.getName())) {
        observedVar = i;
      }
    }
    for (ProbRow r : table) {
      if (r.getValues().get(observedVar).equals(ob.getObservedValue())) {
        newRows.add(r);
      }
    }
    return new Factor(newVariables, newRows);
  }
}
```

6.6 Whole code Variable class

```java
package varelim;

import java.util.ArrayList;

/**
 * Class to represent a variable.
 *
 * @author Marcel de Korte, Moira Berens, Djamari Oetringer, Abdullahi Ali, Leonieke van
     den Bulk
 */
public class Variable {

   private String name;
   private ArrayList<String> possibleValues;
   private ArrayList<Variable> parents; // Note that parents is not set in the constructor,
       but manually set with setParents() because of the .bif file layout
   private String observedValue;
   private boolean observed = false;

   /**
    * Constructor of the class.
    * @param name, name of the variable.
    * @param possibleValues, the possible values of the variable.
    */
   public Variable(String name, ArrayList<String> possibleValues) {
      this.name = name;
      this.possibleValues = possibleValues;
   }

   /**
    * Transform variable and its values to string.
    */
   public String toString() {
      String valuesString = "";
      for(int i = 0; i < possibleValues.size()-1; i++){
         valuesString = valuesString + possibleValues.get(i) + ", ";
      }
      valuesString = valuesString + possibleValues.get(possibleValues.size()-1);
      return name + " - " + valuesString;
   }

   /**
    * Getter of the values.
    * @return the values of the variable as a ArrayList of Strings.
    */
   public ArrayList<String> getValues(){
      return possibleValues;
   }

   /**
    * Check if string v is a value of the variable.
    * @return a boolean denoting if possibleValues contains string v.
    */
   public boolean isValueOf(String v) {
      return possibleValues.contains(v);
   }

   /**
    * Getter of the amount of possible values of the variable.
    * @return the amount of values as an int.
    */
   public int getNumberOfValues() {
      return possibleValues.size();
   }

   /**
    * Getter of the name of the variable.
    * @return the name as a String.
    */
   public String getName() {
      return name;
   }

   /**
    * Getter of the parents of the variable.
    * @return the list of parents as an ArrayList of Variables.
    */
   public ArrayList<Variable> getParents() {
      return parents;
   }

   /**
    * Setter of the parents of the variable.
    * @param the list of parents as an ArrayList of Variables.
    */
   public void setParents(ArrayList<Variable> parents) {
      this.parents = parents;
   }

   /**
    * Check if a variable has parents.
    * @return a boolean denoting if the variable has parents.
    */
   public boolean hasParents(){
      return parents != null;
   }

   /**
    * Getter for the number of parents a variable has.
    * @return the amount of parents as an int.
    */
   public int getNrOfParents() {
      if(parents != null)
         return parents.size();
```

```
        return 0;
    }

    /**
     * Setter of the observed value of the variable.
     * @param observedValue as a String to which observed value the current value of the
         variable should be set.
     */
    public void setObservedValue(String observedValue) {
        this.observedValue = observedValue;
    }

    /**
     * Getter of the observed value of the variable.
     * @return the value of the variable as a String.
     */
    public String getObservedValue(){
        return observedValue;
    }

    /**
     * Setter for if a variable is observed.
     * @param a boolean denoting if the variable is observed or not.
     */
    public void setObserved(boolean observed) {
        this.observed = observed;
    }

    /**
     * Getter for if a variable is observed.
     * @return a boolean denoting if the variable is observed or not.
     */
    public boolean getObserved() {
        return observed;
    }
}
```

6.7 Whole code Table class

```
package varelim;

import java.util.ArrayList;

/**
 * Class to represent a probability table consisting of probability rows.
 *
 * @author Marcel de Korte, Moira Berens, Djamari Oetringer, Abdullahi Ali, Leonieke van
     den Bulk
 */
```

```java
public class Table {

   private Variable variable;
   private ArrayList<ProbRow> table;

   /**
    * Constructor of the class.
    * @param variable, variable belonging to the current probability table.
    * @param table, table made out of probability rows (ProbRows).
    */
   public Table(Variable variable, ArrayList<ProbRow> table) {
      this.variable = variable;
      this.table = table;
   }

   /**
    * Returns the size of the Table (amount of probability rows).
    * @return amount of rows in the table as an int.
    */
   public int size() {
      return table.size();
   }

   /**
    * Transform table to string.
    */
   public String toString() {
      String tableString = variable.getName() + " | ";
      for (int i = 0; i < variable.getNrOfParents(); i++) {
         tableString = tableString + variable.getParents().get(i).getName();
         if(!(i == variable.getParents().size()-1)) {
            tableString = tableString + ", ";
         }
      }
      for(ProbRow row: table) {
         tableString = tableString + "\n" + row.toString();
      }
      return tableString;
   }

   /**
     * Gets the i'th element from the ArrayList of ProbRows.
     * @param i index as an int.
     * @return i'th ProbRow in Table.
     */
   public ProbRow get(int i) {
      return table.get(i);
   }

   /**
    * Getter of the table made out of ProbRows
    * @return table as an ArrayList of ProbRows.
```

```
     */
    public ArrayList<ProbRow> getTable() {
       return table;
    }

     /**
     * Getter of the variable that belongs to the probability table.
     * @return the variable.
     */
    public Variable getVariable() {
       return variable;
    }

    /**
     * Getter of the parents that belong to the node of the probability table.
     * @return the parents as an ArrayList of Variables.
     */
    public ArrayList<Variable> getParents() {
       return variable.getParents();
    }
}
```

6.8 Whole code ProbRow class

```
package varelim;

import java.util.ArrayList;

/**
 * Class to represent a row of a probability table by its values and a probability.
 *
 * @author Marcel de Korte, Moira Berens, Djamari Oetringer, Abdullahi Ali, Leonieke van
      den Bulk
 */
public class ProbRow {
    private ArrayList<String> values;
    private double prob;

    /**
     * Constructor of the class.
     * @param values, values of the variables (main variable+parents) in the row, of which
        the value of the main variable itself is always first.
     * @param prob, probability belonging to this row of values.
     */
    public ProbRow(ArrayList<String> values, double prob) {
       this.prob = prob;
       this.values = values;
    }

    /**
```

```java
     * Transform probabilities to string.
     */
    public String toString() {
       String valuesString = "";
       for(int i = 0; i < values.size()-1; i++){
          valuesString = valuesString + values.get(i) + ", ";
       }
       valuesString = valuesString + values.get(values.size()-1);
       return valuesString + " | " + Double.toString(prob);
    }

    /**
     * Getter of the values of this probability row.
     * @return ArrayList<String> of values
     */
    public ArrayList<String> getValues() {
       return values;
    }

    /**
     * Getter of the probability of this probability row
     * @return the probability as a double.
     */
    public double getProb() {
       return prob;
    }


}
```

6.9 Whole code NetworkReader class

```java
package varelim;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;

/**
 * Class that reads in a network from a .bif file and puts the variables and probabilities
     at the right places.
 *
 * @author Marcel de Korte, Moira Berens, Djamari Oetringer, Abdullahi Ali, Leonieke van
     den Bulk
 */

public class Networkreader {
```

```
private ArrayList<Variable> vs = new ArrayList<Variable>();
private ArrayList<Table> ps = new ArrayList<Table>();
private ArrayList<ProbRow> probRows;
private String varName;
private String probName;
private ArrayList<Variable> parents = new ArrayList<Variable>();
private int nrOfRows;

/**
 * Constructor reads in the data file and adds the variables and its
 * probabilities to the designated arrayLists.
 *
 * @param file, the name of the .bif file that contains the network.
 */
public Networkreader(String file) {
   BufferedReader br = null;
   try {
      String cur; // Keeping track of current line observed by BufferedReader
      br = new BufferedReader(new FileReader(file));
      try {
         while ((cur = br.readLine()) != null) {
            if (cur.contains("variable")) {
               //Add variable to the list
               varName = cur.substring(9, cur.length() - 2);
               cur = br.readLine();
               ArrayList<String> possibleValues = searchForValues(cur);
               vs.add(new Variable(varName, possibleValues));
            }
            if (cur.contains("{")) {
               parents = new ArrayList<Variable>();
            }
            if (cur.contains("probability")) {
               // Conditional to check for parents of selected variable
               searchForParents(cur);
            }
            if (cur.contains("table")) {
               //Conditional to find probabilities of 1 row and add Probabilities to
               // probability list
               ArrayList<ProbRow> currentProbRows = searchForProbs(cur);
               for(ProbRow p : currentProbRows) {
                  probRows.add(p);
               }
               Table table = new Table(getByName(probName), probRows);
               ps.add(table);
            }
            if (cur.contains(")") && cur.contains("(") && !cur.contains("prob")) {
               // Conditional to find probabilities of more than 1 row;
               // add probabilities to probability list
               ArrayList<ProbRow> currentProbRows = searchForProbs(cur);
               for(ProbRow p : currentProbRows) {
                  probRows.add(p);
               }
               if (probRows.size() == nrOfRows) {
                  Table table = new Table(getByName(probName), probRows);
                  ps.add(table);
               }
            }
         }
      } catch (IOException e) {
      }
   } catch (FileNotFoundException e) {
      System.out.println("This file does not exist.");
      System.exit(0);
   }
}

/**
 * Searches for a row of probabilities in a string
 *
 * @param a string s
 * @return a ProbRow
 */
public ArrayList<ProbRow> searchForProbs(String s) {
   ArrayList<ProbRow> currentProbRows = new ArrayList<ProbRow>();
   int beginIndex = s.indexOf(')') + 2;
   if (s.contains("table")) {
      beginIndex = s.indexOf('e') + 2;
   }

   int endIndex = s.length() - 1;
   String subString = s.substring(beginIndex, endIndex);
   String[] probsString = subString.split(", ");
   double[] probs = new double[probsString.length];
   for (int i = 0; i < probsString.length; i++) {
      probs[i] = Double.parseDouble(probsString[i]);
   }

   if (!s.contains("table")) {
      ArrayList<String> parentsValues = new ArrayList<String>();
      ArrayList<String> nodeValues = new ArrayList<String>();
      beginIndex = s.indexOf('(') + 1;
      endIndex = s.indexOf(')');
      subString = s.substring(beginIndex, endIndex);
      String[] stringValues = subString.split(", ");
      for(String value : stringValues) {
         parentsValues.add(value);
      }
      for(Variable v : vs) {
         if(probName.equals(v.getName())) {
            nodeValues = v.getValues();
         }
      }
      for(int i=0; i<probs.length; i++) {
         parentsValues.add(0,(String) nodeValues.get(i));
         ArrayList<String> currentVal = new ArrayList<String>(parentsValues);
         currentProbRows.add(new ProbRow(currentVal, probs[i] ));
         parentsValues.remove(0);
      }
   }
   else {
      ArrayList<String> values = new ArrayList<String>();
      ArrayList<String> nodeValues = new ArrayList<String>();
      for(Variable v : vs) {
         if(probName.equals(v.getName())) {
            values = v.getValues();
         }
      }
      for(int i=0; i<probs.length; i++) {
         nodeValues.add(values.get(i));
         ArrayList<String> currentVal = new ArrayList<String>(nodeValues);
         ProbRow prob = new ProbRow(currentVal, probs[i]);
         currentProbRows.add(prob);
         nodeValues.clear();
      }
   }
   return currentProbRows;
}

/**
 * Searches for values in a string
 *
 * @param a string s
 * @return a list of values
 */
public ArrayList<String> searchForValues(String s) {
   int beginIndex = s.indexOf('{') + 2;
   int endIndex = s.length() - 3;
   String subString = s.substring(beginIndex, endIndex);
   String[] valueArray = subString.split(", ");
   return new ArrayList<String>(Arrays.asList(valueArray));
}

/**
 * Method to check parents of chosen variable.
 *
 * @param cur, which gives the current line.
 */
public void searchForParents(String cur) {
   if (cur.contains("|")) { // Variable has parents
      extractParents(cur);
   } else { // Variable has no parents
      probName = cur.substring(14, cur.length() - 4);
      for(Variable v : vs) {
         if(probName.equals(v.getName())) {
            nrOfRows = v.getNumberOfValues();
         }
      }
   }
   probRows = new ArrayList<ProbRow>();
}

/**
 * Gets a variable from variable Vs when the name is given
 *
 * @param name
 * @return variable with name as name
 */
private Variable getByName(String name) {
   Variable var = null;
   for (int i = 0; i < vs.size(); i++) {
      if (vs.get(i).getName().equals(name))
         var = vs.get(i);
   }
   return var;
}

/**
 * Extracts parents and puts them in a list of parents of that node.
 *
 * @param cur, a string to extract from
 */
public void extractParents(String cur) {
   probName = cur.substring(14, cur.indexOf("|") - 1);
   Variable var = getByName(probName);
   String sub = cur.substring(cur.indexOf('|') + 2, cur.indexOf(')') - 1);
   while (sub.contains(",")) { // Variable has more parents
      String current = sub.substring(0, sub.indexOf(','));
      sub = sub.substring(sub.indexOf(',') + 2);
      for (int i = 0; i < vs.size(); i++) {
         if (vs.get(i).getName().equals(current)) {
            parents.add(vs.get(i)); // Add parent to list
         }
      }
   }
   if (!sub.contains(",")) { // Variable has no more parents
      for (int i = 0; i < vs.size(); i++) {
         if (vs.get(i).getName().equals(sub)) {
            parents.add(vs.get(i)); //
         }
      }
   }

   var.setParents(parents);
   nrOfRows = computeNrOfRows(probName);
}

/**
```

```
    * Computes the number of rows needed given the current parents
    *
    * @return the number of rows
    */
   private int computeNrOfRows(String probName) {
      int fac = 1;
      for (int i = 0; i < parents.size(); i++) {
         fac = fac * parents.get(i).getNumberOfValues();
      }
      for(Variable v : vs) {
         if(probName.equals(v.getName())) {
            fac = fac * v.getNumberOfValues();
         }
      }
      return fac;
   }

   /**
    * Getter of the variables in the network.
    *
    * @return the list of variables in the network.
    */
   public ArrayList<Variable> getVs() {
      return vs;
   }

   /**
    * Getter of the probabilities in the network.
    *
    * @return the list of probabilities in the network.
    */
   public ArrayList<Table> getPs() {
      return ps;
   }

}
```

6.10 Whole code UserInterface class

```
package varelim;

import java.util.ArrayList;
import java.util.Scanner;

/**
 * Class that handles the communication with the user.
 *
 * @author Marcel de Korte, Moira Berens, Djamari Oetringer, Abdullahi Ali, Leonieke van
     den Bulk
 */
```

```java
public class UserInterface {

   private ArrayList<Variable> vs;
   ArrayList<Table> ps;
   private Variable query = null;
   private ArrayList<Variable> obs = new ArrayList<Variable>();
   private String line;
   private String heuristic;
   private Scanner scan;

   /**
    * Constructor of the user interface.
    * @param vs, the list of variables.
    * @param ps, the list of probability tables.
    */
   public UserInterface(ArrayList<Variable> vs, ArrayList<Table> ps) {
      this.vs = vs;
      this.ps = ps;
   }

   /**
    * Asks for a query from the user.
    */
   public void askForQuery() {
      System.out.println("\nWhich variable(s) do you want to query? Please enter in the
          number of the variable.");
      for (int i = 0; i < vs.size(); i++) {
         System.out.println("Variable " + i + ": " + vs.get(i).getName());
      }
      scan = new Scanner(System.in);
      line = scan.nextLine();
      if (line.isEmpty()) {
         System.out.println("You have not chosen a query value. Please choose a query
             value.");
         askForQuery();
      }
      try {
         int queriedVar = Integer.parseInt(line);
         if (queriedVar >= 0 && queriedVar < vs.size()) {
            query = vs.get(queriedVar);
         } else {
            System.out.println("This is not a correct index. Please choose an index between
                " + 0 + " and "
                 + (vs.size() - 1) + ".");
            askForQuery();
         }
      } catch (NumberFormatException ex) {
         System.out.println("This is not a correct index. Please choose an index between "
             + 0 + " and "
              + (vs.size() - 1) + ".");
         askForQuery();
   }
}

/**
 * Ask the user for observed variables in the network.
 */
public void askForObservedVariables() {

   obs.clear();
   System.out.println("Which variable(s) do you want to observe? Please enter in the
       number of the variable, \n"
         + "followed by a comma and the value of the observed variable. Do not use
             spaces. \n"
         + "If you want to query multiple variables, delimit them with a ';' and no
             spaces.\n"
         + "Example: '2,True;3,False'");
   for (int i = 0; i < vs.size(); i++) {
      String values = "";
      for (int j = 0; j < vs.get(i).getNumberOfValues() - 1; j++) {
         values = values + vs.get(i).getValues().get(j) + ", ";
      }
      values = values + vs.get(i).getValues().get(vs.get(i).getNumberOfValues() - 1);

      System.out.println("Variable " + i + ": " + vs.get(i).getName() + " - " + values);
   }
   scan = new Scanner(System.in);
   line = scan.nextLine();
   if (line.isEmpty()) {
   } else {
      if (!line.contains(",")) {
         System.out.println("You did not enter a comma between values. Please try
             again");
         askForObservedVariables();
         return;
      } else {
         while (line.contains(";")) { // Multiple observed variables
            try {
               int queriedVar = Integer.parseInt(line.substring(0, line.indexOf(",")));
               String bool = line.substring(line.indexOf(",") + 1, line.indexOf(";"));
               changeVariableToObserved(queriedVar, bool);
               line = line.substring(line.indexOf(";") + 1); // Continue
                                                   // with
                                                   // next
                                                   // observed
                                                   // variable.
            } catch (NumberFormatException ex) {
               System.out.println("This is not a correct input. Please choose an index
                   between " + 0 + " and "
                    + (vs.size() - 1) + ".");
               askForObservedVariables();
               return;
            }
            }
            if (!line.contains(";")) { // Only one observed variable
               try {

                  int queriedVar = Integer.parseInt(line.substring(0, line.indexOf(",")));
                  String bool = line.substring(line.indexOf(",") + 1);
                  changeVariableToObserved(queriedVar, bool);
               } catch (NumberFormatException ex) {
                  System.out.println("This is not a correct input. Please choose an index
                      between " + 0 + " and "
                       + (vs.size() - 1) + ".");
                  askForObservedVariables();
               }
            }
         }
      }
   }

   /**
    * Checks whether a number and value represent a valid observed value or not and if so,
        adds it to the
    * observed list. If not, asks again for new input.
    */
   public void changeVariableToObserved(int queriedVar, String value) {
      Variable ob;
      if (queriedVar >= 0 && queriedVar < vs.size()) {
         ob = vs.get(queriedVar);
         if (ob.isValueOf(value)) {
            ob.setObservedValue(value);
            ob.setObserved(true);
         }

         else {
            System.out.println("Apparently you did not fill in the value correctly. You
                typed: \"" + value
                 + "\"Please try again");
            askForObservedVariables();
            return;
         }
         obs.add(ob); // Adding observed variable and it's value to list.
      } else {
         System.out.println("You have chosen an incorrect index. Please choose an index
             between " + 0 + " and "
              + (vs.size() - 1));
         askForObservedVariables();
      }
   }

   /**
    * Print the network that was read-in (by printing the variables, parents and
        probabilities).
    *
```

```
 * @param The list of variables.
 * @param The list of probabilities.
 */
public void printNetwork() {
   System.out.println("The variables:");
   for (int i = 0; i < vs.size(); i++) {
      String values = "";
      for (int j = 0; j < vs.get(i).getNumberOfValues() - 1; j++) {
         values = values + vs.get(i).getValues().get(j) + ", ";
      }
      values = values + vs.get(i).getValues().get(vs.get(i).getNumberOfValues() - 1);
      System.out.println((i + 1) + ") " + vs.get(i).getName() + " - " + values); //
          Printing
                                                         // the
                                                         // variables.
   }
   System.out.println("\nThe probabilities:");
   for (int i = 0; i < ps.size(); i++) {
      if (vs.get(i).getNrOfParents() == 1) {
         System.out.println(ps.get(i).getVariable().getName() + " has parent "
               + vs.get(i).getParents().get(0).getName());
      } else if (vs.get(i).getNrOfParents() > 1) {
         String parentsList = "";
         for (int j = 0; j < vs.get(i).getParents().size(); j++) {
            parentsList = parentsList + vs.get(i).getParents().get(j).getName();
            if (!(j == vs.get(i).getParents().size()-1)) {
               parentsList = parentsList + " and ";
            }
         }
         System.out.println(ps.get(i).getVariable().getName() + " has parents "
               + parentsList);
      } else {
         System.out.println(ps.get(i).getVariable().getName() + " has no parents.");
      }

      Table probs = ps.get(i);
      for (int l = 0; l < probs.size(); l++) {
         System.out.println(probs.get(l));     // Printing
      }                                  // the
      System.out.println();                 // probabilities.
   }
}

/**
 * Prints the query and observed variables given in by the user.
 *
 * @param The query variable(s).
 * @param The observed variable(s).
 */
public void printQueryAndObserved(Variable query, ArrayList<Variable> Obs) {
   System.out.println("\nThe queried variable(s) is/are: "); // Printing
                                              // the
```

```
                                            // queried
                                            // variables.
   System.out.println(query.getName());
   if (!Obs.isEmpty()) {
      System.out.println("The observed variable(s) is/are: "); // Printing
                                               // the
                                               // observed
                                               // variables.
      for (int m = 0; m < Obs.size(); m++) {
         System.out.println(Obs.get(m).getName());
         System.out.println("This variable has the value: " +
              Obs.get(m).getObservedValue());
      }
   }
}

/**
 * Asks for a heuristic.
 */
public void askForHeuristic() {
   System.out.println("Supply a heuristic. Input 1 for least-incoming, 2 for
        fewest-factors and enter for random");
   scan = new Scanner(System.in);
   line = scan.nextLine();
   if (line.isEmpty()) {
      heuristic = "empty";
      System.out.println("You have chosen for random");
   } else if (line.equals("1")) {
      heuristic = "least-incoming";
      System.out.println("You have chosen for least-incoming");
   } else if (line.equals("2")) {
      heuristic = "fewest-factors";
      System.out.println("You have chosen for fewest-factors");
   } else {
      System.out.println(line + " is not an option. Please try again");
      askForHeuristic();
   }
   scan.close();
}

/**
 * Getter of the observed variables.
 *
 * @return a list of observed variables given by the user.
 */
public ArrayList<Variable> getObservedVariables() {
   return obs;
}

/**
 * Getter of the queried variables.
 *
```

```
     * @return the variable the user wants to query.
     */
    public Variable getQueriedVariable() {
       return query;
    }

    /**
     * Getter of the heuristic.
     *
     * @return the name of the heuristic.
     */
    public String getHeuristic() {
       return heuristic;
    }
}
```