AI PRINCIPLES & TECHNIQUES

Variable Elimination

Quince Kersten *S1073935*
Gregors Lasenbergs *S1075747*
Radboud University *March 8, 2024*

1 Introduction

Estimating probabilistic values is a difficult task, especially when considering multiple influencing factors. Bayesian networks serve as powerful tools for modeling uncertain relationships among variables like these. It is however often difficult to extract meaningful insights from these networks. A solution for this is variable elimination. The variable elimination algorithm proves invaluable in estimating conditional or marginal probability distributions, applicable to both Bayesian and Markov networks. Operating within a graphical model framework, it systematically removes variables by computing intermediate factors and iteratively combining them. Through this process, the algorithm progressively refines the computation until arriving at the final marginal distribution.

Contents

2 Method

2.1 Getting familiar with the code

At first, we examined the provided code, taking note of the classes and variables. These are explained in the sections below.

2.1.1 Read bayesnet

The class `read_bayesnet` is already completely given to us. It provides a brief overview of the Bayesian network that we will be examining for this task. Three dictionaries are included in it: `parents`, `probabilities`, and `variables`. Upon reading the file, the values will be entered into each of them. The methods for parsing the parents, probabilities, and variables are listed below. Additionally, it features a function `node` that outputs the network's variable names.

2.1.2 Run

The class `Run` gives us an overview of how the variable elimination algorithm should be implemented. The `earthquake.bif` file's Bayesian Network will be read and prepared in accordance with the implementation. Some parts of this file still had to be implemented.

2.1.3 Variable Elimination

In the Variable Elimination (VE) class provided, an initialization method accepts a single argument, `network`, representing the Bayesian Network upon which the VE algorithm will operate.
Subsequently, the `run()` function is defined with parameters `query`, `observed`, and `elim_order`. The `query` parameter designates the variable under investigation, aiming to ascertain its values. The `observed` parameter comprises a dictionary containing variables with their determined values. Lastly, the `elim_order` parameter can either be a specified list detailing the elimination order or a function responsible for determining the elimination order dynamically during runtime.
The function produces a variable with the `query` variable's probability distribution in its output.

2.2 Variable elimination algorithm

Variable elimination (VE) is a critical algorithm used in probabilistic graphical models like Bayesian networks. It's all about efficiently answering questions by getting rid of unnecessary variables one at a time. How well it works depends a lot on the order we choose to eliminate variables, often picked using smart guesswork.

2.2.1 Factors

In VE, we use factors, which are multi-dimensional tables holding different values for different combinations of variables. In Bayesian networks, these factors represent the chances of different events happening given certain conditions. There are three main things we do with factors:
1. **Reduction**: When a variable is observed, the factor is made to preserve only the rows corresponding to the observed value, subsequently removing the observed variable from the table to streamline computation. For example given A = A_1:

$$f_1(A, B, C) = f_2(B, C)$$

2. **Marginalization**: Also known as summing-out, this operation eliminates a single variable from a set of factors, resuling in a set of factors with reduced dimensions. For example:

$$\sum_A f_1(A,B,C) = f_2(B,C)$$

3. **Multiplication**: Represented as factor product, this operation combines two or more factors, retaining all possible combinations of rows, thereby facilitating joint distribution computation.

$$f_1(A,B) \cdot f_2(B,C) = f_3(A,B,C)$$

The Variable Elimination Algorithm executes these operations in a systematic manner, as outlined below:
1. **Initialization**: The algorithm is primed with a set of variables, factors, observed variables, and the query variable.
2. **Elimination Ordering**: An elimination order for each variable is determined, either randomly or through heuristics.
3. **Iteration**: - For each variable in the elimination order: - If the variable is observed, the involved factors are adjusted accordingly, combining them and reducing dimensions. - If the variable is unobserved, factors involving it are multiplied to create a composite factor, followed by reduction and normalization steps to result in a probability distribution.
This iterative process involves the following steps: 1. Identification and reduction of factors containing observed variables. 2. Establishment of an elimination order. 3. Using the factors: For each variable in the order we picked: -If we know its value, we update the factors accordingly. -If not, we combine relevant factors, simplify them, and calculate the probabilities.

3 Implementation

3.0.1 Reduction

Reduction is crucial for taking observed values in to the calculations. To do this, we made a function that has two parameters: 'factors' and 'observed_values'. 'factors' is a dictionary that holds the factor number and containing variables as keys and pandas.dataFrame as value. 'observed_values' is a dictionary holding name of the variable which was observed as key and the observation as value. We loop trough all keys of factors and check if the factor variables contain one of the variables from observed values. If it does we use dataFrame.drop() to remove the probabilities that doesn't contain the observation.

```
def reduce_factors(factors, observed_values):
    # Loop through all factors
    for key in factors:
        # Check through all observed variables if they are in factor
        for variable, value in observed_values.items():
            if variable in key[1]:
                # Take current factor we are checking
                current_factor_df = factors[key]
                # Check in the factor where we need to change observed
                                     variable value
```

```python
                mismatched_indices = current_factor_df.index[current_factor_df[variable] != value]
                # Delete values from factor that don't match the observation
                current_factor_df.drop(mismatched_indices, inplace=True)
```

3.0.2 Marginalization

Marginalization function takes 2 variables: 'variable' and 'factor_for_sum'. 'variable' is the variable we are about to sum out of the factor. 'factor_for_sum' is the factor for which we will carry out all of the actions. For marginalization we first check which values we want to keep after marginalization will be done. Then we use dataFrame.groupby() to put together rows where the values are the same. But if we just do groupby() then we would end up with 2 probabilities, one for each of the previous rows. So then we also use dataFrame['prob'].sum() to add these two probabilities together.

```python
def marginalize(variable, factor_for_sum):
    """
    Marginalize the factor based on the variable
    """
    vars_to_keep = [col for col in factor_for_sum.columns if col != variable and col != 'prob']
    if len(vars_to_keep) > 0:
        factor_for_sum = factor_for_sum.groupby(vars_to_keep)['prob'].sum().reset_index()
    return vars_to_keep, factor_for_sum
```

3.0.3 Multiplication

Multiplication function has 2 input parameters: 'first_factor', 'second_factor'. 'first_factor' and 'second_factor' are pandas.dataFrame objects that will be multiplied together. Before the multiplication, we first define common variables to ensure that the next steps work properly. We do this by creating a set from both factor variables, which we then turn to list. To do the multiplication we use pandas.merge() which combines two dataFrames based on their values. Again this will lead us to having two 'prob' columns called 'prob_x' and 'prob_y'. So we take these two columns and for each row we multiply them together and place the result in a new column called 'prob'. Lastly, we remove the unnecessary 'prob_x' and 'prob_y' and return the new dataFrame.

```python
def multiply_factor(first_factor, other_factor):
    """
    Multiply two factors based on the variable
    """
    common_vars = list(set(first_factor.columns[:-1]) & set(second_factor.columns[:-1]))
    merged_factors = pd.merge(first_factor, second_factor, on=common_vars)
    merged_factors['prob'] = merged_factors['prob_x'] * merged_factors['prob_y']
    merged_factors.drop(columns=['prob_x', 'prob_y'], inplace=True)
    return merged_factors
```

3.1 Variable Elimination

The main loop of variable elimination is defined in the run function. It takes 3 inputs: 'query', 'observed', 'elim_order'. 'query' is the a string holding name of the variable for which we are trying to find probability distribution for. 'observed' is a dictionary with keys being the name of the variable and values being the observation of this variable.

And lastly 'elim_order' holds a list with elements arranged in the elimination order. 'elim_order' does no't hold query variable.
To implement variable eliminations, we loop trough all the variables in elimination order. The next steps we do until we have processed every variable in the order:

1. We loop trough all the factors and collect the factors that contain the current variable in a list called 'factors_containing_variable'.
2. If in this list we have more than 1 factor, it means that we have to do factor multiplication for all of the factors in the list, which we can do with a modification of our 'multiply_factor' function called 'multiply_all_variable_factors'. This will result us in one factor.
3. After we have only one factor, or if we initially had only one factor containing this variable, we now can marginalize this factor using our 'marginalize' function.
4. After the marginalization is done, we delete all the factors used to get the resulting factor from the variable elimination factor list
5. We add our now resulting factor to the variable elimination factor list.

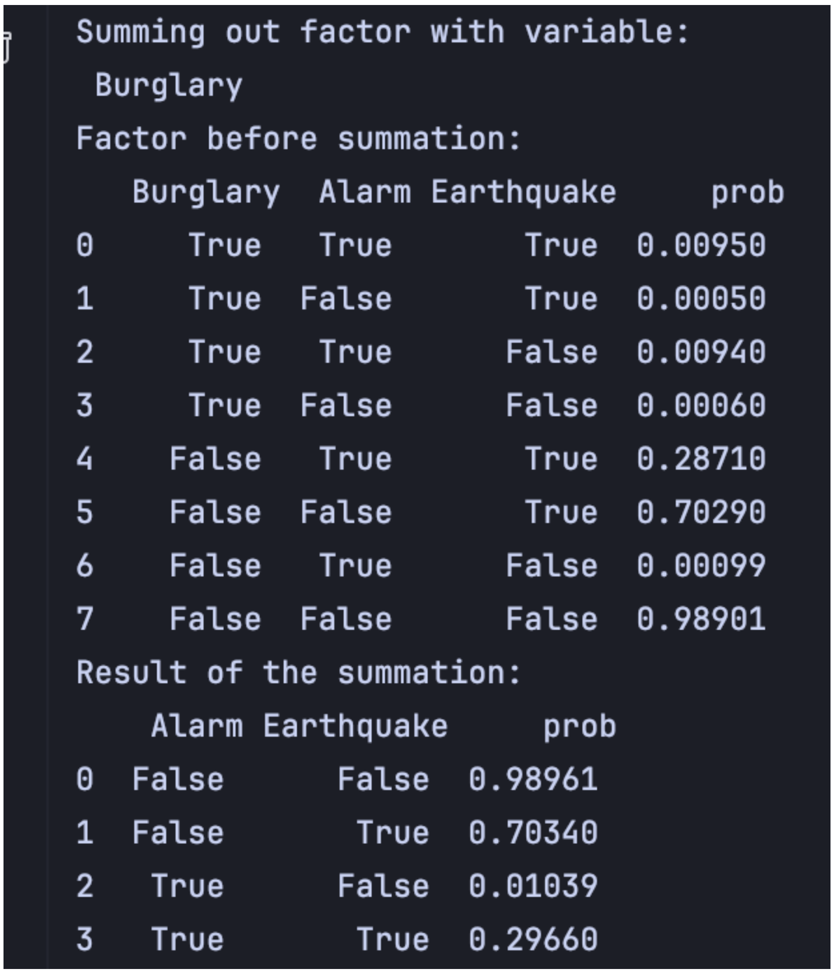
```
    def run(self, query, observed, elim_order):
        var = 0
        i = len(self.factors)
        self.initialize_factors(observed)
        # Loop through all variables in the elimination order
        for variable in elim_order:
            contains = 0
            factors_containing_variable = []
            # Check which factors contain variable
            for factor in self.factors:
                if variable in factor[1]:
                    contains += 1
                    factors_containing_variable.append(factor)
            # If there are multiple factors containing the variable, multiply them
            if contains > 1:
                variables, factor_for_sum = self.multiply_all_variable_factors(variable, factors_containing_variable)

                for i in range(0, len(factors_containing_variable)):
                    self.factors.pop(factors_containing_variable[i])
            # If there is only one factor containing the variable, sum out the variable
            else:
                factor_for_sum = self.factors[factors_containing_variable[0]]
                self.factors.pop(factors_containing_variable[0])

            # Sum out the variable
            variables, summed_out_fac = marginalize(variable, factor_for_sum)

            self.factors[i, tuple(variables)] = summed_out_fac
            i += 1
```

After the loop described above is done, we will be left with factors containing only query variable. Same as before, if there are multiple factors, we first multiply them using 'multiply_all_variable_factors'. When we have only one factor left, again, we use 'marginalize' function on this factor. We will be left up with query variable factor which has only true and false values and the probabilities for them. The result gotten from this might look like the end distribution, but we have one more step left to do, which is normalizing the factor. We can do this by summing up the end probabilities from the factor. And then dividing each of these probabilities with the sum we got. This process will ensure that the end probabilities sum up to 1. Now the Variable Elimination is done.

```
        lastindex = i
        # If there are multiple factors containing the query variable,
                                      multiply them
        if len(self.factors) > 1:
            factors_containing_variable = []
            for fact in self.factors:
                factors_containing_variable.append(fact)

            result = self.multiply_all_variable_factors(query,
                                      factors_containing_variable)

            for i in range(0, len(factors_containing_variable)):
                self.factors.pop(factors_containing_variable[i])

            factor = result[1]
            self.factors[lastindex, tuple([query, ])] = result
        # If there is only one factor containing the query variable, then
                                      continue
        else:
            factor = summed_out_fac

        # Normalize the factor
        prob_sum = factor['prob'].sum()
        factor['prob'] = factor['prob'] / prob_sum
        print("Resulting probability distribution after variable
                                      elimination: \n", factor)
```

4 Results

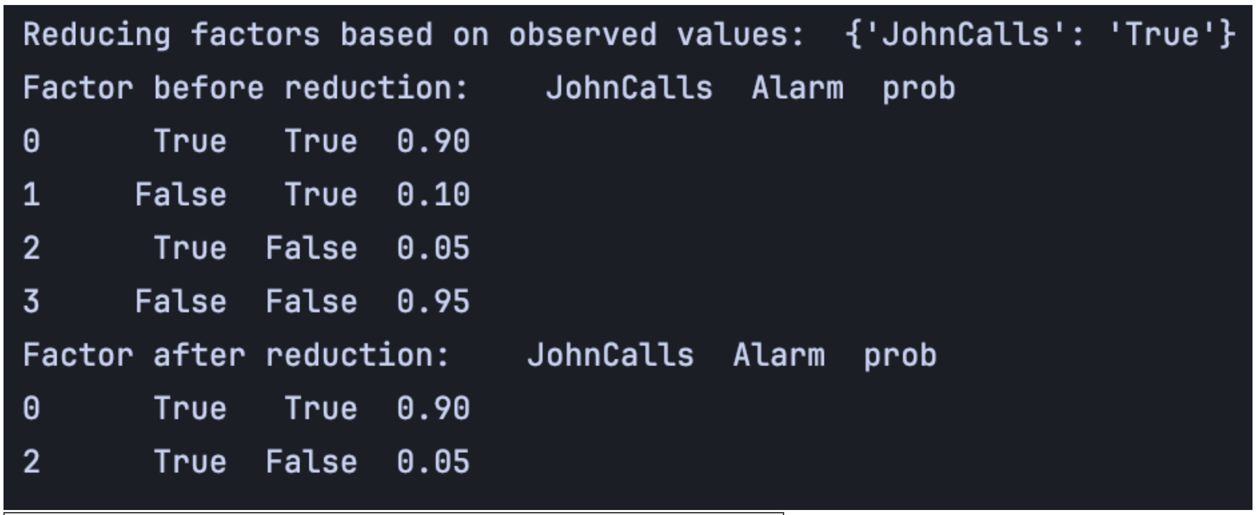
4.1 Marginalization



```
Summing out factor with variable:
 Burglary
Factor before summation:
   Burglary  Alarm Earthquake     prob
0      True   True       True  0.00950
1      True  False       True  0.00050
2      True   True      False  0.00940
3      True  False      False  0.00060
4     False   True       True  0.28710
5     False  False       True  0.70290
6     False   True      False  0.00099
7     False  False      False  0.98901
Result of the summation:
    Alarm Earthquake     prob
0  False      False  0.98961
1  False       True  0.70340
2   True      False  0.01039
3   True       True  0.29660
```

Marginalization		
Alarm	Earthquake	prob
False	False	0.98901 + 0.0006 = 0.98961
False	True	0.70290 + 0.0005 = 0.70340
True	False	0.00094+ 0.00099 = 0.01039
True	True	0.28710 + 0.00950 = 0.29660

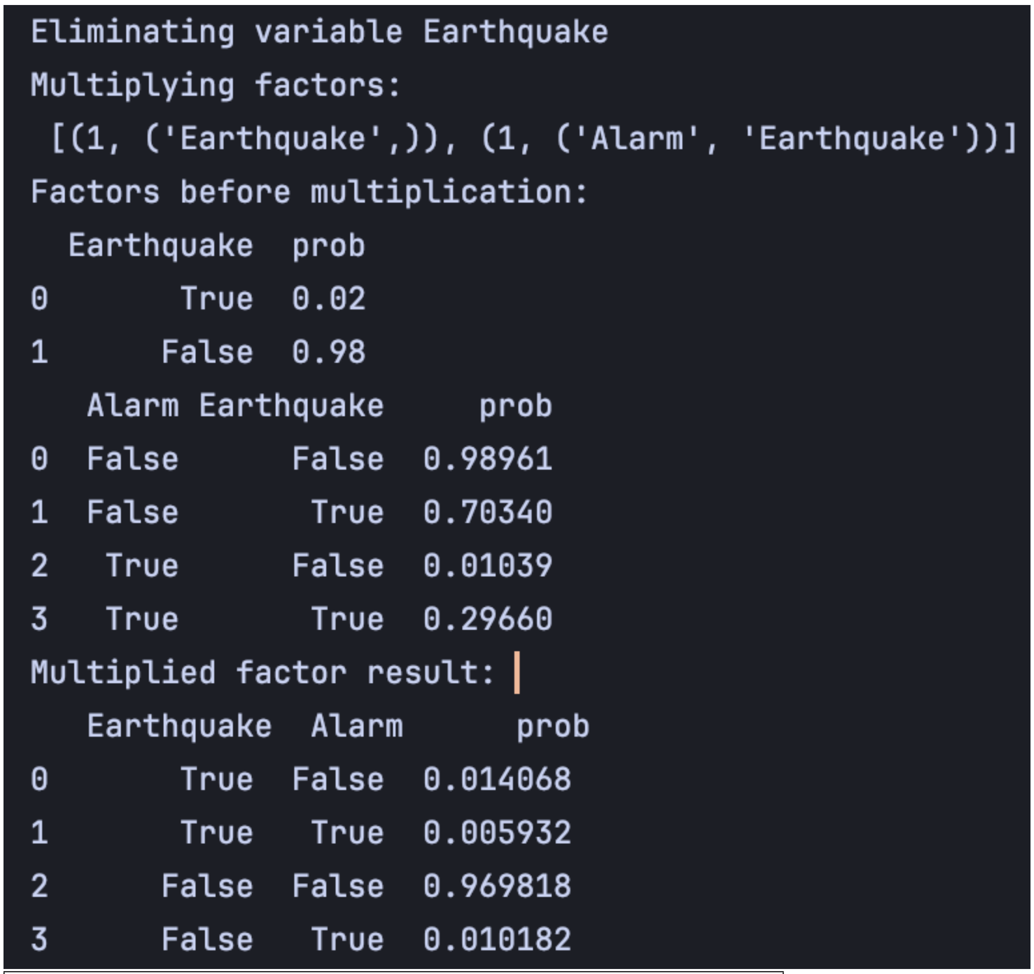
4.2 Reduction



```
Reducing factors based on observed values:  {'JohnCalls': 'True'}
Factor before reduction:    JohnCalls  Alarm  prob
0      True   True  0.90
1     False   True  0.10
2      True  False  0.05
3     False  False  0.95
Factor after reduction:    JohnCalls  Alarm  prob
0      True   True  0.90
2      True  False  0.05
```

Reduction		
JohnCalls	Alarm	prob
True	True	0.90
~~False~~	~~True~~	~~0.10~~
True	False	0.05
~~False~~	~~False~~	~~0.95~~

4.3 Multiplication



```
Eliminating variable Earthquake
Multiplying factors:
 [(1, ('Earthquake',)), (1, ('Alarm', 'Earthquake'))]
Factors before multiplication:
  Earthquake  prob
0       True  0.02
1      False  0.98
   Alarm Earthquake     prob
0  False      False  0.98961
1  False       True  0.70340
2   True      False  0.01039
3   True       True  0.29660
Multiplied factor result:
   Earthquake  Alarm      prob
0        True  False  0.014068
1        True   True  0.005932
2       False  False  0.969818
3       False   True  0.010182
```

Marginalization

Earthquake	Alarm	prob
True	False	0.01039 * 0.02 = 0.014068
True	True	0.29660 * 0.02 = 0.005932
False	False	0.98961 * 0.98 = 0.969818
False	True	0.70340 * 0.98 = 0.010182

5 Conclusion

In the above-shown screenshots and tables, it is shown that our factor operations work. In the screenshot, our program calculates the certain factor operations on the variable, and in the table, we calculate it by hand. As visible the answers match and so we know that the program works in terms of doing the factor operations.
Our implementation Variable Elimination algorithm can work on vectors that have both binary and non-binary variables. It seems to work for small and medium Bayesian networks.

6 Discussion

When trying to apply variable elimination to larger networks with more nodes, arcs and parameters at some point algorithm will have to do factor multiplication with factors that might have thousand or sometimes even million rows. Different elimination orderings might improve the algorithm so that it can carry out such tasks. Intuition for solving this problem would be to try to reduce the amount of columns in the factors used for multiplication. Because the more columns the factor has the more combinations of all variable values there are. Another thing to aim for, if trying to make the algorithm calculate larger networks, is to try to reduce the amount of factors that have to be multiplied at the same time. Maybe there is a smarter way to do it if you first multiply the most "similar" factors, this would be factors with the same variables in them, and only after that multiply the "dissimilar" ones.

7 References

- Kwisthout, J. (2023). Bayesian Networks 1-4 [Slide show; Radboud Universiteit Nijmegen]. https://brightspace.ru.nl/d2l/le/content/431290/viewContent/2421542/View