

Exact Inference

Aayush Bajaj

August 31, 2026

a bayesian network stores a joint distribution; inference is spending that investment — computing probabilities the tables do not state directly. exact inference is np-hard in general, but its true cost is exponential only in a structural parameter of the graph (the *treewidth*), not in the number of variables. this page builds the exact toolchain in the order the ideas force themselves on you: factors, variable elimination, the fight for a good elimination order, and finally the jointree algorithm — variable elimination industrialised into a message-passing scheme that answers every marginal at once.

when the treewidth is too large, exact methods drown and you switch to the sampling algorithms on the approximate inference page

Contents

1 the queries	1
2 factors	2
3 variable elimination	2
3.1 worked example: $P(E)$ in the sprinkler network	2
3.2 pruning before you eliminate	3
4 elimination orders and treewidth	3
4.1 the interaction graph	3
4.2 treewidth and heuristics	3
5 from variable to factor elimination	4
5.1 message passing	4
6 the jointree algorithm	4
6.1 shenoy-shafer vs hugin	5
7 mpe and map by elimination	6
7.1 mpe: replace sum with max	6
7.2 map: the order is constrained	6
8 results	6
9 see also	7

1 the queries

four query types cover practically everything asked of a network with variables $\mathbf{V}$, evidence $\mathbf{e}$, and query variables $\mathbf{Q} \subseteq \mathbf{V}$:

- **probability of evidence** — $P(\mathbf{e})$, a single number; the normalising constant of everything else.

- **marginals** — prior $P(\mathbf{Q})$ or posterior $P(\mathbf{Q} \mid \mathbf{e})$; a table, not a number. computed as $P(\mathbf{Q}, \mathbf{e})$, then normalised: $P(\mathbf{Q} \mid \mathbf{e}) = P(\mathbf{Q}, \mathbf{e}) / \sum_{\mathbf{q}} P(\mathbf{q}, \mathbf{e})$.
- **most probable explanation (mpe)** — the single most likely instantiation of *all* unobserved variables given $\mathbf{e}$.
- **maximum a posteriori (map)** — the most likely instantiation of a chosen *subset* $\mathbf{M}$ of variables, the rest summed out. mpe is the special case $\mathbf{M} = \mathbf{V} \setminus \mathbf{E}$; despite the containment, map is computationally much harder.

a common abuse maps mpe answers onto map questions because mpe is cheap — the projections of an mpe need not maximise the sub-query

2 factors

the working data structure is the **factor**: any function $f(\mathbf{X})$ from instantiations of a variable set $\text{vars}(f) = \mathbf{X}$ to non-negative reals. cpts are factors; intermediate results of inference are factors that are usually not distributions of anything. two operations suffice:

- **multiplication** — $(f_1 f_2)(\mathbf{x}) = f_1(\mathbf{x}_1) f_2(\mathbf{x}_2)$, matching on shared variables. multiplying m factors whose union has w variables costs $O(m \exp w)$ time and space.
- **summing out** — $(\sum_X f)(\mathbf{y}) = \sum_x f(x, \mathbf{y})$, collapsing a dimension. $O(\exp w)$ for a factor of w variables.

evidence enters by **reduction**: zero out (or drop) every row of every factor inconsistent with $\mathbf{e}$. the chain rule says the product of all cpt factors is the joint, so in principle any query is “multiply everything, sum out the rest” — at exponential cost. the entire subject of exact inference is doing the sums *early*.

3 variable elimination

interchange summation and multiplication wherever the distributive law allows: to sum out X , only the factors mentioning X need multiplying first. the algorithm is three loops:

1. collect the factors mentioning the next variable $\pi(i)$ in the elimination order π ;
2. multiply them and sum $\pi(i)$ out, producing one new factor;
3. return it to the pool; repeat, and finally multiply what remains.

3.1 worked example: $P(E)$ in the sprinkler network

eliminating with order $\pi = B, C, A, D$ from the factor pool $\{\theta_A, \theta_{B|A}, \theta_{C|A}, \theta_{D|B,C}, \theta_{E|C}\}$:

step	eliminate	computation	vars in new factor
1	B	$f_1(A, C, D) = \sum_B \theta_{B A} \theta_{D B,C}$	3
2	C	$f_2(A, D, E) = \sum_C \theta_{C A} \theta_{E C} f_1$	3
3	A	$f_3(D, E) = \sum_A \theta_A f_2$	2
4	D	$f_4(E) = \sum_D f_3$	1

f_4 is $P(E)$. the size-3 intermediate factors are the price of this order; the **width of an order** is the number of variables in the largest factor built along the way (here 3), and runtime is

$$O(n \exp(w) + n \exp(|\mathbf{Q}|)) \quad (1)$$

for n variables and order width w . a bad order — eliminating a hub variable first, so every factor gets multiplied at once — can blow the width up to n .

3.2 pruning before you eliminate

for a specific query $(\mathbf{Q}, \mathbf{e})$, the network usually contains structure the query cannot feel, and deleting it first is free speedup:

- **node pruning** — repeatedly remove any leaf not in $\mathbf{Q} \cup \mathbf{E}$ (a barren node sums out to a factor of ones);
- **edge pruning** — delete edges leaving evidence nodes, substituting the observed value into the child cpts.

the result can be dramatically smaller, and the same surgery is the engine of the linear-time d-separation test.

4 elimination orders and treewidth

4.1 the interaction graph

widths live on an undirected object. the **interaction graph** of a factor pool joins two variables iff they appear in a common factor; for a bayesian network's cpts this is the **moral graph** — connect (“marry”) every pair of parents with a common child, then drop directions.

the moral graph is exactly the markov network that is a minimal i-map of the bayesian network — see the markov networks page

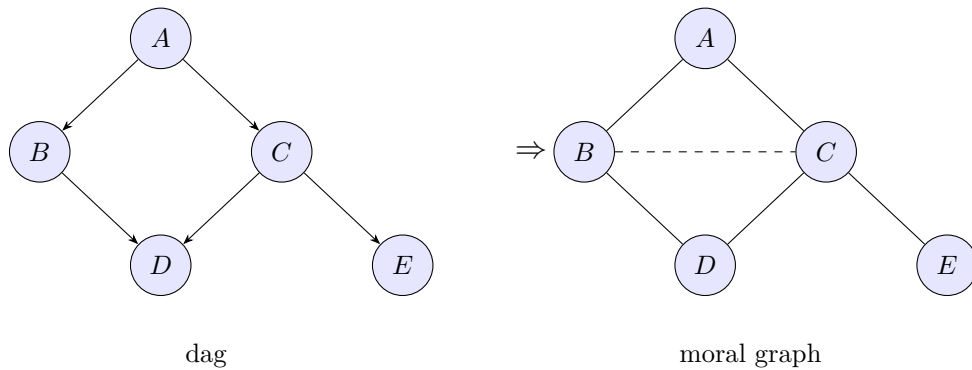


Figure 1: moralising the sprinkler network: parents B and C of the common child D are married (dashed), then directions are dropped. widths of elimination orders are read off this graph.

simulating elimination on the interaction graph: to eliminate X , connect all its neighbours pairwise (these **fill-in edges** are the new factor), then delete X . the width of the order is the largest neighbourhood encountered.

4.2 treewidth and heuristics

the **treewidth** of a graph is the minimum width over all $n!$ elimination orders — a property of the graph alone, and the fundamental barrier: no elimination order, and in a precise sense no exact algorithm of this family, beats $\exp(\text{treewidth})$. computing it is np-hard, so practice runs on greedy heuristics:

- **min-degree** — eliminate the variable with fewest neighbours next; cheap and good.
- **min-fill** — eliminate the variable adding fewest fill-in edges; usually better, slightly dearer.

optimal orders can be found for modest graphs by branch-and-bound over prefixes, pruned with treewidth lower bounds (e.g. mmd — maximum minimum degree) and simplification rules (simplicial and almost-simplicial vertices can be eliminated first without loss). useful facts: the number of nodes does not lower-bound treewidth, but the largest clique minus one does, and trees have treewidth 1 — inference on polytrees is linear.

5 from variable to factor elimination

variable elimination answers one query; ask for the posterior of every variable and you pay $O(n^2 \exp w)$. the fix is a change of viewpoint. **factor elimination** removes *factors* rather than variables: to eliminate factor f_i , sum out the variables appearing only in f_i , and multiply the result into some surviving neighbour f_j . organising the eliminations along a tree — an **elimination tree** over the factors — makes the two open choices (which factor next, into whom) canonical: repeatedly eliminate a leaf into its unique neighbour, towards a chosen root that contains the query.

two definitions carry the analysis:

- the **separator** of edge (i, j) is $S_{ij} = \text{vars}(i\text{'s side}) \cap \text{vars}(j\text{'s side})$ — the variables that must *not* be summed out when crossing the edge;
- the **cluster** of node i is $C_i = \text{vars}(f_i) \cup \bigcup_j S_{ij}$; the tree's width is the largest cluster size minus one.

5.1 message passing

eliminating leaf i into neighbour j is precisely sending a **message**

$$M_{ij} = \sum_{C_i \setminus S_{ij}} \phi_i \prod_{k \neq j} M_{ki}, \quad (2)$$

multiply what has arrived, sum down to the separator, pass it on. to answer a query at root r : push all messages towards r , multiply them into ϕ_r , sum out what the query does not mention. the dynamic-programming payoff: messages do not depend on the root. running an **inward** pass to any node and an **outward** pass back — $2(m-1)$ messages for m factors — leaves every node holding all its incoming messages, so *every* cluster marginal $P(C_i, \mathbf{e})$ is available for the price of two sweeps: $O(m \exp w)$ total.

this is the same trick as forward-backward in hmms — which is literally this algorithm on a chain

6 the jointree algorithm

a **jointree** (junction tree, clique tree) for a graph G is a tree T whose nodes carry variable-set labels (**clusters**) C_i such that:

1. every factor of the model fits inside some cluster;

2. **running intersection property (rip)**: if a variable appears in clusters C_i and C_j , it appears in every cluster on the path between them.

separators are edge intersections $S_{ij} = C_i \cap C_j$; jointree width is the largest cluster minus one. jointrees and elimination orders are inter-convertible in polynomial time, width for width: elimination along an order yields the **cluster sequence** whose maximal clusters, tree-connected via rip (e.g. by maximum-spanning-tree on separator sizes), form a jointree; conversely a jointree induces elimination orders of the same width. non-maximal clusters are absorbed into the clusters containing them.

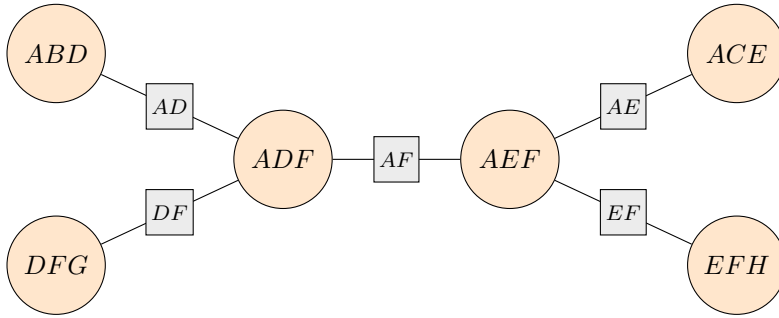


Figure 2: a jointree for a network over $A, \dots, H$. rectangles are separators $S_{ij} = C_i \cap C_j$; every cpt lives in a cluster that covers it, and each variable's clusters form a connected subtree (rip).

the **classical jointree algorithm**:

1. construct a jointree for the network;
2. assign each cpt (and each evidence indicator λ_X) to a cluster containing its variables;
3. propagate: inward pass to a root, outward pass back;
4. read off $P(C_i, \mathbf{e})$ at every cluster; any marginal over variables sharing a cluster is a summation away, and $P(\mathbf{e})$ is the normalisation at any node.

queries whose variables do not share a cluster need either an out-of-clique computation or a jointree modification — clusters can be merged or extended (at a width cost) while preserving rip.

6.1 shenoy-shafer vs hugin

the two classical propagation architectures trade space against time:

- **shenoy-shafer** — store all $2(m-1)$ messages; a cluster's belief is recomputed by multiplying its stored inputs. more space-frugal per cluster but re-multiplies; supports retracting evidence trivially (messages are pure functions of their subtrees).
- **hugin** — store a running potential per cluster and per separator; a message *updates* the neighbour's potential by multiplying the ratio of new to old separator marginal. faster propagation via division, more storage in the potentials, and division by zero needs care.

7 mpe and map by elimination

7.1 mpe: replace sum with max

maximisation distributes over products exactly as summation does, so variable elimination with max in place of $\sum$ computes

$$\text{MPE}_P(\mathbf{e}) = \max_{\mathbf{v}} \prod_i \theta_{v_i | \mathbf{u}_i} \Big|_{\mathbf{e}}, \quad (3)$$

with identical complexity $O(n \exp w)$. to recover the maximising *instantiation*, not just its probability, use **extended factors** that carry along the argmax choices made at each max-out, then backtrack from the final trivial factor. probabilities shrink as evidence accumulates, so implementations work in log-space, turning products into numerically stable sums.

on an hmm's chain-structured network, max-out ve with the temporal order is exactly the viterbi algorithm, and sum-out ve is exactly the forward algorithm

7.2 map: the order is constrained

for map over variables $\mathbf{M}$, sum out $\mathbf{V} \setminus \mathbf{M}$ *first*, then max out $\mathbf{M}$:

$$\text{MAP}_P(\mathbf{M}, \mathbf{e}) = \arg \max_{\mathbf{m}} \sum_{\mathbf{v} \setminus \mathbf{m}} P(\mathbf{v}, \mathbf{e}). \quad (4)$$

$\sum$ and max do not commute, so the elimination order must keep all map variables last — a *constrained* width that can be exponentially worse than the unconstrained one. this is the structural reason map is harder than mpe: an hmm's mpe is linear (viterbi) while a map over a scattered subset of its states can be intractable on the same network.

8 results

- **variable elimination** — computes $P(\mathbf{Q}, \mathbf{e})$ in $O(n \exp w)$ time and space, w the width of the elimination order used.
- **width via interaction graph** — eliminating X connects its neighbours (fill-in) and deletes it; order width = largest neighbourhood met.
- **treewidth** — $\min_{\pi} \text{width}(\pi)$; np-hard to compute; min-degree and min-fill are the workhorse heuristics; cliques give lower bounds, trees have treewidth 1.
- **jointree $\leftrightarrow$ elimination order** — polynomial, width-preserving conversions in both directions.
- **jointree propagation** — $2(m-1)$ messages, $O(m \exp w)$ total, yields *all* cluster marginals $P(C_i, \mathbf{e})$ and $P(\mathbf{e})$.
- **mpe by max-out ve** — same complexity as sum-out ve; instantiation recovered with extended factors.
- **map by constrained ve** — map variables eliminated last; constrained width $\geq$ width, sometimes exponentially so.

9 see also

- bayesian networks — the models these algorithms run on
- markov networks — same algorithms, undirected factors, plus the partition function
- approximate inference — sampling and loopy propagation when treewidth wins
- markov chains and hidden markov models — forward and viterbi as chain-structured special cases
- modeling and reasoning with bayesian networks — darwiche's book, chapters 6–13 cover this page in depth