Variable Elimination for Bayesian Networks

Bram Pulles – S1015194

December 19, 2022

Contents

1 Introduction

A Bayesian network is a probabilistic model that represents conditional dependencies between a set of variables through the use of a directed acyclic graph. Such a network can be utilised to calculate the joint probability of a set of variables given certain evidence. There are several algorithms in existence which can infer exactly this information from a Bayesian network, variable elimination being one of them. In this report we explain how variable elimination works and we provide a reference implementation.

To this end, we describe the variable elimination algorithm in section 2. After this, we talk about the implementation of variable elimination in 3. Lastly, we consider the conclusions which can be drawn from this work in section 4

2 Algorithm

The variable elimination algorithm, shown in block 1, operates on factors, variables, a query, and evidence. In a Bayesian network, every node is a variable with a domain of possible values and a corresponding conditional probability table, which is translated into a factor with the same properties. The query is a set of variables for which we want to compute the likelihood given a particular set of evidence. Evidence consists of variable-value pairs that fixate the variables to the provided values.

Algorithm 1 Variable Elimination

Require: Bayesian network factors and variables, query and evidence
Ensure: probability table of the query variables given the evidence

```
1:  function VE(factors, variables, query, evidence)
2:      barren := BARREN(variables, query, evidence)
3:      factors := reduce all nonbarren factors using the evidence
4:      order := variables \ barren ∪ query ∪ evidence
5:      for φ ∈ order do
6:          factor := product of all factors containing φ
7:          factor := marginalize φ in factor
8:          factors := remove all factors containing φ
9:          factors := add new factor
10:     end for
11:     factor := product of all factors
12:     factor := normalize factor
13:     return factor
14: end function
```

The first step of the algorithm is to identify all the barren variables, which are variables that do not need to be considered when determining the likelihood of the query given the evidence. Removing the computation steps related to these variables can significantly reduce the overall cost. A barren variable is defined as any variable that is neither queried nor observed, and has either no children or only barren nodes as children. The definition of a barren variable allows for the use of a recursive algorithm, which is described in block 2.

Algorithm 2 Barren

```
1: function RECURS(variable, nonbarren, barren)
2:     if variable ∈ nonbarren then
3:         for child ∈ variable.children do
4:             RECURS(child, nonbarren, barren)
5:         end for
6:         return false
7:     end if
8:     if variable.children = ∅ then
9:         barren := barren ∪ variable
10:        return true
11:    end if
12:    if ∀ child ∈ variable.children [RECURS(child, nonbarren, barren)] then
13:        barren := barren ∪ variable
14:        return true
15:    end if
16:    for child ∈ variable.children do
17:        RECURS(child, nonbarren, barren)
18:    end for
19:    return false
20: end function
21: function BARREN(variables, query, evidence)
22:    nonbarren := query ∪ evidence
23:    barren := ∅
24:    parents := ∀ variable [variable.parents = ∅]
25:    for variable ∈ parents do
26:        if RECURS(variable, nonbarren, barren) then
27:            barren := barren ∪ variable
28:        end if
29:    end for
30:    return barren
31: end function
```

The second step of the algorithm is to reduce all the factors of nonbarren variables using the evidence. This means that the evidence variables are set to their provided value in all factors.

The third step of the algorithm determines the variables which need to be marginalized. These are all the nonbarren variables that are not queried or in the evidence. In this step it is also possible to give an ordered set, as the order greatly improves the performance of the algorithm. In our implementation, the order is given by the user or an arbitrary order is chosen.

Now the main loop of the algorithm begins. During each iteration all factors containing the current variable φ are multiplied together. Meaning all rows compatible with each other are multiplied together creating a new factor. The variable φ is marginalized from the product factor. The marginalized factor is added back to the factors list while all other factors containing φ are removed.

In the final step we multiply all the remaining factors together and normalize the resulting factor. This factor now contains the likelihood of our query given the evidence in a nice table.

3 Implementation

The implementation of the variable elimination algorithm is done in Python, it can compute barren nodes, supports a set of query variables, a set of evidence pairs and works for any variable domain size. There is also support for writing the output to a file or the prompt, and three different verbosity levels: quiet (0), brief (1) and elaborate (2).

The code is stacked with comments to denote what is happening. Nevertheless, it might be worth to note that the factor is implemented as a list of variables, reduced variables and values. The values are implemented as an ordered list of the probability column for the factor. The comments in the code gives more detail about how this is used to compute the three operations on factors. A lot of effort has been put in to make it readable.

Given that more than half of the code is tests, we got relatively high confidence in the correctness of the code. Tests are written for five algorithms: all three factor operations, the barren nodes computation and the variable elimination algorithm. The former two are relatively straightforward. The variable elimination algorithm is tested using an oracle. The oracle is formed from the code of AIPython2. Sadly, this oracle does not support multiple query variables and only contains the alarm network for which we also have a `bif` file available. However, we do use this network to its fullest. The tester generates a test for *every* possible configuration of query and evidence variables that the oracle supports for the alarm network. All tests compare the output of our program with the oracle and verifies the answers are the same, which luckily, is the case.

The tests can be run using `python -m unittest discover tests`. Using the `python src/main.py` you can interface with our implementation and with the oracle. The `survey.bif` network can be used to verify correctness when using domains bigger than two.

4 Conclusion

In conclusion, the variable elimination algorithm is a powerful method for calculating the joint probability of a set of variables in a Bayesian network, given certain evidence. In this work, an oracle was used to validate the results of the algorithm. While the oracle was not complete and did not cover all possible cases, it was still useful in assessing the accuracy of the algorithm. This highlights the importance of having some form of ground truth for testing and validating the results of the algorithm, even if it is not comprehensive. The provided reference implementation serves as a useful guide for implementing the algorithm in practice, and careful testing and validation using appropriate methods can help ensure the accuracy of the results.