

2IDC0 Artificial Intelligence: Bayesian Networks

J.P.H. Snoeren (0772658)

R. Stoof (0767157)

April 2, 2014

This report describes how we implemented the variable elimination algorithm to find probability distributions of certain variables in a Bayesian network, given a number of conditions. First we describe how the implementation of the variable elimination algorithm is structured and discuss the components of this implementation. Then we take a closer look at the operations that we perform on the dataset while executing the variable elimination algorithm. When the concept of our implementation of the variable elimination is clear, we discuss the results of our implementation, compared with the results of the PIT system.

1 Algorithms

1.1 Overview

To find the probability distribution of a certain variable in a Bayesian network, we execute the variable elimination algorithm. The variable elimination algorithm uses the dynamic programming paradigm to reduce work being done multiple times. To execute the variable elimination algorithm we have to evaluate an expression based on the query and the Bayesian network. For each of the variables included in the query (that is, the query variable and the variables in the conditions), the parents and ancestors in the Bayesian network should be included in this expression, since they influence these variables. All vertices that are no parents or ancestors of any of these variables are *not relevant* and thus should not be included in our query. In this way, we reduce the computation time. Our first step will be to find only the *relevant* variables in the network. Then, we use a reversed topological sort to find an order in the Bayesian network in which we evaluate the expression. Once we have found this expression, we evaluate it by multiplying and marginalizing the conditional probability tables (CPTs) corresponding to the variables in the order. Then we finally return the result.

1.2 Finding the relevant variables

We can find the relevant variables in a Bayesian network starting with a certain set of variables by using a bottom-up breadth-first search. The parents of each variable are relevant, so we can add these to the set of relevant variables. But the parents of these parents are also relevant, so we add these as well. We continue to traverse the Bayesian network in this fashion until all vertices that we found are roots (and thus have no parents). These roots correspond to the a priori probabilities in the Bayesian network, which do not depend on any other variables.

Algorithm `FINDRELEVANTVARIABLES` finds the relevant variables in a Bayesian network. Its input is the query variable *queryVar*, the variables in the conditions *givenVars* and the Bayesian network *network*. It maintains a set *R* which contains the relevant variables, and uses an auxiliary

set H that contains the variables corresponding to the vertices on the level that we are inspecting at the moment.

FINDRELEVANTVARIABLES(*queryVar*, *givenVars*, *network*)

```

1  Add queryVar and all variables in givenVars to  $R$ .
2  Let  $H$  be a new set, containing initially the variables in  $R$ .
3  while  $H$  is not empty
4      for all vertices  $v$  in  $H$ 
5          add the parents of  $v$  both to  $R$  and to  $H$ .
6          remove  $v$  from  $H$ 
7  return  $R$ 

```

First we add all variables that are present in the query to R , since these are trivially relevant. The loop in lines 4 - 6 corresponds to adding the parents of all variables in H to R , since these are relevant as well. In line 5 we add the parents to H , such that we inspect the ancestors of the original vertex as well. Note that the implementation of this algorithm includes some details, including a solution to adding variables to H while looping over H .

1.3 Topological sort

The next step towards finding the correct expression to answer our query is to use a reversed topological sort on the Bayesian network. That is, we start at the leaves and add them to a list. Then we find the parents of these leaves and add them to the list, after the leaves. We can use topological sort on the Bayesian network since it is a directed acyclic graph. We call this algorithm reversed topological sort since we treat the Bayesian network as if all edges are reversed.

Algorithm REVERSEDTOPOLOGICALSORT returns a list of variables which contains all variables in the Bayesian network bn (not only the relevant variables). Its sole input is the Bayesian network bn .

REVERSEDTOPOLOGICALSORT(bn)

```

1  Let  $L$  be a new list
2  Let  $V$  be the set of variables in  $bn$ 
3  while  $V$  is not empty
4      add all leaves of  $bn$  using only variables in  $V$  to  $L$ 
5      remove all leaves of  $bn$  from  $V$ 
6  return  $L$ 

```

The algorithm works as follows. Initially we let L , our output list be empty and let V be the set of all variables in the Bayesian network. We add all leaves of the network to L (in any order) and then remove these leaves from the network. The next iteration finds the leaves of the network without the vertices in L , that is, the parents of the original leaves. Continuing in this fashion leads to an ordering in which parents of vertices always appear later than the vertices themselves.

1.4 Finding the leaves

Algorithm REVERSEDTOPOLOGICALSORT depends heavily on finding the leaves in the Bayesian network, but we have not specified how to find these leaves. We can find the leaves easily, since the leaves are the only vertices that have no children. However, in our implementation we do not store the children of each vertex. Therefore we find the leaves by exploiting the following simple observation: The only vertices in the Bayesian network that are not parents of another vertex, are the leaves.

Algorithm FINDLEAVES finds the leaves in a Bayesian network. Its inputs are the Bayesian network bn and a set of variables V that specifies the vertices that we need to look at. All variables that are in the Bayesian network, but not in V are not considered.

FINDLEAVES(bn, V)

```

1  Let Parents be a new set
2  for all variables  $var \in V$ 
3      add all parents of  $var$  to Parents
4  return  $V \setminus Parents$ 

```

The algorithm is conceptually very simple. We store all parents that we can find in the set *Parents*. We check for each variable what the parents are and add them to *Parents*. Then we simply return the set of all variables minus the parents, which results in the leaves as observed above.

1.5 Evaluating the expression

Now we have all tools from the previous subsections in hand, we can construct the formula which will give the probability distribution asked in the query. This formula consists of certain factors. A factor is represented by a CPT, possibly reduced by applying a particular *filter* over that CPT. A filter is a condition on a variable present in the CPT. For example, in the burglary example a CPT may consist of the variables *Alarm*, *Burglary* and *Earthquake* and a possible filter is *Earthquake = yes*. The result will be a new CPT where all rows where *Earthquake = no* are removed. Algorithm VARIABLEELIMINATION finds the expression corresponding to the query and evaluates it.

VARIABLEELIMINATION($bn, queryVar, givenVars$)

```

1   $R = \text{FINDRELEVANTVARIABLES}(queryVar, givenVars, bn)$ 
2   $V = \text{REVERSEDTOPOLOGICALSORT}(bn)$ 
3  Let result be a new factor representing the result
4  result = the first variable  $v$  in  $R$ 
5  if  $v \in givenVars$ 
6      result = FILTER(result,  $v=v.value$ )
7  for all variables  $var \in V \setminus \{v\}$ 
8      if  $var \in R$ 
9          Let factor be a new factor representing the factor of  $var$ 
10         if  $var \in givenVars$ 
11             factor = FILTER(factor,  $var=var.value$ )
12             result = MULTIPLY(factor, result)
13         else if  $var \neq queryVar$ 
14             result = MULTIPLY(factor, result)
15             result = ELIMINATE(result)
16         else
17             result = MULTIPLY(result, factor)
18  return NORMALIZE(result)

```

To find the probability distribution based on the query, we have to multiply and marginalize certain factors. We treat the CPTs in the topologically sorted order that REVERSEDTOPOLOGICALSORT provides us. We initialize our result factor by the first *relevant* variable that appears in the topologically sorted order and filter it if needed. If we encounter a variable which is irrelevant, we will skip it. Note that this does not change the probability required, but it improves computation time slightly. For each next relevant variable, we check whether it is fixed or free. If it is fixed, we multiply the factor with the current factor, where we only use the rows where the variables are in common. This is also referred to as pointwise multiplication. If the variable is free, we multiply the whole factor and then eliminate (or sum out) the variable. In the case of our variable being

the query variable, we do not eliminate the variable (since we would not get a result otherwise), but only multiply without filtering first. We return the result, normalized by dividing by the sum of the probabilities.

In our implementation, instead of returning a tuple which contains the probabilities of the values in the order that they appear in the data set, we return them in arbitrary order with their labels, which is more informative. For example, suppose we ask $P(\text{Burglary} \mid \text{Mary}=\text{yes}, \text{John}=\text{yes})$. Instead of returning the very uninformative output (0.284, 0.716) we return either (yes = 0.284, no = 0.716) or (no = 0.716, yes = 0.284).

1.6 Operations on factors

1.6.1 Pointwise multiplication

+ pseudocode of filter? + pseudocode of multiply

1.6.2 Elimination

+ pseudocode

2 Results

In this section we discuss the results of our implementation by comparing them to the results of the PIT system for the burglary Bayesian network. When executed, we find the values for the queries given in table 1.

Query	Our Probability	Probability PIT
$P(\text{Burglary}=\text{yes} \mid \text{John}=\text{yes}, \text{Mary}=\text{yes})$	0.2842	0.2841
$P(\text{Burglary}=\text{no} \mid \text{John}=\text{yes}, \text{Mary}=\text{yes})$	0.7158	0.7159
$P(\text{Alarm}=\text{yes} \mid \text{Earthquake}=\text{yes})$	0.2907	0.2903
$P(\text{John}=\text{yes} \mid \text{Earthquake}=\text{yes}, \text{Burglary}=\text{yes})$	0.8575	0.8575

Table 1: Results of queries on the burglary Bayesian network using our implementation and the PIT system

We can see that the probability seems correct. The small variations in the probabilities are due to rounding the values.

Next, we use our implementation to answer some queries for the BirthAsphyxia network. The results for some example queries can be found in table 2.

Query	Probability Distribution
P(BirthAsphyxia CardiacMixing=Mild, CO2=Low)	(yes = 0.1405, no = 0.8595)
P(ChestXray Grunting=yes)	(normal = 0.1483, Oligaemic = 0.2329, Plethoric = 0.1336, Grd.Glass = 0.151, Asy/Patchy = 0.3342)
P(RUQO2 HypoxiaInO2=Severe)	(<5 = 0.5, 5-12 = 0.4, 12+ = 0.1)
P(Disease GruntingReport=yes, LowerBodyO2=5-12, RUQO2=12+, CO2Report=>=7.5, XrayReport=Plethoric)	(PFC = 0.0839, TGA = 0.3402, Fallot = 0.1565, PAIVS = 0.1225, TAPVD = 0.1084, Lung = 0.1885)

Table 2: Results for some example queries found by our implementation

3 Contributions

During the coding of this assignment, we did most of the code together, in order to both think about the correct implementation and make it more probable to find bugs early on. In this way we can also discuss certain decisions quickly. This report has been written by us together, meaning that we made some sections by ourselves and then checked each others sections to improve them.

4 Manual

In this section, we give a short manual on how to reproduce the results of the project.

1. Run the program
2. Provide a query in the form "P(" *QueryVar* "|" < *Var=Value*>+ ")"
3. Our program will return the probability distribution over *QueryVar*. The order of the values is the same as the order in the data set.