Exact Inference

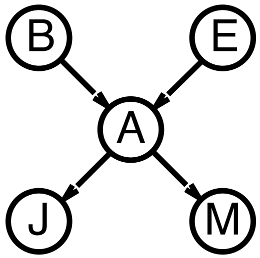
Inference

- Basic task for inference:
  - Compute a posterior distribution for some query variables given some observed evidence
  - Sum out nuisance variables
- In general inference in GMs is intractable…
  - Tractable in certain cases, e.g. HMMs, trees
  - Approximate inference techniques
    - Active research area…
  - More later

Inference by enumeration

Slightly intelligent way to sum out variables from the joint without actually constructing its explicit representation

Simple query on the burglary network:

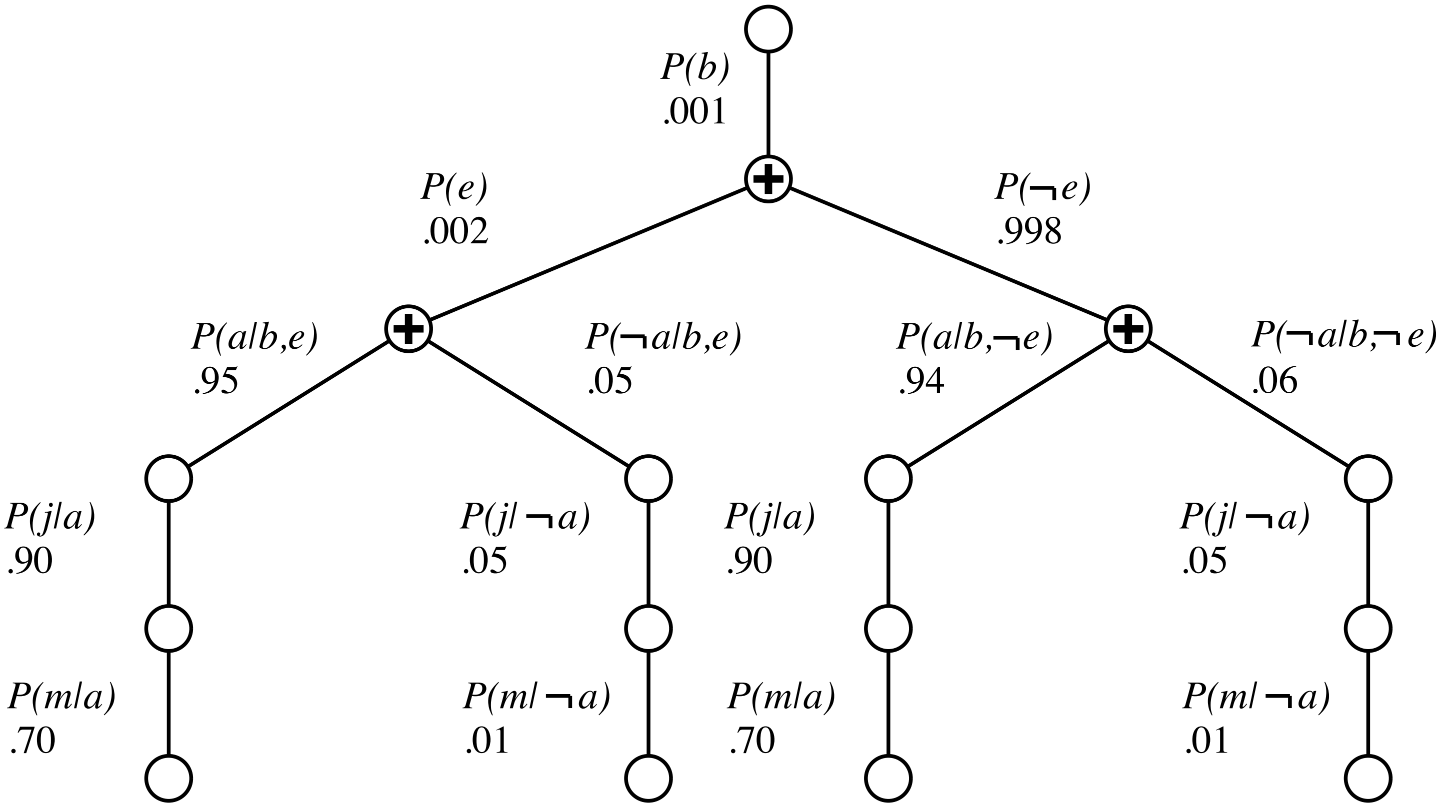


$\mathbf{P}(B|j,m)$
$= \mathbf{P}(B,j,m)/P(j,m)$
$= \alpha\mathbf{P}(B,j,m)$
$= \alpha \sum_e \sum_a \mathbf{P}(B,e,a,j,m)$

Rewrite full joint entries using product of CPT entries:
$\mathbf{P}(B|j,m)$
$= \alpha \sum_e \sum_a \mathbf{P}(B)P(e)\mathbf{P}(a|B,e)P(j|a)P(m|a)$
$= \alpha\mathbf{P}(B) \sum_e P(e) \sum_a \mathbf{P}(a|B,e)P(j|a)P(m|a)$

Recursive depth-first enumeration: $O(n)$ space, $O(d^n)$ time

Evaluation tree



Enumeration is inefficient: repeated computation
e.g., computes $P(j|a)P(m|a)$ for each value of e

Inference by variable elimination

Variable elimination: carry out summations right-to-left,
storing intermediate results (factors) to avoid recomputation

$$\mathbf{P}(B|j,m)$$
$$= \alpha \underbrace{\mathbf{P}(B)}_{B} \Sigma_e \underbrace{P(e)}_{E} \Sigma_a \underbrace{\mathbf{P}(a|B,e)}_{A} \underbrace{P(j|a)}_{J} \underbrace{P(m|a)}_{M}$$
$$= \alpha \mathbf{P}(B) \Sigma_e P(e) \Sigma_a \mathbf{P}(a|B,e) P(j|a) f_M(a)$$
$$= \alpha \mathbf{P}(B) \Sigma_e P(e) \Sigma_a \mathbf{P}(a|B,e) f_J(a) f_M(a)$$
$$= \alpha \mathbf{P}(B) \Sigma_e P(e) \Sigma_a f_A(a,b,e) f_J(a) f_M(a)$$
$$= \alpha \mathbf{P}(B) \Sigma_e P(e) f_{\bar{A}JM}(b,e) \text{ (sum out } A\text{)}$$
$$= \alpha \mathbf{P}(B) f_{\bar{E}\bar{A}JM}(b) \text{ (sum out } E\text{)}$$
$$= \alpha f_B(b) \times f_{\bar{E}\bar{A}JM}(b)$$

Variable elimination: Basic operations

Summing out a variable from a product of factors:
move any constant factors outside the summation
add up submatrices in pointwise product of remaining factors

$\sum_x f_1 \times \cdots \times f_k = f_1 \times \cdots \times f_i \sum_x f_{i+1} \times \cdots \times f_k = f_1 \times \cdots \times f_i \times f_{\bar{X}}$

assuming $f_1, \ldots, f_i$ do not depend on X

Pointwise product of factors f_1 and f_2:

$$f_1(x_1, \ldots, x_j, y_1, \ldots, y_k) \times f_2(y_1, \ldots, y_k, z_1, \ldots, z_l)$$
$$= f(x_1, \ldots, x_j, y_1, \ldots, y_k, z_1, \ldots, z_l)$$

E.g., $f_1(a, b) \times f_2(b, c) = f(a, b, c)$

Summing Out A Variable From a Factor

a^1	b^1	c^1	0.25
a^1	b^1	c^2	0.35
a^1	b^2	c^1	0.08
a^1	b^2	c^2	0.16
a^2	b^1	c^1	0.05
a^2	b^1	c^2	0.07
a^2	b^2	c^1	0
a^2	b^2	c^2	0
a^3	b^1	c^1	0.15
a^3	b^1	c^2	0.21
a^3	b^2	c^1	0.09
a^3	b^2	c^2	0.18

a^1	c^1	0.33
a^1	c^2	0.51
a^2	c^1	0.05
a^2	c^2	0.07
a^3	c^1	0.24
a^3	c^2	0.39

Factor Product

a^1	b^1	0.5
a^1	b^2	0.8
a^2	b^1	0.1
a^2	b^2	0
a^3	b^1	0.3
a^3	b^2	0.9

b^1	c^1	0.5
b^1	c^2	0.7
b^2	c^1	0.1
b^2	c^2	0.2

a^1	b^1	c^1	$0.5 \cdot 0.5 = 0.25$
a^1	b^1	c^2	$0.5 \cdot 0.7 = 0.35$
a^1	b^2	c^1	$0.8 \cdot 0.1 = 0.08$
a^1	b^2	c^2	$0.8 \cdot 0.2 = 0.16$
a^2	b^1	c^1	$0.1 \cdot 0.5 = 0.05$
a^2	b^1	c^2	$0.1 \cdot 0.7 = 0.07$
a^2	b^2	c^1	$0 \cdot 0.1 = 0$
a^2	b^2	c^2	$0 \cdot 0.2 = 0$
a^3	b^1	c^1	$0.3 \cdot 0.5 = 0.15$
a^3	b^1	c^2	$0.3 \cdot 0.7 = 0.21$
a^3	b^2	c^1	$0.9 \cdot 0.1 = 0.09$
a^3	b^2	c^2	$0.9 \cdot 0.2 = 0.18$

Variable elimination algorithm

```
function Elimination-Ask(X, e, bn) returns a distribution over X
   inputs: X, the query variable
           e, evidence specified as an event
           bn, a belief network specifying joint distribution P(X_1, ..., X_n)

   factors ← [ ]; vars ← Reverse(Vars[bn])
   for each var in vars do
      factors ← [Make-Factor(var, e)|factors]
      if var is a hidden variable then factors ← Sum-Out(var, factors)
   return Normalize(Pointwise-Product(factors))
```

Belief Propagation: Motivation

- What if we want to compute all marginals, not just one?
- Doing variable elimination for each one in turn is inefficient
- Solution: Belief Propagation
  - Same idea as Forward-backward for HMMs

Belief Propagation

- Previously: Forward-backward algorithm
  - Exactly computes posterior marginals P(h_i|V) for chain-structured graphical models (e.g. HMMs)
    - Where V are visible variables
    - h_i is the hidden variable at position I
- Now we will generalize this to arbitrary graphs
  - Bayesian and Markov Networks
  - Arbitrary graph structures (not just chains)
- We'll just describe the algorithms and omit derivations (K+F book has good coverage)

BP: Initial Assumptions

- Pairwise MRF:

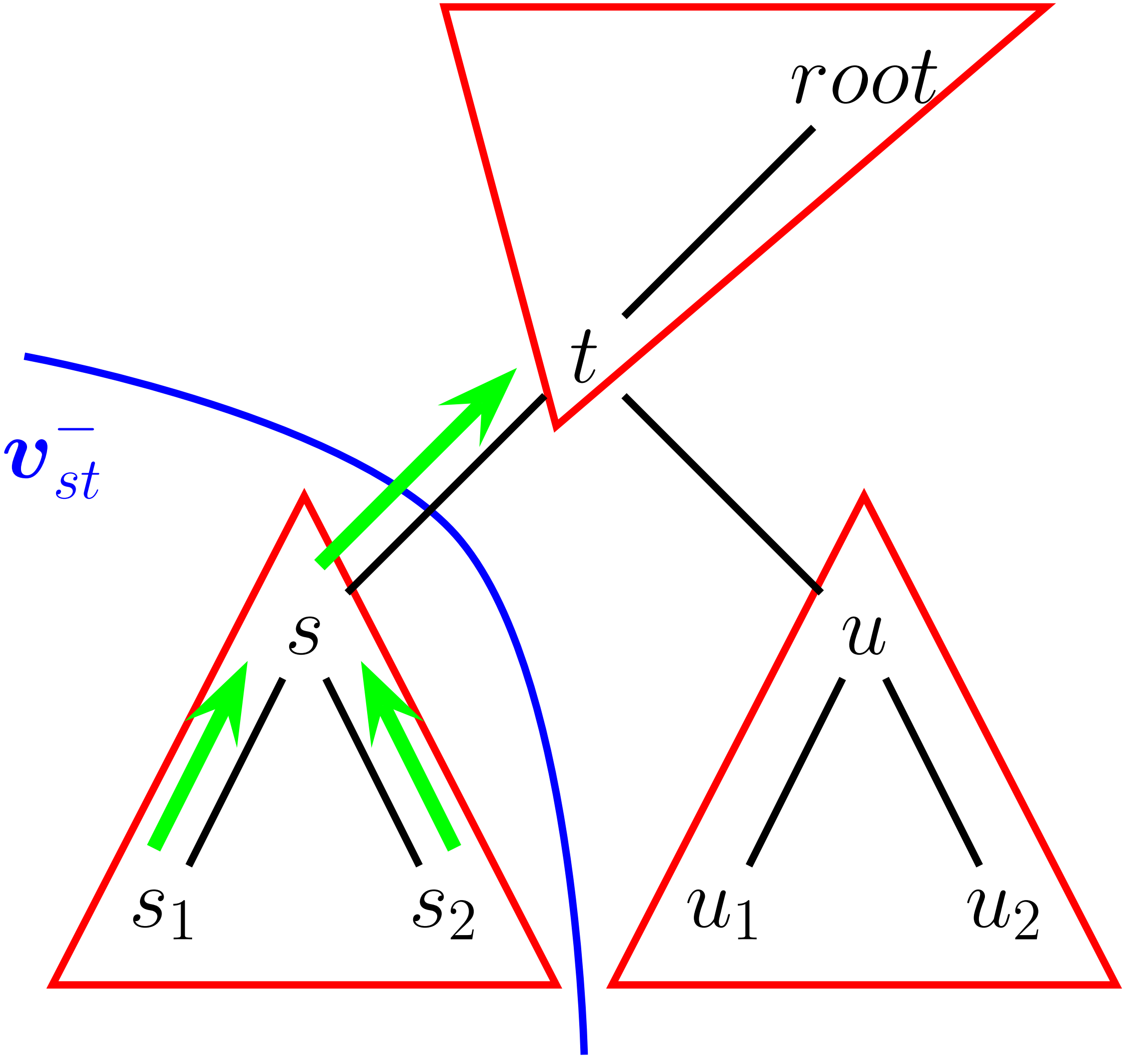
$$P(\mathbf{x}|\mathbf{v}) = \frac{1}{Z(\mathbf{v})} \prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \psi_s(x_s, x_t)$$

- One factor for each variable
- One factor for each edge
- Tree-structure
- models with higher-order cliques later…

Belief Propagation

- Pick an arbitrary node: call it the root
- Orient edges away from root (dangle down)
- Well-defined notion of parent and child
- 2 phases to BP algorithm:
  1. Send messages up to root (collect evidence)
  2. Send messages back down from the root (distribute evidence)
- Generalize forward-backward from chains to trees

Collect to root phase



Collect to root: Details

- Bottom-up belief state: $\mathrm{bel}_t^-(x_t) \equiv p(x_t | \mathbf{v}_t^-)$
  - Probability of x_t given all the evidence at or below node t in the tree
- How to compute the bottom up belief state?
- **“messages”** from t’s children
  - Recursively defined based on belief states of children
  - Summarize what they think t should know about the evidence in their subtrees

$$m_{s \to t}^-(x_t) \equiv p(x_t | \mathbf{v}_{st}^-)$$

Computing the upward belief state

$$\mathrm{bel}_t^-(x_t) \equiv p(x_t|\mathbf{v}_t^-) = \frac{1}{Z_t}\psi_t(x_t) \prod_{c \in ch(t)} m_{c \to t}^-(x_t)$$

- Belief state at node t is the normalized product of:
  - Incoming messages from children
  - Local evidence

Q: how to compute upward messages?

- Assume we have computed belief states of children, then message is:

$$m^{-}_{s\to t}(x_t) = \sum_{x_s} \psi_{st}(x_s, x_t)\mathrm{bel}^{-}_{s}(x_s)$$

- Convert beliefs about child (s) into belifs about parent (t) by using the edge potential

Completing the Upward Pass

- Continue in this way until we reach the root
- Analogous to forward pass in HMM
- Can compute the probability of evidence as a side effect

Can now pass messages down from root

Computing the belief state for node s

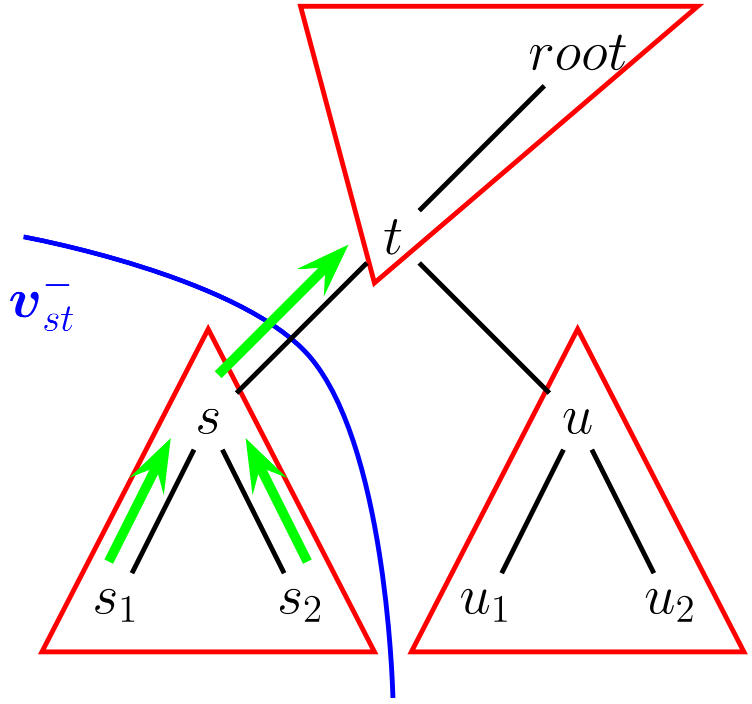
$$\text{bel}_s(x_s) \equiv p(x_s|\mathbf{v})$$

- Combine the bottom-up belief for node s with a top-down message for t
  - Top-down message summarizes all the information in the rest of the graph:

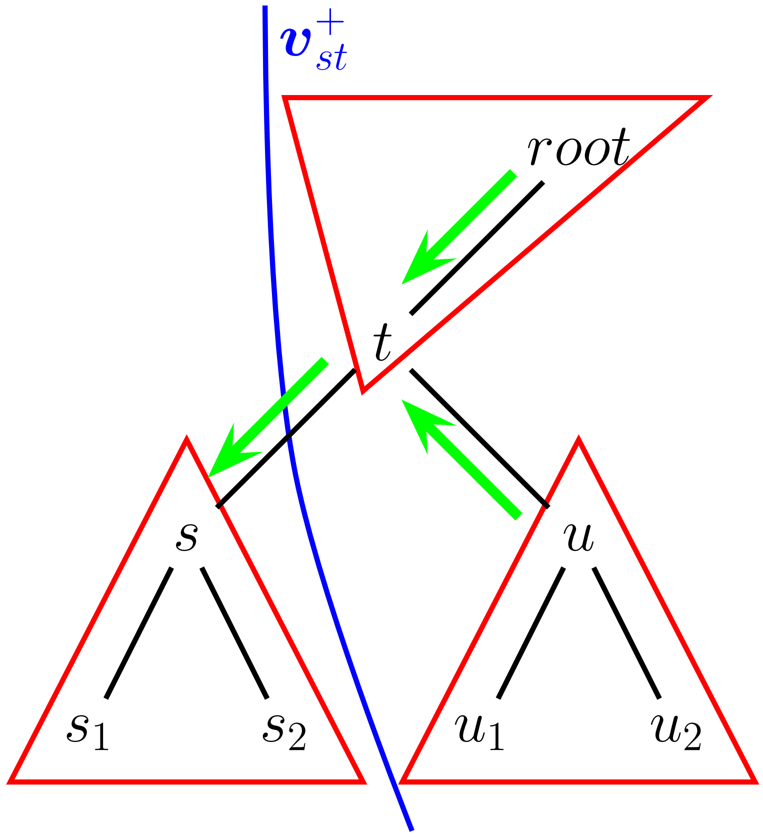
$$m^{+}_{t\rightarrow s}(x_s) \equiv p(x_t|\mathbf{v}^{+}_{st})$$

  - v_st+ is all the evidence on the upstream (root) side of the edge s - t

Send to Root



Distribute from Root



Computing Beliefs:

$$\text{bel}_s(x_s) \equiv p(x_s|\mathbf{v}) \propto \text{bel}_s^-(x_s) \prod_{t \in pa(s)} m_{t \to s}^+(x_s)$$

- Combine bottom-up beliefs with top-down messages

Q: how to compute top-down messages?

- Consider the message from **t** to **s**
- Suppose t’s parent is r
- t’s children are s and u
- (like in the figure)

Q: how to compute top-down messages?

- We want the message to include all the information t has received except information that s sent it

$$m^{+}_{t\to s}(x_s) \equiv p(x_t|\mathbf{v}^{+}_{st}) = \sum_{x_t} \psi(x_s, x_t) \frac{\mathrm{bel}_t(x_t)}{m^{-}_{s\to t}(x_t)}$$

Sum-product algorithm

- Really just the same thing
- Rather than dividing, plug in the definition of node t's belief to get:

$$m^+_{t\to s}(x_s) = \sum_{x_t} \psi_{st}(x_s, x_t)\psi_t(x_t) \prod_{c\in ch(t), c\neq s} m^-_{c\to t}(x_t) \prod_{p\in pa(t)} m^+_{p\to t}(x_t)$$

- Multiply together all messages coming into t
  - except message recipient node (s)

Parallel BP

- So far we described the "serial" version
  - This is optimal for tree-structured GMs
  - Natural extension of forward-backward
- Can also do in parallel
  - All nodes receive messages from their neighbors in parallel
  - Initialize messages to all 1's
  - Each node absorbs messages from all it's neighbors
  - Each node sends messages to each of it's neighbors
- Converges to the correct posterior marginal

Loopy BP

- Approach to "approximate inference"
- BP is only guaranteed to give the correct answer on tree-structured graphs
- But, can run it on graphs with loops, and it gives an approximate answer
  - Sometimes doesn't converge

Generalized Distributive Law

- Abstractly VE can be thought of as computing the following expression:

$$P(\mathbf{x}_q|\mathbf{x}_v) \propto \sum_{\mathbf{x}} \prod_{c} \psi_c(\mathbf{x}_c)$$

  - Where visible variables are clamped and not summed over
  - Intermediate results are cached and not re-computed

Generalized Distributive Law

- Other important task: **MAP inference**

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \prod_c \psi_c(\mathbf{x}_c)$$

  - Essentially the same algorithm can be used
  - Just replace sum with max (also traceback step)

Generalized Distributive Law

- In general VE can be applied to any commutative **semi-ring**
  - A set K, together with two binary operations called "+" and "×" which satisfy the axioms:
    - The operation "+" is associative and commutative
    - There is an additive identity "0"
      - $k + 0 = k$
    - The operation "×" is associative and commutative
    - There is a multiplicative identity "1"
      - $k \times 1 = k$
    - The distributive law holds:
      - $(a \times b) + (a \times c) = a \times (b + c)$

Generalized Distributive Law

- **Semi-ring** For marginal inference (sum-product):
  - "×" = multiplication
  - "+" = sum
- **Semi-ring** For MAP inference (max-product):
  - "×" = multiplication
  - "+" = max