Assignment 3: Variable Elimination Algorithm Project Report

Andrea Minichova - s1021688

February 15, 2022

Contents

1 Introduction

In this report, we will discuss the implementation of the Variable elimination algorithm (VE). In section 2, we describe the concepts of Bayesian Networks (BNs), factor calculus, and we provide the explanation of VE itself. In Section 3, we discuss our implementation of VE. In section 4, we show results of several tests and reason about the results. Lastly, we conclude the report by stating possible improvements to the implementation.

2 Project description

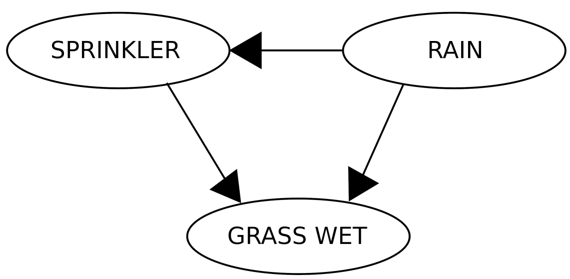
VE algorithm serves to infer any prior probability given a Bayesian network. The algorithm iteratively eliminates variables according to the rules of factor calculus.

2.1 Bayesian Networks

Bayesian network is an acyclic graph, where nodes are the variables. There is an arc from a parent to a child, denoting conditional dependence. Associated with the BN are the conditional probability tables (CPTs) which map each variable-value combination to a real number representing the probability [2].

Below is an example of a BN from [6]. Here, variable Sprinkler is dependant on Rain, Grass Wet is dependant on both Sprinkler and Rain and Rain is not dependant on any variable since it does not have a parent. Notice that each variable has a corresponding CPT.

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Figure 1: Bayesian Network: Example

2.2 Factor Calculus

In order to perform the VE algorithm, we can temporarily convert the CPTs into factors and make use of the calculus. A factor is a function from a set of random variables into a number, but it is no longer necessarily a probability distribution, unline in CPTs.

During the execution of the VE algorithm, we will need three different operations on factors:

2.2.1 Factor Product

Multiplication of two factors, say $f_1(A, B)$ and $f_2(B, C)$ results in factor $f_3(A, B, C)$ where $f3(a, b, c) = f1(a, b) \cdot f_2(b, c)$ for all $a \in A$, $b \in B$ and $c \in C$. See example below [5]:

f_1

a_1	b_1	0.5
a_1	b_2	0.8
a_2	b_1	0.1
a_2	b_2	0
a_3	b_1	0.3
a_3	b_2	0.9

x

f_2

b_1	c_1	0.5
b_1	c_2	0.7
b_2	c_1	0.1
b_2	c_2	0.2

=

f_3

a_1	b_1	c_1	0.5*0.5 = 0.25
a_1	b_1	c_2	0.5*0.7 = 0.35
a_1	b_2	c_1	0.8*0.1 = 0.08
a_1	b_2	c_2	0.8*0.2 = 0.16
a_2	b_1	c_1	0.1*0.5 = 0.05
a_2	b_1	c_2	0.1*0.7 = 0.07
a_2	b_2	c_1	0*0.1 = 0
a_2	b_2	c_2	0*0.2 = 0
a_3	b_1	c_1	0.3*0.5 = 0.15
a_3	b_1	c_2	0.3*0.7 = 0.21
a_3	b_2	c_1	0.9*0.1 = 0.09
a_3	b_2	c_2	0.9*0.2 = 0.18

Figure 2: Factor Product: Example

2.2.2 Factor Marginalization

Summing out a factor, e.i. factor marginalization of variable C in factor $f_3(A, B, C)$ results in factor $f_4(A, B)$: $\Sigma_B f_3(A, B, C) = f_4(A, C)$. Here, we sum out each row in which A and B are equal and C has a unique value. See example below [5]:

f_3

a_1	b_1	c_1	0.5*0.5 = 0.25
a_1	b_1	c_2	0.5*0.7 = 0.35
a_1	b_2	c_1	0.8*0.1 = 0.08
a_1	b_2	c_2	0.8*0.2 = 0.16
a_2	b_1	c_1	0.1*0.5 = 0.05
a_2	b_1	c_2	0.1*0.7 = 0.07
a_2	b_2	c_1	0*0.1 = 0
a_2	b_2	c_2	0*0.2 = 0
a_3	b_1	c_1	0.3*0.5 = 0.15
a_3	b_1	c_2	0.3*0.7 = 0.21
a_3	b_2	c_1	0.9*0.1 = 0.09
a_3	b_2	c_2	0.9*0.2 = 0.18

f_4

a_1	c_1	0.25+0.08 = 0.33
a_1	c_2	0.35+0.16 = 0.51
a_2	c_1	0.05+0 = 0.05
a_2	c_2	0.07+0 = 0.07
a_3	c_1	0.15+0.09 = 0.24
a_3	c_2	0.21+0.18 = 0.39

Figure 3: Factor Marginalization: Example

2.2.3 Factor Reduction

Factor reduction based on evidence reduces the factor by removing the rows which contradict our evidence. See example below [5]:

a_1	b_1	c_1	0.5*0.5 = 0.25
a_1	b_1	c_2	0.5*0.7 = 0.35
a_1	b_2	c_1	0.8*0.1 = 0.08
a_1	b_2	c_2	0.8*0.2 = 0.16
a_2	b_1	c_1	0.1*0.5 = 0.05
a_2	b_1	c_2	0.1*0.7 = 0.07
a_2	b_2	c_1	0*0.1 = 0
a_2	b_2	c_2	0*0.2 = 0
a_3	b_1	c_1	0.3*0.5 = 0.15
a_3	b_1	c_2	0.3*0.7 = 0.21
a_3	b_2	c_1	0.9*0.1 = 0.09
a_3	b_2	c_2	0.9*0.2 = 0.18

f_5

a_1	b_1	0.25
a_1	b_2	0.08
a_2	b_1	0.05
a_2	b_2	0
a_3	b_1	0.15
a_3	b_2	0.09

Figure 4: Factor Reduction: Example

2.3 Variable Elimination Algorithm

The VE algorithm serves to infer any prior probability from a given bayesian network. The algorithm is exponential, but could be made more efficient given a good ordering of elimination variables. See below the pseudocode [3]:

procedure *VE_BN*(Vs, Ps, e, Q)
  Inputs
    Vs: set of variables
    Ps: set of factors representing the conditional probabilities
    e: the evidence, a variable-value assignment to some of the variables
    Q: a query variable
  Output
    posterior distribution on Q
  $Fs := Ps$ ▷ Fs is the current set of factors
  for each $X \in Vs - \{Q\}$ using some elimination ordering **do**
    if X is observed **then**
      for each $F \in Fs$ that involves X **do**
        assign X in F to its observed value in e
    else
      $Rs := \{F \in Fs : F \text{ involves } X\}$
      let T be the product of the factors in Rs
      $N := \sum_X T$
      $Fs := Fs \setminus Rs \cup \{N\}$
  let T be the product of the factors in Fs
  $N := \sum_Q T$
  return T/N

Figure 5: Variable Elimination Algorithm: Pseudocode

The algorithm first initialized the factors, which are initially just the CPTs. If there are observed variables (evidence), the factors are reduced with this evidence accordingly. The algorithm then loops over the variables to be eliminated (given the specified ordering). For each variable, the factor product is computed from all factors containing this variable. The factor is then marginalized over the current variable. Factors which were used in the current computation are removed from the list of factors and the newly computed factor is added. In the end, only one last factor remains, which becomes a probability distribution after normalization step and the algorithm is finished.

3 Implementation

This section describes our implementation of the VE algorithm in Python. Code snippets are provided alongside the explanation of the implementation.

3.1 Structure

The project consists of three Python modules. The main module `run` serves to read a network, set the query, observed variables and elimination ordering, and run the algorithm. The `bayesnet` module was provided and serves to read the provided BN into a dictionary of probabilities (these are implemented as `pandas` dataframes. The last module, `variable_elim` implements the factor calculus as well as the VE. The decision was made to also use dataframes for factors, which made it obsolete to create a separate class for them. Factors are stored in a dictionary. Each factor can be accessed by its key, where key represents the index (for easy identification) and the variables present in the factor.

3.2 Factor Calculus Implementation

3.3 Factor Product

In our implementation, we first generate a final factor which we will populate with probabilities as we perform the multiplication. This factor is created as a truth table from all variables present in factors to be multiplied. This is achieved through the `generate_factor` function.

Then for each row of this product, we iterate over the factors and multiply the intermediate probability by the probability from the row which matched the variable-value pairs in the product (this is determined by the `is_in` function). Finally, we update the product with the newly computed probabilities. See the code snippet below for the exact implementation.

```
    def multiply(self, factors):
        """"
        Return a factor (and its variables) which is a product of 'factors'
        """
        # Generate a new factor which we will populate with multiplied probabilities
        vars = []
        for key in factors:
            vars.extend(list(self.factors[key].columns[:-1]))
        vars = list(set(vars))
        product = self.generate_factor(vars)

        # Multiply probabilities and return the final product
        probabilities = []
        product_prob = current_prob = 1
        for i in range (0,product.shape[0]):
            product_row = product.iloc[[i]]
            for key in factors:
                for j in range(0, self.factors[key].shape[0]):
                    current_row = self.factors[key].iloc[[j]]
                    if self.is_in(current_row, product_row):
                        current_prob = float(current_row.iloc[0]['prob'])
                        product_prob = product_prob * current_prob
                        break
            probabilities.append(product_prob)
            product_prob = 1
        product['prob'] = probabilities
        return vars, product
```

3.4 Factor Marginalization

In factor marginalization, we consider every combination of two rows of a product and sum out the given variable in case the other variables from the rows have identical values. This condition is determined by the function `can_sum_out`. Each time we append the new probability to our new rows. Then we create a new factor out of these rows and return it.

```
    def sum_out(self, var, factor):
        """"
        Return a factor (and its variables) in which 'var' was summed out of 'factor'
        """
        vars = [x for x in list(factor.columns[:-1]) if x != var]
        data = []
        for i in range (0,factor.shape[0]):
            for j in range (i+1,factor.shape[0]):
                if self.can_sum_out(factor, i, j, vars):
                    sum_prob = factor.iloc[i]['prob'] + factor.iloc[j]['prob']
                    row = []
                    for v in vars:
                        row.append(factor.loc[factor.index[i], v])
                    row.append(str(sum_prob))
                    data.append(row)
        sum_f = pd.DataFrame(data, columns = vars + ['prob'])
        return vars, sum_f
```

3.5 Factor Reduction

Factor reduction simply drops the rows where the variable-value pairs do not much our evidence. This happens when initializing factors, but also when generating a new factor before factor multiplication. The following snippet is taken from the `init_factors` function and demonstrates how all factors are updated based on evidence.

```
# Reduce the factors given the observation
self.factors = new_factors
for key in self.factors:
    for o in observed:
        if o in key[1]:
            self.factors[key].drop(self.factors[key].index[self.factors[key][o] != observed[o]],
             inplace = True)
```

3.6 Variable Elimination Implementation

Following code snippet shows our implementation of the VE algorithm (without the print statements for logs). On line **6**, we loop through variables in their specific elimination order. On each such variable, we perform factor product (line **9**) and factor marginalization (line **10**). The new factor is added (line**11**) while the old ones are removed (lines **12-13**). Finally, once we looped through all the variables, we perform the normalizaiton step and print the resulting probability distribution.

```
1         if observed:
2             self.observed = observed
3         self.init_factors(observed)
4
5         i = len(self.factors) # Currently the highest factor index
6         for v in elim_order:
7             if v != query and v not in observed.keys():
```

```
            factors_with_v = self.get_factors(v)
            vars, mult_factor = self.multiply(factors_with_v)
            vars, sum_factor = self.sum_out(v, mult_factor)
            self.factors[i, tuple(vars)] = sum_factor
            for key in factors_with_v:
                self.factors.pop(key)
            i += 1
    vars, final_prob = self.multiply(self.factors.keys())

    final_prob['prob'] = pd.to_numeric(final_prob['prob'], downcast="float")
    total = final_prob['prob'].sum()
    final_prob['prob'] = final_prob['prob'] / total
    print(f'\n {final_prob}')
```

4 Tests

Several tests were made to exhibit the behaviour of the VE algorithm, using the `earthquake` network [4]. To follow the steps of the algorithm, we keep a log of the computations. Find an example of such log in **Appendix**. Here, the algorithm is run with the default ordering, the query variable is `MaryCalls` and the variable `JohnCalls` is observed to be true.

Keeping the same query and removing the evidence, we performed a few tests on different elimination orders. The following table shows the elimination orders and the maximum factor size they create. The connection between these concepts is explained in the section **4.2**

4.1 Results

Ordering	Biggest factor size
'Alarm', 'Burglary', 'Earthquake', 'Johncalls'	5
'Alarm', 'Burglary', 'Johncalls', 'Earthquake'	5
'Alarm', 'Earthquake', 'Burglary', 'Johncalls'	5
'Alarm', 'Earthquake', 'Johncalls', 'Burglary'	5
'Alarm', 'Johncalls', 'Burglary', 'Earthquake'	5
'Alarm', 'Johncalls', 'Earthquake', 'Burglary'	5
'Burglary', 'Alarm', 'Earthquake', 'Johncalls'	4
'Burglary', 'Alarm', 'Johncalls', 'Earthquake'	4
'Burglary', 'Earthquake', 'Alarm', 'Johncalls'	3
'Burglary', 'Earthquake', 'Johncalls', 'Alarm'	3
'Burglary', 'Johncalls', 'Alarm', 'Earthquake'	4
'Burglary', 'Johncalls', 'Earthquake', 'Alarm'	3
'Earthquake', 'Alarm', 'Burglary', 'Johncalls'	4
'Earthquake', 'Alarm', 'Johncalls', 'Burglary'	4
'Earthquake', 'Burglary', 'Alarm', 'Johncalls'	3
'Earthquake', 'Burglary', 'Johncalls', 'Alarm'	3
'Earthquake', 'Johncalls', 'Alarm', 'Burglary'	4
'Earthquake', 'Johncalls', 'Burglary', 'Alarm'	3
'Johncalls', 'Alarm', 'Burglary', 'Earthquake'	5
'Johncalls', 'Alarm', 'Earthquake', 'Burglary'	5
'Johncalls', 'Burglary', 'Alarm', 'Earthquake'	4
'Johncalls', 'Burglary', 'Earthquake', 'Alarm'	3
'Johncalls', 'Earthquake', 'Alarm', 'Burglary'	4
'Johncalls', 'Earthquake', 'Burglary', 'Alarm'	3

4.2 Interpretation

Variable elimination order has an effect on factor size, which in turn has an effect on complexity (the larger the factors are, the more multiplications we need). There are different heuristics we could experiment with, most common ones are: [3]

- Minimum Degree: Eliminate the variable which results in constructing the smallest factor possible. [1]
- Minimum Fill: By constructing an undirected graph showing variable relations expressed by all CPTs, eliminate the variable which would result in the least edges to be added post elimination. [1]

In this project, we have not tried different heuristics, but we can see effects of the orderings in the size of the biggest factor. In general, eliminating variables which are contained in smallest number of factors work best. We can see that `'Alarm'` is present in 3 factors, `'Burglary'` is present in 2 as well as `'Earthquake'` and the remaining variables are each present in 1 factor only.

Looking in the table, we can see that eliminating `'Alarm'` first is always a bad idea as it creates the biggest factor each time. When taking other variables first, putting `'Alarm'` at the last (or at least second-to-last) position gives us factor of size 3. Similarly, all the orderings where `'Alarm'` is at the last place give us the best factor size.

5 Conclusions

We have described our implementation of VE algorithm. We have seen that experimenting with different elimination orderings can be beneficial. Finding an optimal ordering is again exponential, however, it would be interesting to experiment with heuristics for orderings. Another improvement to the implementation would be to allow non-binary variable values. A better knowledge of pandas dataframes could improve the code further by eliminating some unnecessary or long code.

References

[1] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[2] David Poole, Alan Mackworth. Artificial intelligence: Foundations of computational agents: Belief networks. http://artint.info/2e/html/ArtInt2e.Ch8.S3.html. [Online; accessed 09-February-2022].

[3] David Poole, Alan Mackworth. Artificial intelligence: Foundations of computational agents: Belief networks. https://artint.info/2e/html/ArtInt2e.Ch8.S4.SS1.html. [Online; accessed 09-February-2022].

[4] Marco Scutari. bnlearn - an r package for bayesian network learning and inference. https://www.bnlearn.com/bnrepository/discrete-small.html#earthquake. [Online; accessed 09-February-2022].

[5] Paul Kamsteeg and Elena Marchiori. Ai: Principles and techniques, bayesian networks 2 (slides). https://brightspace.ru.nl/d2l/le/content/260099/viewContent/1488692/View. [Online; accessed 09-February-2022].

[6] Wikipedia contributors. Bayesian network. https://en.wikipedia.org/wiki/Bayesian_network. [Online; accessed 09-February-2022].

Appendix

```
-----------------------------
|  Ntework specifications:  |
-----------------------------

Nodes:
['Burglary', 'Earthquake', 'Alarm', 'JohnCalls', 'MaryCalls']

Values:
{'Burglary': ['True', 'False'], 'Earthquake': ['True', 'False'],
'Alarm': ['True', 'False'], 'JohnCalls': ['True', 'False'], 'MaryCalls': ['True', 'False']}

Parents:
{'Burglary': [], 'Earthquake': [], 'Alarm': ['Burglary', 'Earthquake'],
'JohnCalls': ['Alarm'], 'MaryCalls': ['Alarm']}

Probabilities:
{'Alarm':    Alarm Burglary Earthquake   prob
0   True     True       True  0.950
1  False     True       True  0.050
2   True    False       True  0.290
3  False    False       True  0.710
4   True     True      False  0.940
5  False     True      False  0.060
6   True    False      False  0.001
7  False    False      False  0.999,
 'Burglary':   Burglary  prob
0     True  0.01
1    False  0.99,
 'Earthquake':   Earthquake  prob
0       True  0.02
1      False  0.98,
 'JohnCalls':   JohnCalls  Alarm  prob
0      True   True  0.90
1     False   True  0.10
2      True  False  0.05
3     False  False  0.95,
 'MaryCalls':   MaryCalls  Alarm  prob
0      True   True  0.70
1     False   True  0.30
2      True  False  0.01
3     False  False  0.99}

------------------------------------
|  Variable Elimination Algorithm  |
```

```
------------------------------------

A) The query variable: MaryCalls

B) The observed variables: {'JohnCalls': 'True'}

D) The factors:

{(0, ('Burglary',)):   Burglary  prob
0     True  0.01
1    False  0.99,
 (1, ('Earthquake',)):   Earthquake  prob
0       True  0.02
1      False  0.98,
 (2, ('Alarm', 'Burglary', 'Earthquake')):    Alarm Burglary Earthquake   prob
0   True     True       True  0.950
1  False     True       True  0.050
2   True    False       True  0.290
3  False    False       True  0.710
4   True     True      False  0.940
5  False     True      False  0.060
6   True    False      False  0.001
7  False    False      False  0.999,
 (3, ('JohnCalls', 'Alarm')):   JohnCalls  Alarm  prob
0      True   True  0.90
2      True  False  0.05,
 (4, ('MaryCalls', 'Alarm')):   MaryCalls  Alarm  prob
0      True   True  0.70
1     False   True  0.30
2      True  False  0.01
3     False  False  0.99}

E) The elimination ordering: ['Burglary', 'Earthquake', 'Alarm', 'JohnCalls', 'MaryCalls']

----------------------------
|  F) The elimination loop  |
----------------------------

The variable to eliminate: Burglary

Factors to multiply:
[(0, ('Burglary',)), (2, ('Alarm', 'Burglary', 'Earthquake'))]

Factor after multiplication:
   Alarm Earthquake Burglary     prob
0   True       True     True  0.00950
```

```
1   True       True    False  0.28710
2   True      False     True  0.00940
3   True      False    False  0.00099
4  False       True     True  0.00050
5  False       True    False  0.70290
6  False      False     True  0.00060
7  False      False    False  0.98901

Factor after summing out Burglary:
   Alarm Earthquake                prob
0   True       True              0.2966
1   True      False             0.01039
2  False       True  0.7033999999999999
3  False      False             0.98961

New factors:
{(1, ('Earthquake',)):   Earthquake  prob
0       True  0.02
1      False  0.98,
 (3, ('JohnCalls', 'Alarm')):   JohnCalls  Alarm  prob
0      True   True  0.90
2      True  False  0.05,
 (4, ('MaryCalls', 'Alarm')):   MaryCalls  Alarm  prob
0      True   True  0.70
1     False   True  0.30
2      True  False  0.01
3     False  False  0.99,
 (5, ('Alarm', 'Earthquake')):    Alarm Earthquake                prob
0   True       True              0.2966
1   True      False             0.01039
2  False       True  0.7033999999999999
3  False      False             0.98961}

--------------------
|  Next iteration  |
--------------------

The variable to eliminate: Earthquake

Factors to multiply:
[(1, ('Earthquake',)), (5, ('Alarm', 'Earthquake'))]

Factor after multiplication:
   Alarm Earthquake      prob
0   True       True  0.005932
1   True      False  0.010182
```

```
2  False       True  0.014068
3  False      False  0.969818

Factor after summing out Earthquake:
   Alarm               prob
0   True          0.0161142
1  False  0.9838857999999999

New factors:
{(3, ('JohnCalls', 'Alarm')):   JohnCalls  Alarm  prob
0      True   True  0.90
2      True  False  0.05,
 (4, ('MaryCalls', 'Alarm')):   MaryCalls  Alarm  prob
0      True   True  0.70
1     False   True  0.30
2      True  False  0.01
3     False  False  0.99,
 (6, ('Alarm',)):    Alarm               prob
0   True          0.0161142
1  False  0.9838857999999999}

--------------------
|  Next iteration  |
--------------------

The variable to eliminate: Alarm

Factors to multiply:
[(3, ('JohnCalls', 'Alarm')), (4, ('MaryCalls', 'Alarm')), (6, ('Alarm',))]

Factor after multiplication:
   Alarm JohnCalls MaryCalls      prob
0   True      True      True  0.010152
1   True      True     False  0.004351
4  False      True      True  0.000492
5  False      True     False  0.048702

Factor after summing out Alarm:
  JohnCalls MaryCalls                  prob
0      True      True  0.010643888899999999
1      True     False          0.0530531811

New factors:
{(7, ('JohnCalls', 'MaryCalls')):   JohnCalls MaryCalls                  prob
0      True      True  0.010643888899999999
1      True     False          0.0530531811}
```

```
--------------------
|  Next iteration  |
--------------------

Factors to multiply:
dict_keys([(7, ('JohnCalls', 'MaryCalls'))])

Factor product after the final multiplication:
  JohnCalls MaryCalls      prob
0      True      True  0.010644
1      True     False  0.053053

-----------------------------------------------
|  G) The resulting CPT after normalization:  |
-----------------------------------------------

   JohnCalls MaryCalls      prob
0      True      True  0.167102
1      True     False  0.832898

Done!
```