AI PRINCIPLES & TECHNIQUES

Assignment 3: Variable elimination

SUE-ELLEN BERNADINA *s1035357*
NATALIA KARDAMI *s1034194*
Radboud University *February 2023*

In the field of Artificial Intelligence, the ability to use mathematics to make predictions is a fundamental technique. A popular approach to model data is using Bayesian Statistics. Bayesian models are an integral method in Machine Learning that has proven to be very accurate in spam detection, medical predictions, image processing, information retrieval and more (1). However, deriving information from complex networks requires intensive computational resources, which is why a technique called Variable Elimination is used as a helper tool to make exact inferences from data. In this project, we implement the Variable Elimination algorithm and assess its performance across different heuristics, and test it on two different belief networks.

Contents

1 Approach

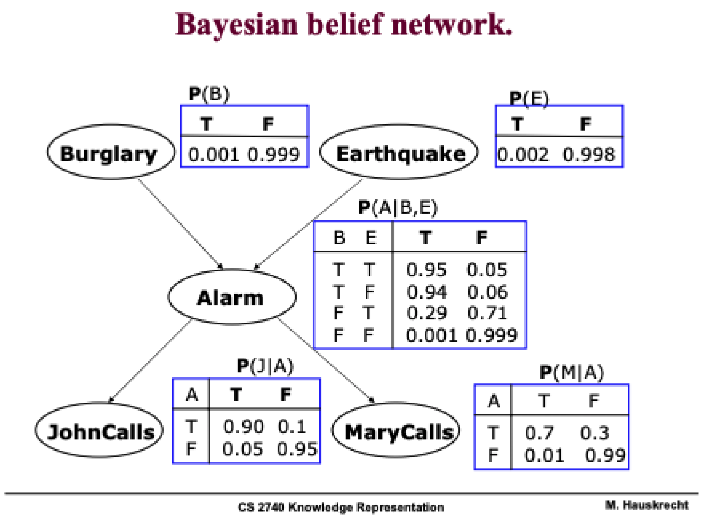
1.1 Project Description

In this project, we attempt to implement the Variable Elimination algorithm to simplify Bayesian Networks and make inferences. The purpose of this project is to understand the Variable Elimination (VE) algorithm, implement it and evaluate different heuristics. In this report, we set the stage by defining Bayesian Networks and the applications of exact inference, then we define the concept of factors and their use in the VE process. Then, we explain the Variable Elimination algorithm and our implementation and lastly, we discuss our results and findings. Visit our github repository to see all relevant files and the development process.

1.2 Background Knowledge

1.2.1 Bayesian Networks

Modelling real-life data is far from novel, knowing that in the environment, there are multiple variables and relationships between them. Actual measurements are complex to interpret since influences between parameters are invariably present. One way to incorporate these relations between variables is to use structures called Bayesian Networks. Bayesian (or Belief) Networks (BNs). BNs are able to model such complex relations in data because they integrate dependencies between variables in prediction-making.
In order to make predictions, we must first build the model for our data by defining its variables and representing their dependence relations. Such networks are usually constructed a) manually, with the help of domain experts, or b) automatically, by extracting information from databases (Horný, 2014). Knowing this, we can formally define a Bayesian Network.
A Bayesian Network is a directed, acyclic model, containing variable nodes, their domains, and a set of conditional probability distributions describing the causal relations between the nodes. Usually, they are represented by a graphical model and the accompanying probabilities in the form of tables.



P(B)

T	F
0.001	0.999

P(E)

T	F
0.002	0.998

P(A|B,E)

B	E	T	F
T	T	0.95	0.05
T	F	0.94	0.06
F	T	0.29	0.71
F	F	0.001	0.999

P(J|A)

A	T	F
T	0.90	0.1
F	0.05	0.95

P(M|A)

A	T	F
T	0.7	0.3
F	0.01	0.99

Figure 1: "Burglary" Bayesian graphical network and probability tables

Figure 1 illustrates an example of such a network. We have 5 nodes, Burglary, Earthquake, Alarm, JohnCalls and MaryCalls. Each of these nodes signifies a binary variable that can be True or False. Note that BNs are not restricted to binary variables. The directed arrows signify a dependency relationship. We formally define the Burglary network as

- A set of Variables:
  $V = \{B, E, A, J, M\}$
- A set of Domains:
  $D_v = \{\text{True,False}\}$
- A set of Probability Distributions:
  $P_v = \{P(B), P(E), P(A \mid B, E), P(J \mid A), P(M \mid A)\}$

One of the most important properties of Bayesian Networks is the Markov Blanket assumption. This means that we crudely assume that a node is conditionally dependent only on its direct parent nodes. In the Burglary network example, we define the probability distribution (p.d.) of 'Alarm' as such: $P(\text{Alarm} \mid \text{Burglary, Earthquake}$ and the p.d. Of 'JohnCalls' as: $P(\text{JohnCalls} \mid \text{Alarm})$. We define the relations for the rest of the nodes of the network likewise.

1.2.2 Exact Inference in Bayesian Networks

So how can we use BNs to make probabilistic predictions? Sometimes we observe some variable(s) (the evidence) and record its value and we care to know what is the probability of another value of a variable (the query) being observed. This process is called exact inference. We want to accurately calculate, not approximate, the true probability of a query given some evidence, in a set of variables X. We can calculate this probability using Bayes' Theorem (Poole & Mackworth, 2017).

$$P(Q = q \mid E = e) = \frac{P(Q = q, E = e)}{P(E = e)}, \tag{1}$$

where E stands for the evidence variable(s) and Q for the query and e and q likewise denote the values of interest.

However, calculating the joint probability $P(Q, E = e)$ is not as trivial as it seems. Let H denote the set of all the non-evidence and query variables and $X = \{Q, E = e, H\}$ denote the set of all n variables in the network. We call the set of variables H "hidden" because they do not directly appear in the query, but exist in the network and can influence the query and evidence. The formula for calculating this joint probability is:

$$\prod_{i=1}^{n} P(X_i = v_i \mid X_1 = v_1, ..., X_{n-1} = v_{n-1}) \tag{2}$$

We observe that regardless of the query and evidence, all the other hidden variables in the problem are involved. We can see that this expression quickly becomes very complicated and computationally expensive, increasing along with the size of a network. In order to simplify this expression, we can use the **Markov Blanket assumption** (see 1.2.1) to signify that each node is conditionally dependent on its parents. Therefore, the joint probability becomes:

$$\prod_{i=1}^{n} P(X_i \mid Parents(X_i)) \tag{3}$$

The reduction of this product signifies the essence of Variable Elimination, as a Bayesian Network defines this joint probability as the "**factorization**" of the conditional probability distributions of its nodes (Poole & Mackworth, 2017). The term Factor becomes increasingly significant in the explanation and implementation of the VE algorithm, which will become more apparent in the following sections.

We will now define this reduction in the domain of the Burglary network. We can derive a global, all-inclusive joint probability for the network by combining equations (2) and (3):

$$P = P(B) \times P(E) \times P(A \mid B, E) \times P(J \mid A) \times P(M \mid A) \tag{4}$$

Now, let's make an example inquiry. We want to know the probability of Mary calling if we observe that there was no alarm. In Bayesian terms, this posterior probability is defined as

$$P(M = t \mid A = f) = \frac{P(M = t, A = f)}{P(A)} \tag{5}$$

For reasons that will become apparent in the later sections, we categorize the variables as:

- $Q = \{M\}$ the Query variable
- $E = \{A = f\}$ the set of Evidence variables and their observed values
- $H = \{E, B, J\}$ the set of Hidden variables, which are not directly relevant to our inquiry

To calculate the numerator, we use equation (4) for the specific inquiry. (More details on how to enforce evidence will be explained in section 1.3.2.) This equation is proven to be less computationally intensive and might seem feasible for small, binary networks, however, it is not computationally effective for larger or non-binary valued networks. (Gao, 2021)

1.3 Factors

We saw how a Bayesian Network is defined, and we used its Markov property to simplify the computation of an inquiry (as seen in equation 3). Before we get into the specifics of the VE algorithm, we must introduce the concept of factors. Factor operations will allow us to simplify equation (3) even further and allow us to reduce this complex summation of our inquiry.

1.3.1 Definition

A factor f is an arbitrary mapping of either a conditional or a joint distribution into a table that holds all the possible value combinations of its variables and their corresponding probabilities. A factor does not explicitly hold the nature of dependence between its variables, only their values and probabilities. The Variable Elimination algorithm works by converting the conditional probabilities of a network, transforming them into factors and performing operations on them in order to get to the desired query. First, we will introduce and define these operations and partly apply them to the Burglary network to illustrate them. For this, the initial factors $F_i = \{f_1(B), f_2(E), f_3(A, B, E), f_4(A, J), f_5(A, M)\}$ are defined as shown in the blue tables in Figure 1. In the next section, we will define the steps of the Variable Elimination Algorithm. Each of the following operations serves a vital role in this algorithm. For the following sections, we refer to the *size* of the factor as the number of probabilities it holds, and the *dimension* of the factor as the number of variables involved.

1.3.2 Reduction

To reduce a factor means that we "delete" some of its entries that are not consistent with the evidence. Since we know the value that the evidence variables take, entries that are inconsistent with the evidence value are redundant and can safely be removed. This not only makes computations easier, but it is necessary to reach the desired posterior probability. We apply this to factors that contain at least one of the evidence variables. Applying this to the Burglary Network, we have the inquiry $P(M = t \mid A = f)$. The evidence says that we are certain that the alarm does not go off, therefore its value is A=false. As an example, let's reduce $f_4(A, J)$ by the evidence A = false. This operation is very straightforward. We check the values of A in f_4 and if they do not equal the evidence, we simply delete that row. In this case, we get a reduced factor $f_5(A, J) = \{A = f : \{J = t : 0.05, J = f : 0.95\}\}$ For practical reasons, we will refer to this operation as `REDUCE()`.

1.3.3 Marginalization

The marginalization operation effectively removes a variable from a factor by summing out all its possible values. Generally, the marginalization operation on a factor $f(X = X_n)$ for a variable X_1 is defined as:

$$\sum_{X_1}(X_2, ..., X_n) = (f(X_1 = v_1, ..., X_n) + ... + f(X_1 = v_k, ..., X_n)) \tag{6}$$

This operation is crucial to eliminate a variable from a factor while preserving its information within the remaining variables. In VE this operation is performed to eliminate the hidden variables. Let us apply this once again to the Burglary network. We arbitrarily select $f_4(A, B, E)$ and illustrate how to "Sum out" B. We will sum the probabilities of the possible values of B for each combination of the other variables, A, E. First, we observe the entry $A = t, E = t$. This occurs twice, once for each possible value of B. We want to sum out those two probabilities. We replace these two entries with all 3 variables in them with one singular entry of 2 variables: $\{(A = t, E = t) : \{0.95(B = t) + 0.29(B = f) = \mathbf{1.24}\}\}$. We continue this process for all tuples of A and E and in the end, we effectively have a table that no longer has B in it. The new factor resulting from this would be a 2×2-entry factor $f_6(A, E)$, compared to the original 8-entry factor. The size of a new factor after summing out a variable is reduced by the size of the possible values of the variable being eliminated. We will refer to this operation as `SUM_OUT()`. (Gao, 2021)

1.3.4 Multiplication

Now, we will define the multiplication operation. This is the only operation which takes two factors and produces a larger factor. We use the multiplication operation to "merge" multiple factors into one. In the previous section, we mentioned that we can eliminate a variable from a factor by marginalization. However, variables may appear in multiple factors. This is where factor multiplication comes in. By multiplying all the factors that contain a variable, we yield a single factor that contains all the information on that variable, thus allowing us to marginalize and remove it. The result of the multiplication is a factor containing all the unique variables involved in the two factors. Each entry is the union of the corresponding sets in the two factors. The size of the new factor is the product between the number of possible values for each variable involved. First, we build a table that contains all the variables and their possible values. For each row in the new factor, we find the corresponding entries in the two factors and multiply their probabilities. We will refer to this operation as `MULTIPLY()`.

In the Burglary example, we may want to, for instance, multiply all factors that contain E. For the sake of simplicity, we will multiply the initial factors $(f_2(E), f_3(A, B, E))$ of E from section 1.3.1. We create a new $2 \times 2 \times 2$ factor $f_7(A, B, E) = (f_2(E) \times f_3(A, B, E))$. For each entry in the new factor, we multiply the corresponding probabilities. For example, we want to fill in the new probability for $(A = t, B = t, E = f)$. We take the probability from $f_2(E = f) = 0.998$ and multiply it with the probability from $f_3(A = t, B = t, E = f) = 0.94$. This yields the entry $\{(f_7(A = t, B = t, E = f) =) = 0.998 \cdot 0.94 = \mathbf{0.93812}\}$. Likewise, we can do the same for the remaining 7 rows to get the full factor.

1.3.5 Normalize

Sometimes, after some operations, we might end up with a one-dimensional factor where the variable's probabilities do not sum to 1. This is something that is not desirable, therefore we want to normalize the factor probabilities. Normalization is quite trivial. We divide each probability of a variable by the sum of all its probabilities. We will refer to this operation as `NORM()`.
We illustrate this with a toy example on an arbitrary factor f, with a variable X with 3 possible values: $f(X) = \{positive : 0.55;\ negative : 0.60;\ zero : 0.1\}$. We get the sum of all the probabilities $= 1.25 > 1$. To normalize the factor, we divide every probability by this sum, so the new factor is:

$$f_{norm} = \{positive : 0.44;\ negative : 0.48;\ zero : 0.08\}$$

1.4 Variable Elimination

Now that we have discussed all factor operations, we can define the steps of the Variable Elimination algorithm. We have a Bayesian Network defined by:

- A query variable Q
- A set of i evidence variables E
- A set of n hidden variables

We want to compute equation (1). This requires a complex computation of the joint probability, in its simplified version as seen in equation (3). The variable elimination algorithm transforms that product of probability distributions into a product of factors. Note that every time we perform an operation on a factor we discard it and "create" a new one. Here are the steps:

1. For each variable in the network, construct a factor
2. Find the factors that contain at least one evidence variable. `RESTRICT()` each factor by the value of the evidence variable(s).
3. For each hidden variable $h \ni H$ in a certain elimination ordering *
   - `MULTIPLY()` all the factors that contain the variable h
   - `SUM_OUT()` the variable h from the new multiplied factor
4. `MULTIPLY()` the remainder of the factors.
5. `NORM()` the resulting factor. This yields the joint posterior distribution of the query variable.

2 Methods

In this project, we attempt to implement the Variable Elimination algorithm in Java v19, explore heuristics and evaluate them, and finally test the performance of our algorithm with sample networks "Earthquake" and "Survey" provided in a .bif template. Results were collected in Google Sheets and processed with JupyterLab running Python v3.9. All result analysis files can be found under results/notebook in the project folder.

2.1 Design

The base code for Variable Elimination Algorithm was provided as a template from the course. The template included a fully functional User Interface class and .bif network reader class, along with helper classes such as: `Condition, ObsVar,Variable` and `Main`. The `VarElim` and `Factor` classes were added at our discretion.

- `Variable` represents a node and includes a variable name, possible values, the probability table and the parent nodes,
- `Observed Variable` represents a variable and its corresponding value that is observed
- `Condition` represents an array of observed variables
- `Factor` represents a factor object that an array of variables involved and the corresponding probability table. This class implements all the operations as described in section 1.3
- `VarElim` is the class that handles the variable elimination steps, and it holds a priority queue of all the variables in the network. The elimination ordering is handled dynamically by the priority queue which is sorted according to evaluations from different heuristics.

2.2 Complexity

In the variable elimination algorithm, the order of elimination of the hidden variables matters. Constructing a factor with 8 variables is way exponentially more computationally expensive than constructing a factor with 3. The *elimination width* is the dimension of the largest factor that occurs throughout the entire algorithm. The *tree width* w is the minimum elimination width of any of the orderings of the variables. For a specific ordering π, and an elimination width of k, the complexity of the variable elimination algorithm is $2^{O(k)}$. The best-case complexity is equal to the *tree width* of the network: $2^{O(w)}$, and the worst-case complexity is $2^{O(n)}$, where n is the number of all variables (Ghaderi, 2006). Elimination ordering in variable elimination changes dynamically with each step, as factors are deleted and created, therefore the elimination ordering for each state might be different.

2.3 Heuristics

2.3.1 Least Incoming Arc

Nodes with the smallest number of parents are prioritised. This is a very simple and effective heuristic since nodes with parents are conditionally dependent on them, therefore the resulting factors are much larger. In the Burglary network, the initial ordering would be $\pi = \{B = E, \rightarrow J = M, \rightarrow A\}$

2.3.2 Fewest Factors

This heuristic prioritises nodes that are contained in the smallest number of factors. This way, the elimination starts with factors that produce smaller multiplications, followed by a hidden variable elimination. This can significantly decrease complexity. In the Burglary network, the ordering would be $\pi = \{J = M, \to B = E, \to A\}$

3 Results

We managed to successfully implement the variable elimination algorithm and 3 different heuristics. Our implementation comes with 2 major limitations. It only works with binary networks and a maximum of 1 observed variable. Our design is intended to process non-binary variables and multiple observed variables, however, when trying with these conditions compilation is successful, but the output is null. To see a sample run, check `/test_run.txt` in the project hand-in. We can evaluate our heuristics by means of elimination width. As mentioned in 2.2, the elimination width, thus the complexity of an algorithm run is dependent on the elimination ordering, which is also varied across different queries. This is why we cannot offer a conclusive evaluation of the heuristics at a global level. To combat this, at the start and end of each run we display the elimination width. This way we can compare across heuristics for a single query manually. In the example run, due to the simplicity of the network, is 3 across all heuristics, which is very satisfactory, since it is the best-case complexity for this network. We also evaluated the runtime for each heuristic but there was no significant difference, therefore we omitted that measure.

4 Discussion

Our implementation bears a lot of possible improvements. Multiple observed variables and non-binary variables should be added to this implementation to deem it complete. Moreover, we find complexity evaluation very significant and we believe there is not enough research on heuristics and complexity in inference algorithms. Nevertheless, this project has been an elaborate challenge and learning experience, which has led to an extremely deep understanding of Bayesian networks, exact inference, and the mathematics between complexity reduction.

5 Sources

- Ghaderi, H. (2006). Intro to Artificial Intelligence Reasoning under Uncertainty. CSC384. Retrieved February 22, 2023, from http://www.cs.toronto.edu/
- Hauskrecht, M. (n.d.). Bayesian belief networks - University of Pittsburgh. Retrieved February 28, 2023, from https://people.cs.pitt.edu/ milos/courses/cs2740
- Poole, D. L., & Mackworth, A. K. (2017).
  Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press. (Poole & Mackworth, 2017), Retrieved February 28, 2023, from https://artint.info/2e/html/ArtInt2e.Ch8.S4.SS1
- turing.com. (2022, June 30). An overview of Bayesian networks in Artificial Intelligence.
  Retrieved February 28, 2023, from https://www.turing.com/kb/an-overview-of-bayesian-networks-in-ai