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 the variable elimination is clear, we discuss the results of our implementation, compared with some 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 variables 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 FindRelevantVeriables 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.

ReversedToplogicalSort(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 ReversedToplogicalSort 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 formula

+ pseudocode

1.6 Operations on factors

1.6.1 Multiply

+ pseudocode

1.6.2 Marginalize

+ pseudocode

2 Results

Describe the results of our implementation and the PIT system on the burglary network, and the results on the BirthAsphyxia.

3 Contributions

Blabla.

4 Manual

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

1.
2.
3.
4.

5.