# Bayesian Networks and Variable Elimination

Chiara Benini

February 20, 2025

## Contents

## 1 Introduction

Bayesian networks are a class of probabilistic graphical models that represent random variables and their conditional dependencies via a directed acyclic graph (DAG). They serve as a powerful tool in fields such as artificial intelligence, bioinformatics, and decision analysis by compactly modeling uncertainty and enabling efficient reasoning under incomplete information. Alongside these networks, the Variable Elimination (VE) algorithm stands out as a fundamental technique for performing exact inference. This essay provides a comprehensive discussion on Bayesian networks, details the VE algorithm, and explains the code implementation that reads Bayesian networks, processes factors, and executes variable elimination. Moreover, it delves into the complexity of the algorithm and the influence of various heuristics on performance.

## 2 Theory Background

### 2.1 Bayesian Networks: Definition and Key Components

A Bayesian network (BN) encapsulates uncertain knowledge about the world by using nodes to represent random variables and directed edges to capture direct probabilistic dependencies. The structure of the network encodes assumptions about conditional independencies — for example, the absence of an edge between two nodes suggests that they are conditionally independent given their parents. This factorization allows complex joint probability distributions to be expressed as a product of local conditional probability distributions.

**Key Components:**

- **Nodes (Variables):** Each node represents a random variable with a specific domain of possible values.
- **Edges (Dependencies):** Directed links that indicate causal or influential relationships between variables.
- **Conditional Probability Tables (CPTs):** For every node, the CPT quantifies the influence of its parent variables, providing probabilities for each possible state given every combination of parent states.

- **Graphical Structure:** The directed acyclic graph (DAG) ensures the absence of cycles, thus making probabilistic inference tractable.

Bayesian networks are highly effective for applications such as diagnostic systems, risk assessment, and predictive modeling.

## 2.2 The Inference Problem and Variable Elimination

One of the primary challenges in utilizing Bayesian networks is performing inference — computing the probability distribution of one or more query variables given observed evidence. Exact inference can be computationally intensive due to the potentially huge joint distribution, which is why methods like Variable Elimination (VE) are employed.

**Variable Elimination (VE)** is an algorithm that computes marginal distributions by "eliminating" (i.e., summing out) non-essential variables. It leverages the factorized structure provided by local CPTs to work with smaller factors rather than the full joint distribution. This process reduces computational complexity and ensures exact probabilities.

**Step-by-Step Process:**

1. **Incorporating Evidence (Simplification):** Observed evidence is integrated by filtering each factor (CPT) to retain only the rows consistent with the evidence. This reduces the dimensionality of the problem.

2. **Determining an Elimination Order:** The algorithm selects an order in which non-query, non-evidence variables are eliminated. The choice of this order is crucial; common heuristics include:

    - **Min-size/Min-weight:** Choose the variable whose elimination produces the smallest intermediate factor.
    - **Least-incoming-arcs:** Favor variables with fewer parent dependencies.
    - **Random:** Although sometimes used, this approach rarely yields optimal performance.

3. **Factor Multiplication:** For each variable to be eliminated, all factors that include it are multiplied (merged on common variables) to create a composite factor.

4. **Marginalization (Summing Out):** The target variable is summed out from the composite factor by grouping over the remaining variables and summing their probabilities.

5. **Final Multiplication and Normalization:** After eliminating all non-query variables, the remaining factors are multiplied together and normalized to ensure the final probability distribution sums to 1.

VE is particularly beneficial for exact inference in sparse networks with small domains or when only a subset of variables is queried.

## 3 Code Overview

The provided Python code implements the VE algorithm and supports operations on multi-dimensional factors. This section outlines the code structure and the role of each component.

### 3.1 Variable Elimination Class

The main class, responsible for performing inference, is invoked by a separate runner (Run.py). Its primary goal is to compute the probability distribution of a query variable given some observed evidence through systematic elimination of irrelevant variables.

**Overall Process:**

1. **Simplification:** Evidence is applied to the network's probability tables using the `simplification(self, observed)` function, reducing each factor to only include rows that match observed values.

2. **Elimination Order Determination:** The `elimination_order(self, heuristic, observed)` function selects the order in which variables are eliminated. This selection is based on various heuristics discussed later.

3. **Factor Operations:**

   - **Multiplication:** The `multiplication(self, factor1, factor2)` function multiplies two factors by aligning them on their common variables.

    - **Marginalization:** The `sum_out(self, variable, factor)` function eliminates a variable by summing over its possible values.

4. **Final Steps:** Remaining factors are multiplied together, extra variables are removed, and the final factor is normalized.

## 3.2 Main Functions

`run(self, query, observed, heuristic):` Acts as the central coordinator, executing the VE process by calling the simplification, elimination order, multiplication, and marginalization functions in sequence.

`simplification(self, observed):` Incorporates evidence into the network by filtering probability tables, thus reducing unnecessary computations.

`multiplication(self, factor1, factor2):` Combines two factors by joining on common variables and multiplying their probability values.

`sum_out(self, variable, factor):` Eliminates a variable from a factor by grouping over the remaining variables and summing the corresponding probabilities.

`elimination_order(self, heuristic, observed):` Determines the order in which variables are eliminated based on the selected heuristic. The heuristics available include:

- **Random**
- **Min-size/Min-weight**
- **Least-incoming-arcs**
- **Outgoing-arcs-first**
- **Fewest-factors**

## 3.3 Additional Runner Files

- **Run.py:** Loads a Bayesian network from a `.bif` file, prints the network structure, and creates a `VariableElimination` object. It then sets the query variable (e.g., “Alarm”), defines evidence (e.g., {`'Burglary': 'True'`}), chooses a heuristic (e.g., “min-weight”), and runs the VE algorithm.

- **Survey_network.py:** Demonstrates the code on a more complex Bayesian network, testing the algorithm on networks with non-binary variables.

## 4 Complexity and Heuristic Considerations

A critical aspect of the VE algorithm is the selection of an optimal elimination order, as it directly influences both the computational cost of the algorithm and the size of intermediate factors. The code iterates over remaining variables and applies different heuristics, each with its own complexity and impact.

### 4.1 Overall Loop Structure

The elimination order is selected within a loop that iterates until all non-query and non-evidence variables are eliminated. In the worst-case scenario, if there are $n$ variables, the loop iterates $n$ times. Within each iteration, a candidate variable is selected based on the chosen heuristic.

### 4.2 Detailed Heuristic Analysis

#### Random Heuristic

```
var = random.choice(list(remaining_variables))
```

**Complexity:** Converting the set to a list is $O(n)$ and selecting a random element is $O(1)$. Over $n$ iterations, the overall complexity is $O(n^2)$.

**Impact:** This heuristic ignores network structure, potentially leading to large intermediate factors and less efficient elimination.

#### Min-Size / Min-Weight Heuristic

```
var = min(remaining_variables, key=lambda var: len(self.network.probabilities[var]
```

**Complexity:** Evaluating the size of each factor is typically $O(1)$, leading to $O(n)$ per iteration and $O(n^2)$ overall.

**Impact:** By selecting the variable with the smallest factor, this heuristic minimizes the computational cost of subsequent operations.

**Least-Incoming-Arcs Heuristic**

```
var = min(remaining_variables, key=lambda var: len(self.network.parents[var]))
```

**Complexity:** Counting incoming arcs is an $O(1)$ operation per variable. Thus, each iteration is $O(n)$ and the overall complexity is $O(n^2)$.

**Impact:** Favoring variables with fewer parent dependencies tends to yield smaller CPTs, leading to smaller intermediate factors.

**Outgoing-Arcs-First Heuristic**

```
var = max(remaining_variables, key=lambda var: sum(1 for child in self.network.noc
```

**Complexity:** For each candidate, summing over all nodes is $O(n)$, and across $n$ variables, each iteration becomes $O(n^2)$. Over all iterations, the complexity can reach $O(n^3)$.

**Impact:** Although computationally expensive, eliminating variables with many outgoing connections early can simplify the network structure and reduce later computational costs. However, the overhead may sometimes outweigh the benefits.

**Fewest-Factors Heuristic**

```
var = min(remaining_variables, key=lambda var: sum(1 for table in self.network.prc
```

**Complexity:** Iterating over all factors (roughly $O(n)$) for each candidate results in $O(n)$ per iteration, or $O(n^2)$ overall.

**Impact:** This heuristic minimizes the number of multiplications by favoring variables that appear in the fewest factors, resulting in fewer and smaller intermediate results.

## 4.3 Trade-offs and Practical Considerations

- **Computational Overhead vs. Factor Size:** Heuristics such as min-size, least-incoming-arcs, and fewest-factors generally have lower overhead ($O(n^2)$) compared to outgoing-arcs-first ($O(n^3)$). However, if a deeper network analysis (like outgoing-arcs-first) produces significantly smaller factors, it may be beneficial despite the higher computational cost.

- **Scalability:** In larger networks, even minor differences in per-iteration cost can accumulate, so the balance between computational feasibility and the quality of the elimination order becomes critical.
- **Empirical Performance:** The optimal heuristic is highly dependent on the network's structure and the nature of its CPTs. Experimentation is key, as theoretical complexities provide a guideline while actual performance may vary.

## 5 Discussion

The effectiveness of variable elimination is intricately linked to the chosen elimination order. A poor order can lead to exponentially larger intermediate factors, drastically increasing computational time. The discussion above shows that while certain heuristics (e.g., random selection) are simple to implement, they may result in inefficient processing. Conversely, more sophisticated heuristics like outgoing-arcs-first offer the potential for significant efficiency gains but come at the cost of higher per-iteration overhead. This trade-off highlights the need for careful heuristic selection tailored to the specific characteristics of the Bayesian network under consideration.

Furthermore, the code implementation illustrates a structured approach to handling these operations, from evidence incorporation to final normalization. It serves as a practical example of how theoretical concepts in probabilistic inference translate into algorithmic steps and programming challenges.

## 6 Conclusion

In summary, Bayesian networks and the variable elimination algorithm provide a robust framework for exact probabilistic inference. The VE algorithm efficiently computes marginal distributions by leveraging the factorized structure of Bayesian networks. However, the choice of elimination order is crucial, as it directly impacts both the size of intermediate factors and overall computational complexity. The detailed analysis of various heuristics underscores the balance between computational overhead and the quality of intermediate results. The provided Python code offers a clear, modular implementation of these ideas, serving as both a learning tool and a practical example of advanced probabilistic inference.

## 7 References

1. Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers.

2. Jensen, F. V. (1996). *An Introduction to Bayesian Networks.* UCL Press.

3. Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques.* MIT Press.

4. Neapolitan, R. E. (2003). *Learning Bayesian Networks.* Pearson Prentice Hall.

5. Heckerman, D. (1995). A Tutorial on Learning with Bayesian Networks. Microsoft Research.