Lossless Bayesian Network Implementation

Exact Probabilistic Inference in C++

Shyamal Chandra

2025

Overview

What is a Bayesian Network?

- Probabilistic graphical model
- Represents variables and their conditional dependencies
- Directed Acyclic Graph (DAG) structure
- Enables efficient probabilistic reasoning

Joint Distribution Factorization:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \mathrm{Pa}(X_i))$$

What Does "Lossless" Mean?

- **Exact computation**: No approximation in probability calculations
- **Full precision**: All probability values maintained exactly
- **Complete information**: No loss of probabilistic information
- **Deterministic results**: Same inputs always produce same outputs

Key Advantage

Unlike approximate methods (MCMC, variational inference), lossless methods provide exact results, crucial for applications requiring precision.

System Architecture

Core Components:

- `Node` class
  - Variable representation
  - State management
  - Parent relationships
- `CPT` class
  - Conditional probabilities
  - Multi-dimensional storage
  - Normalization
- `BayesianNetwork` class
  - Network construction
  - Inference algorithms
  - File I/O

`network_structure.pdf`

(Conceptual diagram)

Node Structure

```
class Node {
    std::string name;                    // Variable name
    std::vector<std::string> states;     // Possible states
    std::set<std::string> parentIds;    // Parent nodes
    std::map<std::string, int> stateIndexMap;  // Fast lookup
};
```

Features:

- Fast state lookup: O(1) via hash map
- Parent tracking for DAG structure
- Flexible state definitions

Conditional Probability Table

```cpp
class ConditionalProbabilityTable {
    std::vector<double> probabilities;  // Flat storage
    std::vector<size_t> dimensions;     // Multi-dim sizes
    std::vector<size_t> strides;        // Indexing strides
};
```

Multi-dimensional Indexing:

$$\text{index} = \sum_{k=0}^{n-1} i_k \cdot \text{stride}_k$$

Features:

- Efficient storage in flat array
- Automatic normalization
- Validation of probability distributions

Topological Sorting

Kahn's Algorithm:

1. Compute in-degrees for all nodes
2. Initialize queue with nodes (in-degree $= 0$)
3. While queue not empty:
   - Remove node from queue
   - Decrease in-degree of children
   - Add children with in-degree $= 0$ to queue
4. If processed nodes $<$ total nodes: **CYCLE DETECTED**

Purpose

Ensures DAG property and provides ordering for efficient inference.

Exact Inference Algorithm:

1. **Query**: Variables of interest Q
2. **Evidence**: Observed variables $E = e$
3. **Hidden**: Variables to sum out H
4. Compute: $P(Q|E = e) = \frac{\sum_H P(Q,E=e,H)}{\sum_{Q,H} P(Q,E=e,H)}$

Complexity

Exponential in the number of variables, but exact results.

Medical Diagnosis Example

Network Structure:

- `Disease` $\rightarrow$ `Fever`
- `Disease` $\rightarrow$ `Cough`

Inference: Given: Fever=Yes, Cough=Yes
Query: P(Disease)?

Results

- P(Disease=None) = 0.001
- P(Disease=Cold) = 0.234
- P(Disease=Flu) = 0.765

Alarm Network Example

Classic Bayesian Network:

Nodes:

- Burglary
- Earthquake
- Alarm
- JohnCalls
- MaryCalls

Edges:

- Burglary $\rightarrow$ Alarm
- Earthquake $\rightarrow$ Alarm
- Alarm $\rightarrow$ JohnCalls
- Alarm $\rightarrow$ MaryCalls

Inference: Given JohnCalls=True, MaryCalls=True
Query: P(Burglary=True)?

Code Example: Network Construction

```
BayesianNetwork network;

// Add nodes
network.addNode("Disease", "Disease",
                {"None", "Cold", "Flu"});
network.addNode("Symptom", "Fever", {"No", "Yes"});

// Add edge
network.addEdge("Disease", "Symptom");

// Create CPT
std::vector<size_t> dims = {3, 2};
ConditionalProbabilityTable cpt(dims);
cpt.setProbability({0}, 0, 0.9);  // P(No|None) = 0.9
cpt.setProbability({0}, 1, 0.1);  // P(Yes|None) = 0.1
cpt.normalize();
network.setCPT("Symptom", cpt);
```

Code Example: Inference

```
// Set evidence
std::map<std::string, std::string> evidence;
evidence["Symptom"] = "Yes";

// Perform inference
std::vector<std::string> query = {"Disease"};
auto results = network.variableElimination(query, evidence);

// Display results
for (const auto& pair : results) {
    std::cout << "P(Disease=" << pair.first.at("Disease")
              << ")␣=␣" << pair.second << std::endl;
}
```

Key Features

- **Lossless Representation**
  - Exact probability storage
  - No approximation errors
- **Exact Inference**
  - Variable elimination algorithm
  - Deterministic results
- **DAG Validation**
  - Automatic cycle detection
  - Topological sorting
- **Flexible API**
  - Easy network construction
  - Comprehensive error handling
- **File I/O**
  - Network serialization
  - Persistent storage

Summary

- Complete C++ implementation of lossless Bayesian networks
- Exact inference using variable elimination
- Efficient data structures for probability storage
- Comprehensive documentation and examples

Applications

- Medical diagnosis systems
- Risk assessment
- Decision support systems
- Any application requiring exact probabilistic reasoning

Questions?

Thank you for your attention!

For more information, see the full documentation and reference manual.