AI PRINCIPLES & TECHNIQUES

Assignment 3: Variable Elimination

Andrew Schroeder *s1111686*
Radboud University *11-21-2023*

1 Introduction

The importance of Bayesian inferencing in the context of Artifical Intelligence cannot be overstated. Inferencing is a crucial step in the long, complicated, but exciting step towards models that can reason across a wide variety of domains. A crucial aspect of inferencing is the computation of posterior probabilities, that is, given a query variable and a set of observations (evidence), what is the probability distribution of the query variable given the observed evidence? While there are many algorithms capable of performing this computation, perhaps there is none so fundamental or well-known as the classic variable elimination algorithm. In this report the variable elimination algorithm will be implemented along with several well known elimination order heuristics using Python and the powerful Pandas library. The correct implementation of the algorithm is then tested on the alarm and earthquake models. Additionally, a detailed and verbose log file is generated for each network query which shows the progression of the algorithm on a particular query. The testing results demonstrate the algorithm works as intended. While both heuristics work well, the difference between the two heuristics is difficult to see on the small networks tested and larger networks should be tested in the future.

2 Specification

The ability to reason with probabilities and make inferences about the world is one of the core areas of AI under active research. Specifically, the Bayesian Networks proposed by Judea Pearl provide a succinct way to represent independence relationships between the set of random variables that compose the network. The core advantage of utilizing a Bayesian Network as opposed to representing the full joint probability distribution is that the Bayesian Network can represent the full distribution in a factored form - foregoing the need to build large conditional probability tables (CPTs) that cannot be represented in memory. Using factors generated from CPTs provides the opportunity to perform inference on the variables in the network. The goal of this assignment was to develop a thorough understanding of performing inference in Bayesian Networks. While there are many different inferencing algorithms that can perform inference on a Bayesian Network the focus in this paper is on the variable elimination algorithm. The variable elimination algorithm will be implemented and a log file generated which keeps a detailed record of the steps performed by the algorithm on a particular query.

3 Design

3.1 Variable Elimination Algorithm

The pseudocode for the variable elimination algorithm is presented in algorithm 1 (Darwiche et al., 6 Inference by Variable Elimination)

Algorithm 1 Variable Elimination

```
Input:
N: Bayesian network
Q: Query variables in network N
e: Instantiation of some variables in network N
$\pi$: Elimination order for network variables not in Q
Output: the joint marginal Pr(Q, e)
1: procedure VARIABLE-ELIMINATION(N, Q, e, $\pi$)
2:     $S \leftarrow \{f^e : f$ is a CPT of network N$\}$
3:     for $i = 1$ to length of order $\pi$ do
4:         $f \leftarrow \prod_k f_k$, where $f_k$ belongs to $S$ and mentions variable $\pi(i)$
5:         $f_i \leftarrow \sum_{\pi(i)} f$
6:         replace all factors $f_k$ in S by factor $f_i$
7:     return $\prod_{f \in S} f$
```

In words the algorithm can be described by the following steps:

1. Create a set of **factors** from the Bayesian Network CPTs
2. **Reduce** the factors by applying the evidence to remove inconsistent rows of data from the factors. The reduced factors are represented by f^e in line 2.
3. Eliminate each hidden variable in the network according to some variable ordering π (line 3). Repeat the following to eliminate each variable that is not a query variable:
   (a) First **multiply** all factors that contain the hidden variable to be eliminated $\pi(i)$ and store the result in f (line 4)
   (b) Second **sum out** the hidden variable $\pi(i)$ from f and call this new factor f_i (line 5)
   (c) Finally, update the list of factors S by removing all factors f_k that were multiplied together in line 4 and adding the new factor f_i (line 6).
4. After all variables have been eliminated, multiply the remaining factors together and return the result (line 7).

Algorithm 1 returns the joint marginal Pr(Q, e) - the joint probability across the query variables given evidence e. The posterior probability Pr(Q|e) can be computed according to the conditional probability rule:

$$Pr(Q|e) = \frac{P(Q,e)}{P(e)}$$

Thus the output of Algorithm 1 must be normalized by the evidence. Another way of saying this is that because factors do not guarantee any particular probability distribution, the final output returned on line 7 may have a probability distribution that does not sum to one. To fix this, the returned factor must be normalized by adding up all the probabilities in the factor to obtain the total sum and then dividing the probability of each row in the factor by the total sum.

Note that algorithm 1 expects to be provided with the variable elimination order π - obtaining an optimal elimination order is an NP-hard problem and thus is not gernally solveable. However several well known heuristics can be utilized for calculating this order. These heuristics will be discussed in more depth in the testing and discussion sections of the report.

3.2 Software Architecture

The provided Python template files were used as the starting point for building the variable elimination algorithm and associated software. Specifically *read_bayesnet.py* contained the necessary functions to load the Bayesian network Variable Elimination as a Pandas dataframe along with useful attributes including the *values* dictionary which contained the allowed values per variable, the *probabilities* dictionary which contained the CPTs, and the *parents* dictionary which mapped each variable to a list of it's parents. The first step in building the variable elimination algorithm after loading the network was to build the data structure that would represent factors and allow one to perform the three necessary factor operations: reduction, multiplication, and marginalization. The backbone of the Factor class is a CPT that is stored in a Pandas dataframe. Pandas dataframes were chosen because they have very convenient and powerful built-in methods that provide the necessary functionality for the three factor operations. The *dataframe.groupby()* and *pd.merge()* operations were particularly useful for factor marginalization and multiplication, respectively. Finally, the *logger.py* file contains two logger handlers for recording the algorithm progress. The first logs a concise record of the input and the output of the function to the console, while the second logs a verbose and detailed record of each stage of the algorithm for analysis.

4 Implementation

For the sake of brevity and concisesness, the implementation of the variable elimination algorithm will not be presented here as it closely aligns with the provided pseudocode and there is very little of interest left to be described. However, the implementation of factors and specifically the marginalize and multiplication operations is of great interest. There are many ways to implement these methods but for the sake of simplicity the Pandas library was used. The implementation of the marginalize function is provided in listing 1 below.

```
1 def marginalize(self, variable):
2     all_columns = self.dataframe.columns[:-1]
3     columns_to_keep = [col for col in all_columns if col != variable]
4     new_factor = self.dataframe.groupby(columns_to_keep).sum().reset_index()
5     new_factor.drop(variable, axis=1, inplace=True)
6     self.dataframe = new_factor
```

Listing 1: Marginalize

The algorithm begins by extracting all the column names which correspond to the variables in the factor. Only those variables in the table that are not being eliminated are kept however. The most important aspect of the function is on line 5 where the `groubpy` method is called with the `columns_to_keep` as the function argument. This operation combines rows of the table where the variables in `columns_to_keep` have the same instantiation, while ignorning the value of the variable to be eliminated. After grouping the appropriate rows, the sum operation adds the probabilities within each group. Because the `groupby` operation merges rows of the dataframe together the indices or no longer correct so `reset_index` is called. Finally, the column with the variable to be eliminated is removed from the table before returning the marginalizeed factor.

The multiplication of factors, while greatly simplified by using the Pandas library, is nonetheless more complicated than the marginalization process, as seen in listing 2.

```
 1 def multiply(self, factor2: "Factor", variable):
 2     f1_df = self.get_data_frame()
 3     f2_df = factor2.get_data_frame()
 4     all_columns = f1_df.columns[:-1].tolist() + f2_df.columns[:-1].tolist()
 5     all_columns = list(set(all_columns))
 6     all_columns = all_columns + ["prob"]
 7     new_df = pd.merge(f1_df, f2_df, on=variable, suffixes=("_1", "_2"))
 8     new_df["prob"] = new_df["prob_1"] * new_df["prob_2"]
 9     new_df = new_df[all_columns]
10     self.dataframe = new_df
```

Listing 2: Multiply

Lines 2-6 of the procedure are responsible for creating the set of column names that are included in the new factor which is the union of the column names from both of the individual dataframes that are multiplied together. The critical operation that actually merges the two factors into one is performed on line 7 where the `pd.merge` operation is invoked on the two dataframes. When merging the two tables, the function must be told which column to merge on and in this case the column to merge on is the `variable` argument which is the variable that will be eliminated in the steps following the multiplication process. Because each dataframe will have its own probability column, the suffixes argument is specified to append _1 and _2 to these columns to distinguish them in the merged table. Line 8 is then responsible for multiplying the probabilities stored in `prob_1` and `prob_2` into a new column `prob`. Finally the necessary columns are extracted.

5 Testing

To test the implementation of the variable elimination algorithm, the VE_pgmpy.ipynb Google Collab file written by Mika Van Emmerloot was utlizied. Specifically, the outputs from the *Example 1 Earthquake* and *Example 2 Alarm* were used to verify the correct working of the implemented variable elimination algorithm. The exact queries that were executed in these two examples are presented in table 1.

Query	**Network**	**Query Variable**	**Evidence**
1	Alarm.bif	Alarm	{Burglary: True}
2	Earthquake.bif	Tampering	{Smoke: True, Report: True}

Table 1: Query 1 Results

Two heuristic functions for finding the elimination order were utilized: *fewest incoming arcs first* and *contained in fewest factors first*. While the heuristics would have an effect on the size of the tables generated and the overall efficiency of the algorithm, both heuristics should produce the same answer. Additionally, to gain insight into the progression of the algorithm and the intermediate steps it takes to compute the result, a detailed log file was generated. See appendix A for an example of a log file for the alarm network query.

6 Results

The results of performing the two queries mentioned in the Testing section are provided below. Note that both heuristics produced identical outputs to each query.

Alarm	**P(A | B = True)**
False	0.0598
True	0.9402

Table 2: Query 1 Results

Tampering	**P(T | S = True, R = True)**
False	0.971564
True	0.028436

Table 3: Query 2 Results

7 Discussion

The results of the experiments demonstrate that the implemented variable elimination algorithm correctly computes posterior probabilities of a query variable given the evidence. Specifically, it was shown that two heuristics used to obtain the elimination order (least-incoming-arcs-first, contained-in-fewest-factors-first) both lead to the same outcome which is expected. The only difference between the two heuristics is the size of the generated intermediate factors that are generated. Finding the optimal elimination order is an NP-hard problem but there are several well known heuristics that can be used to generate relatively performant orders. The importance of the elimination order cannot be overstated - the optimal elimination order is directly related to the treewidth of the underlying network and the time complexity of variable elimination is exponential in the treewidth. For small networks the difference in elimination order may not be noticeable but for large networks it can be the difference between computing a problem instance within seconds and the computer crashing due to insufficient memory.

8 Conclusion

In this report the notion of bayesian inference in the context of the variable elimination algorithm was investigated. Specifically, the variable elimination algorithm was implemented in Python using the Pandas library to construct the necessary factors required to run the algorithm. The least-incoming-arcs-first and contained-in-fewest-factors-first heuristics were built to generate relatively good elimination orders. The algorithm was then tested by using several well known small bayesian networks including the alarm and earthquake networks. The final probabilities generated for each query demonstrate the implementation is correct - additionally the detailed log file provides a verbose record of exactly what operations are being performed by the algorithm at each step which is particularly useful for debugging. While bayesian inference in general is NP-hard, the variable elimination algorithm has proven to be a reliable and trusted method that can be used to solve reasonably sized inferencing tasks.

9 Reflection

Overall the assignment was well strutured and proceeded smoothly. The critical realization for performing this task well was the use of the Pandas library when constructing the factors class. Additionally the Python logging utility was immensely helpful for generating the output log.

10 Bonus

10.1 Non-Binary Variables

While not explicitly tested, the algorithm is capable of performing inference on networks with categorical variables whose domains are greater than two (non-binary). This is because the marginalization and multiplication operations in the factor class do not rely on the variables having a particular domain and thus are immune to this parameter. The `groupby()` and `merge()` operations work on all variable domains, regardless of the size of the domains.

10.2 Elimination Order Heuristics

The two example elimination orders that were provided in the assignment description were implemented (fewest-incoming-arcs-first, and contained-in-fewest-factors-first). The built-in console interface allows the user to select which heuristic to use and the generated elimination order can be seen in the log output file.

11 Bibliography

1. Darwiche, A. (n.d.). 6 Inference by variable elimination. In Modeling and reasoning with Bayesian Networks (pp. 141–141). essay, Cambridge University Press.

12 Appendix A - Example Log File Output

```


------- NEW VE RUN STARTED -------

Date: 2023-12-19 12:45:16.162123

Elimination order heuristic used: Least incoming arcs first

Elimination Order: ['Fire', 'Leaving', 'Alarm']

Query Variables: Tampering

Observed Variables: {'Smoke': '1', 'Report': '1'}

Original Factors:

    Alarm Tampering Fire    prob
  0     0         0    0  0.9999
  1     1         0    0  0.0001
  2     0         0    1  0.0100
  3     1         0    1  0.9900
  4     0         1    0  0.1500
  5     1         1    0  0.8500
  6     0         1    1  0.5000
  7     1         1    1  0.5000

    Fire  prob
  0    0  0.99
  1    1  0.01

    Leaving Alarm   prob
  0       0     0  0.999
  1       1     0  0.001
  2       0     1  0.120
  3       1     1  0.880

    Report Leaving  prob
  0      0       0  0.99
  1      1       0  0.01
  2      0       1  0.25
  3      1       1  0.75

    Smoke Fire  prob
  0     0    0  0.99
  1     1    0  0.01
  2     0    1  0.10
  3     1    1  0.90

    Tampering  prob
  0         0  0.98
  1         1  0.02

Factors with evidence applied:

   Alarm Tampering Fire    prob
0      0         0    0  0.9999
1      1         0    0  0.0001
2      0         0    1  0.0100
3      1         0    1  0.9900
4      0         1    0  0.1500
5      1         1    0  0.8500
6      0         1    1  0.5000
7      1         1    1  0.5000

   Fire  prob
0     0  0.99
1     1  0.01

   Leaving Alarm   prob
0        0     0  0.999
1        1     0  0.001
2        0     1  0.120
3        1     1  0.880

   Leaving  prob
1        0  0.01
3        1  0.75

   Fire  prob
1     0  0.01
3     1  0.90

   Tampering  prob
0          0  0.98
1          1  0.02

Eliminate variable: Fire

Factors to multiply containing variable: Fire

   Alarm Tampering Fire    prob
0      0         0    0  0.9999
1      1         0    0  0.0001
2      0         0    1  0.0100
3      1         0    1  0.9900
4      0         1    0  0.1500
5      1         1    0  0.8500
6      0         1    1  0.5000
7      1         1    1  0.5000

   Fire  prob
0     0  0.99
1     1  0.01

   Fire  prob
1     0  0.01
3     1  0.90

Result of multiplication:

   Tampering Fire Alarm          prob
```

```
  0         0    0     0  9.899010e-03
  1         0    0     1  9.900000e-07
  2         1    0     0  1.485000e-03
  3         1    0     1  8.415000e-03
  4         0    1     0  9.000000e-05
  5         0    1     1  8.910000e-03
  6         1    1     0  4.500000e-03
  7         1    1     1  4.500000e-03

Sum out Fire

    Tampering Alarm      prob
  0         0     0  0.009989
  1         0     1  0.008911
  2         1     0  0.005985
  3         1     1  0.012915

New List of Factors:

    Leaving Alarm   prob
  0       0     0  0.999
  1       1     0  0.001
  2       0     1  0.120
  3       1     1  0.880

    Leaving  prob
  1       0  0.01
  3       1  0.75

    Tampering  prob
  0         0  0.98
  1         1  0.02

    Tampering Alarm      prob
  0         0     0  0.009989
  1         0     1  0.008911
  2         1     0  0.005985
  3         1     1  0.012915

Eliminate variable: Leaving

Factors to multiply containing variable: Leaving

    Leaving Alarm   prob
  0       0     0  0.999
  1       1     0  0.001
  2       0     1  0.120
  3       1     1  0.880

    Leaving  prob
  1       0  0.01
  3       1  0.75

Result of multiplication:

    Leaving Alarm     prob
  0       0     0  0.00999
  1       0     1  0.00120
  2       1     0  0.00075
  3       1     1  0.66000

Sum out Leaving

    Alarm     prob
  0     0  0.01074
  1     1  0.66120

New List of Factors:

    Tampering  prob
  0         0  0.98
  1         1  0.02

    Tampering Alarm      prob
  0         0     0  0.009989
  1         0     1  0.008911
  2         1     0  0.005985
  3         1     1  0.012915

    Alarm     prob
  0     0  0.01074
  1     1  0.66120

Eliminate variable: Alarm

Factors to multiply containing variable: Alarm

    Tampering Alarm      prob
  0         0     0  0.009989
  1         0     1  0.008911
  2         1     0  0.005985
  3         1     1  0.012915

    Alarm     prob
  0     0  0.01074
  1     1  0.66120

Result of multiplication:

    Tampering Alarm      prob
  0         0     0  0.000107
  1         1     0  0.000064
  2         0     1  0.005892
  3         1     1  0.008539

Sum out Alarm

    Tampering      prob
  0         0  0.005999
  1         1  0.008604

New List of Factors:

    Tampering  prob
  0         0  0.98
  1         1  0.02

    Tampering      prob
  0         0  0.005999
  1         1  0.008604

Elimination step complete. Multiply the following remaining factors
    together.

    Tampering  prob
  0         0  0.98
  1         1  0.02

    Tampering      prob
  0         0  0.005999
  1         1  0.008604

Final factor pre-normalization.

    Tampering      prob
  0         0  0.005879
  1         1  0.000172

Final factor post-normalization.

    Tampering      prob
  0         0  0.971564
  1         1  0.028436
```

Listing 3: Example Run