CS 486/686
Variable Elimination Algorithm

Yuntian Deng

Lecture 9

RN 13.4 · PM 8.4

Reminders

✅ **Chrysalis was down earlier today** (server issue) but is **back up now**. It's still offline for the scheduled outage **Thu Jun 11 (9:30 AM) – Fri Jun 12 (noon)**.

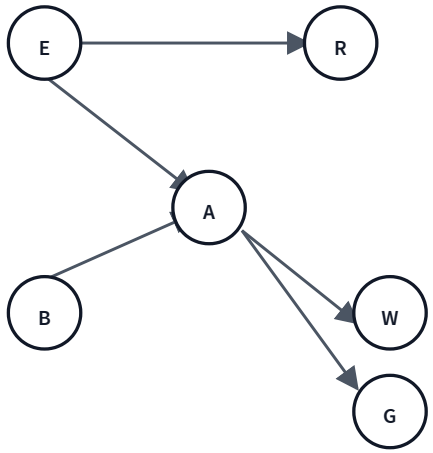
📢 Deadlines still extended: **Chat 4 → Tue Jun 16, Chat 5 → Thu Jun 18**.

📝 **Assignment 1** questions → **Dake Zhang** on Piazza.

Learning goals

- Do inference in Bayes nets **more efficiently** than the joint
- Define **factors** and the four operations on them
- Trace the **variable elimination algorithm** for prior / posterior queries
- Explain how the **elimination ordering** affects complexity
- Know the basics of **approximate** inference (sampling)

A query in the Holmes BN: $P(B|w \wedge g)$



"What's the probability of a burglary, given that both Watson and Gibbon called?"

- **Query variable:** B
- **Evidence variables:** $W = w, G = g$
- **Hidden variables:** E, A, R (to be summed out)

Convention: capital letters are random variables, lowercase letters are specific values.

From a conditional to a sum

By the definition of conditional probability:

$$P(B \mid w \wedge g) = \frac{P(B, W{=}w, G{=}g)}{P(w \wedge g)}$$

The denominator $P(w \wedge g)$ doesn't depend on B, so it's just a normalizing constant:

$$P(B \mid w \wedge g) \;\propto\; P(B, W{=}w, G{=}g)$$

Recover that joint by summing out the hidden variables E, A, R:

$$P(B, W{=}w, G{=}g) = \sum_{e,\,a,\,r} P(B, e, a, r, W{=}w, G{=}g)$$

Naive: brute-force sum-out

Apply the BN factorization to that joint:

$$P(B|w \wedge g) \propto \sum_e \sum_a \sum_r P(B)\, P(e)\, P(a|B \wedge e)\, P(w|a)\, P(g|a)\, P(r|e)$$

Q1. How many additions + multiplications does this require?

A. ≤ 10

B. 11–20

C. 21–40

D. 41–60

E. ≥ 61

D — 47 ops. Eight terms to sum, each needing 5 multiplications: $8 \times 5 + 8 - 1 = 47$ ops. Lots of redundant work.

Push the sums to the right

$P(B)$ doesn't depend on e, a, r — pull it out. $P(r|e)$ only depends on e, r — push $\sum_r$ inward. And so on:

$$P(B) \sum_e P(e) \underbrace{\sum_r P(r|e)}_{=1} \sum_a P(a|B \wedge e)\, P(w|a)\, P(g|a)$$

$$= P(B) \sum_e P(e) \sum_a P(a|B \wedge e)\, P(w|a)\, P(g|a)$$

Q2. Ops now?

> **B — 14 ops.** Inner sum over a: $4\ \text{mul} + 1\ \text{add} = 5$; times 2 for the two values of $e \Rightarrow 10$; multiply each by $P(e)$: $+2$; sum the two e-terms: $+1$; multiply by $P(B)$: $+1$. Total $= 14$.
> **~3x speedup just from re-ordering.**

The big idea

- **Reuse** intermediate computations — dynamic programming for probabilities.
- **Exploit** conditional independence baked into the Bayes net.
- Push sums in *as far as possible*, then operate on small tables of probabilities — **factors**.

This is the **Variable Elimination Algorithm**.

Factors

A **factor** $f(X_1, \ldots, X_j)$ is a function from random-variable assignments to a number. It can represent a joint, a conditional, or a partial assignment.

For the Holmes BN, define one factor per CPT:

$$f_1(B) = P(B), \quad f_2(E) = P(E)$$
$$f_3(A, B, E) = P(A|B \wedge E), \quad f_4(R, E) = P(R|E)$$
$$f_5(W, A) = P(W|A), \quad f_6(G, A) = P(G|A)$$

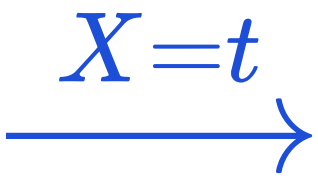
Restrict: assign a variable

Restricting $f(X_1, \ldots, X_j)$ to $X_1 = v_1$ fixes that variable, leaving a smaller factor on $X_2, \ldots, X_j$.

Example: $f_1(X, Y) \rightarrow f_2(Y) = f_1(X = t, Y)$

$f_1(X, Y)$

X	Y	val
t	t	0.1
t	f	0.9
f	t	0.2
f	f	0.8



$f_2(Y)$

Y	val
t	0.1
f	0.9

Sum out: marginalize a variable

Summing out X_1 from $f(X_1, \ldots, X_j)$ gives a factor on $X_2, \ldots, X_j$ defined by $\left(\sum_{X_1} f\right)(X_2, \ldots, X_j) = \sum_v f(X_1 = v, X_2, \ldots, X_j)$.

Example: $\sum_Y f_1(X, Y) \rightarrow f_2(X)$

$f_1(X, Y)$

X	Y	val
t	t	0.1
t	f	0.4
f	t	0.3
f	f	0.5

$\xrightarrow{\sum_Y}$

$f_2(X) = \sum_Y f_1$

X	val
t	0.1 + 0.4 = 0.5
f	0.3 + 0.5 = 0.8

Multiply two factors

$(f_1 \times f_2)(X, Y, Z) = f_1(X, Y) \cdot f_2(Y, Z)$ for shared Y — the product is over the *union* of variables.

$f_1(X, Y)$

X	Y	val
t	t	0.1
t	f	0.9
f	t	0.2
f	f	0.8

×

$f_2(Y, Z)$

Y	Z	val
t	t	0.3
t	f	0.7
f	t	0.6
f	f	0.4

=

$f_1 \times f_2$

X	Y	Z	val
t	t	t	0.03
t	t	f	0.07
t	f	t	0.54
t	f	f	0.36
f	t	t	0.06
f	t	f	0.14
f	f	t	0.48
f	f	f	0.32

Normalize a factor

Divide every entry by the sum, so the values sum to 1 (a valid probability distribution).

$f(Y)$

Y	val
t	0.2
f	0.6

$P(Y)$

Y	val
t	0.25
f	0.75

$$0.2/(0.2+0.6) = 0.25,\ 0.6/0.8 = 0.75.$$

The variable elimination algorithm

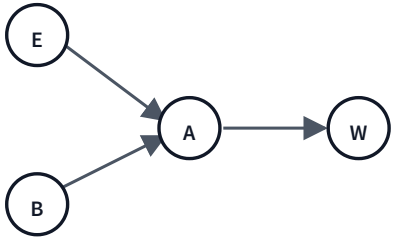
To compute $P(X_q | X_{o_1} = v_1 \wedge \cdots \wedge X_{o_j} = v_j)$:

`variable-elimination(BN, query, evidence)`

1. Construct a factor for each CPT in the BN.
2. Restrict every observed variable to its value.
3. For each hidden variable X_h (in chosen order): multiply all factors that contain X_h, then sum out X_h from the product.
4. Multiply the remaining factors.
5. Normalize the result.

Each iteration removes one hidden variable while keeping all factors small.

Worked example: $P(B|\neg a)$



Sub-BN for the example. B = query, $A = \text{false}$ = evidence.

Given CPTs:

$$P(B) = 0.3, \quad P(E) = 0.1$$
$$P(A|E, B) = 0.8, \quad P(A|E, \neg B) = 0.2$$
$$P(A|\neg E, B) = 0.7, \quad P(A|\neg E, \neg B) = 0.1$$
$$P(W|A) = 0.8, \quad P(W|\neg A) = 0.4$$

Define factors:

$$f_1(B), \; f_2(E), \; f_3(A, B, E), \; f_4(W, A)$$

Eliminate hidden variables

$f_5(B, E) = f_3(\neg a, B, E)$

B	E	val
t	t	0.2
t	f	0.3
f	t	0.8
f	f	0.9

$f_8(B, E) = f_2 \times f_5$

B	E	val
t	t	0.2 × 0.1 = 0.02
t	f	0.3 × 0.9 = 0.27
f	t	0.8 × 0.1 = 0.08
f	f	0.9 × 0.9 = 0.81

$f_9(B) = \sum_E f_8$

B	val
t	0.02 + 0.27 = 0.29
f	0.08 + 0.81 = 0.89

Multiply remaining factors, normalize

Remaining factors: $f_1(B), f_9(B), f_7() = 1.0$. Multiply them:

$f_{10}(B) = f_1 \times f_9 \times f_7$

B	val
t	0.3 × 0.29 × 1 = 0.087
f	0.7 × 0.89 × 1 = 0.623

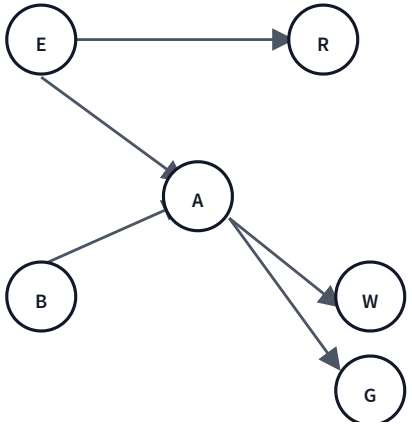
normalize →

$P(B|\neg a)$

B	val
t	0.087 / 0.710 ≈ **0.1225**
f	0.623 / 0.710 ≈ **0.8775**

Burglary is unlikely ($\approx 12\%$) given that the alarm did NOT sound.

Ordering matters



Compute $P(G)$ (no evidence). Initial factors:

$$f_1(E), f_2(E, R), f_3(B)$$
$$f_4(E, B, A), f_5(W, A), f_6(G, A)$$

- **Every** ordering of $\{R, W, E, B, A\}$ yields a correct algorithm.
- Different orderings produce **different intermediate factors**.
- Larger intermediate factors $\Rightarrow$ more space + ops.

Two orderings, very different costs

Good: $R \to W \to E \to B \to A$

- $f_2(E, R) \to f_7(E)$: 2 ops
- $f_5(W, A) \to f_8(A)$: 2 ops
- $f_1 \cdot f_7 \cdot f_4 \to f_9(B, A)$: 20 ops
- $f_9 \cdot f_8 \to f_{10}(A)$: 6 ops
- $f_{10} \cdot f_6 \to f_{11}(G)$: 6 ops

Total: 36 ops

Bad: $A \to B \to E \to R \to W$

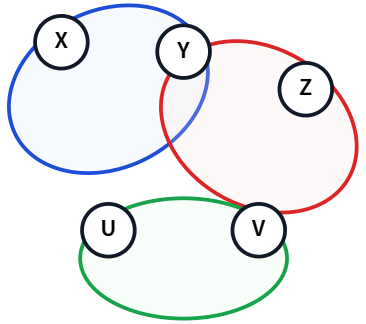
- $f_4 \cdot f_5 \cdot f_6 \to f_7(E, B, W, G)$: 80 ops
- $f_7 \cdot f_3 \to f_8(E, W, G)$: 24 ops
- $f_1 \cdot f_8 \cdot f_2 \to f_9(W, G, R)$: 24 ops
- $f_9 \to f_{10}(W, G)$: 4 ops
- $f_{10} \to f_{11}(G)$: 2 ops

Total: 134 ops

The bad ordering builds a 4-variable intermediate factor $f_7(E, B, W, G)$ — **that's what we pay for.**

Elimination width + tree width

- **Elimination width:** the largest factor (hyperedge) an ordering ever builds.
- **Tree width** ω = smallest width over all orderings; VE costs $2^{O(\omega)}$ time and space.
- Optimal ordering is **NP-hard**, but heuristics do well ($\omega \approx 8$–10 even with 1000s of variables).

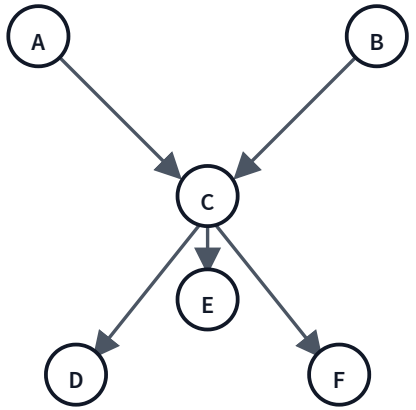


Hyperedges (colored bubbles) = factors.

Polytrees: an easy case

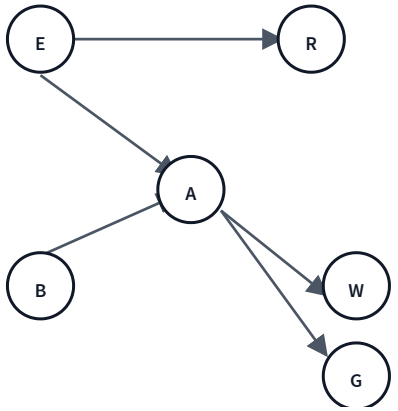
A **polytree** = a **singly-connected** BN: at most one *undirected* path between any two nodes — no loops.

- **Singly-connected** describes the *whole graph*, not one node. Multiple parents are OK: C below has two (A, B) — still a polytree.
- A *diamond* ($A \to B \to D$, $A \to C \to D$) is **not** — two routes A to D.
- Heuristic: eliminate a **single-neighbor** node (everything but C) — never enlarges a factor.
- A node with ≥ 2 parents (like C) waits till last. Tree width = max # parents $\Rightarrow$ VE is **linear**.



A, B, D, E, F each have one neighbor (C) — peel those off first; C (2 parents) last.

Forward sampling: estimate $P(G)$

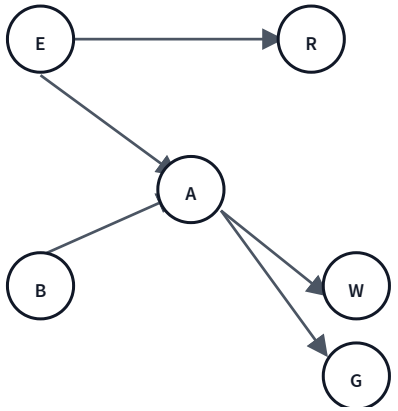


When the BN is too large for exact VE, just **simulate** it.

```
for i in 1..n: e_i ~ P(E) b_i ~ P(B) a_i ~ P(A | E=e_i, B=b_i) g_i ~ P(G | A=a_i) P(G = T) ≈ (1/n) ∑_i g_i
```

Sample in **topological order** so every CPT input is already known. Estimate converges to true $P(G)$ as $n \to \infty$.

Rejection sampling: estimate $P(G|\neg b)$




```
for i in 1..n: forward-sample
e_i, b_i, a_i, g_i if b_i == T:
reject sample else: keep g̃_i
P(G = T | ¬b) ≈ (∑_i g̃_i) / ñ
```

- Only "accepted" samples (consistent with evidence) count.
- **Wasteful** when evidence is rare — most samples thrown away.

Learning goals (recap)

✓ Do inference in Bayes nets **more efficiently** than the joint

✓ Define **factors** and the four operations on them

✓ Trace the **variable elimination algorithm**

✓ Explain how the **elimination ordering** affects complexity

✓ Know the basics of **approximate** inference

Next: Hidden Markov Models

A Bayes net **unrolled over time** — inference about what's happening *now* given a sequence of observations.

VEA in disguise: forward-backward, Viterbi, smoothing.