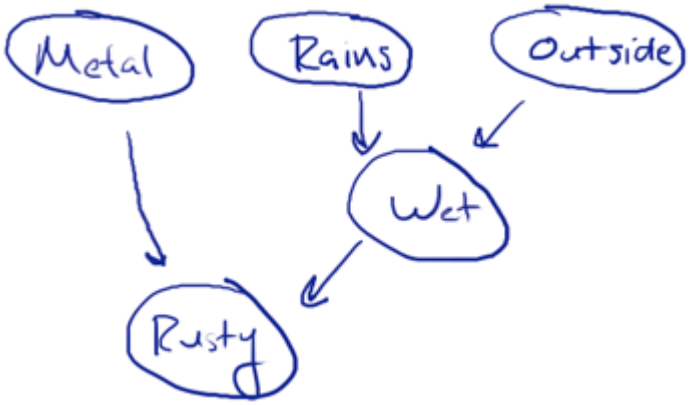
Graphical models

Review

$$\mathbb{P}[(x \vee y \vee \bar{z}) \wedge (\bar{y} \vee \bar{u}) \wedge (z \vee w) \wedge (z \vee u \vee v)]$$

- Dynamic programming on graphs
  - variable elimination example

- Graphical model = graph + model
  - e.g., Bayes net: DAG + CPTs
  - e.g., rusty robot

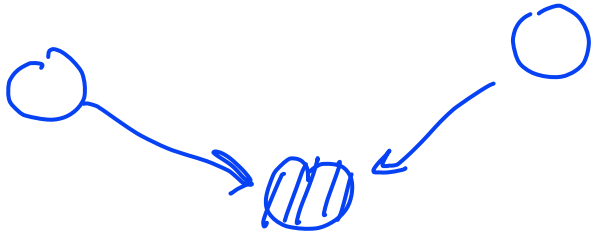


- Benefits:
  - fewer parameters, faster inference
  - some properties (e.g., some conditional independences) depend only on graph

Review

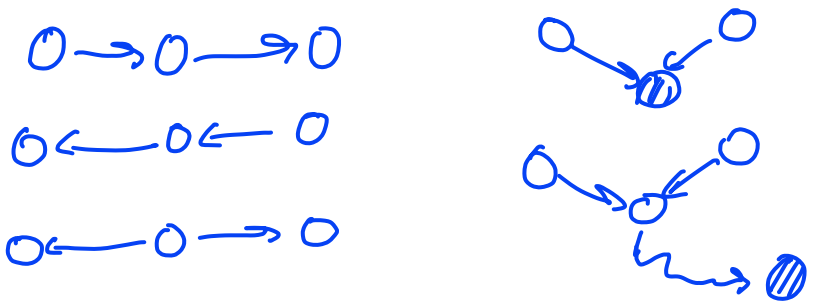
- Blocking

- Explaining away



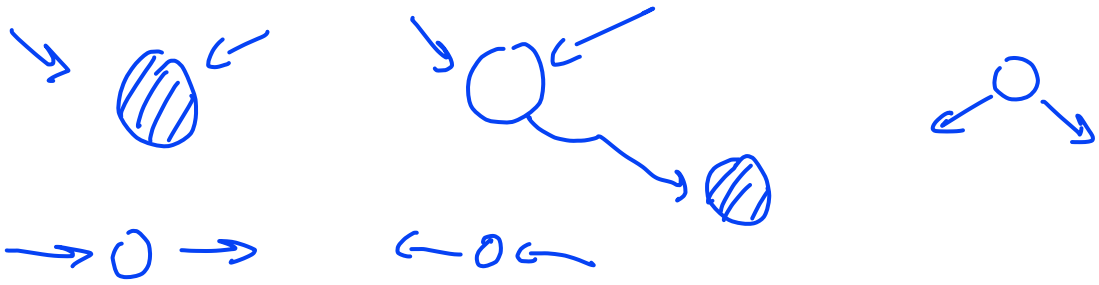
d-separation

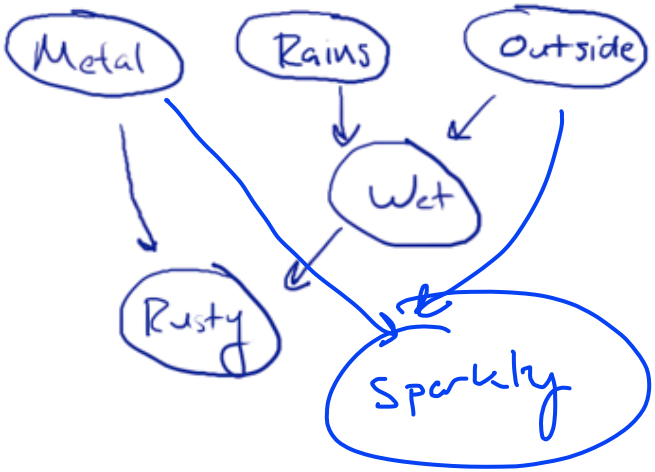
- General graphical test: "d-separation"
  - d = dependence
- $X \perp Y \mid Z$ when there are no active paths between X and Y given Z
  - activity of path depends on conditioning variable/set Z
- Active paths of length 3 ($W \notin$ conditioning set):



Longer paths

- Node X is active (wrt path P) if:





  and inactive o/w

- (Undirected) path is active if ***all*** intermediate nodes are active

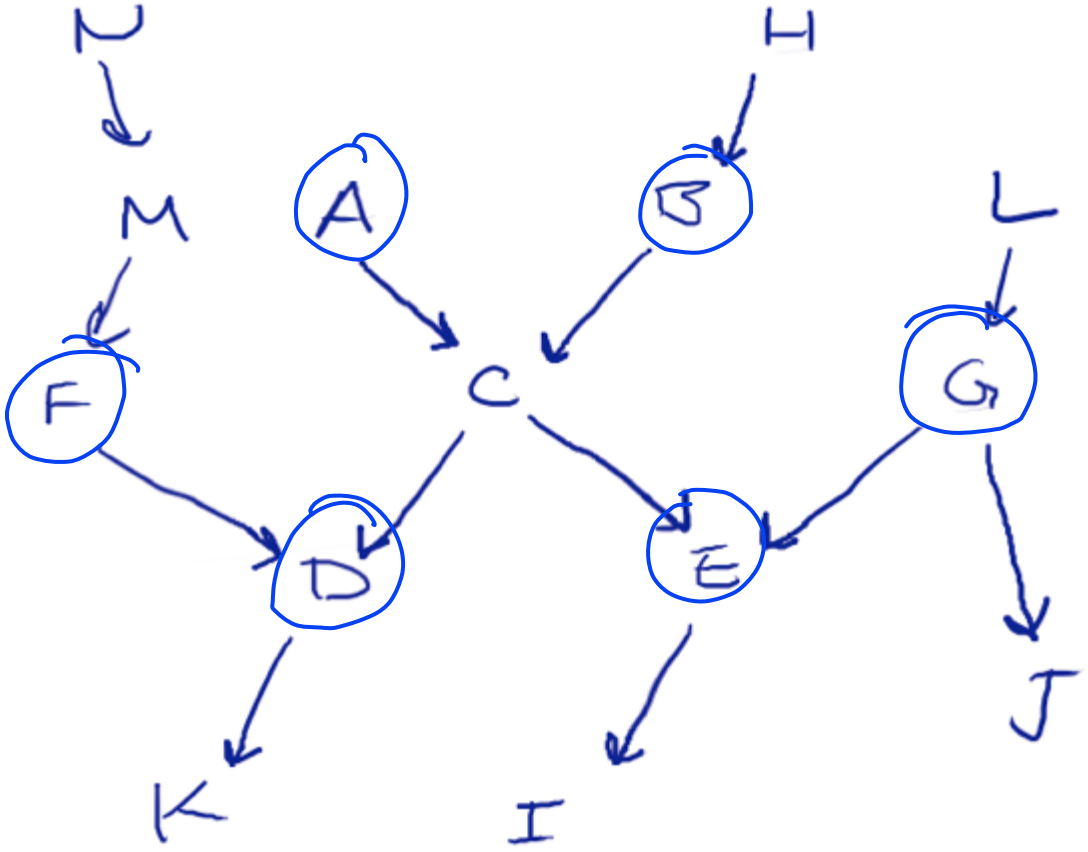
Algorithm: $X \perp Y \mid \{Z_1, Z_2, \ldots\}$?

- For each Z_i:
  - mark self and ancestors by traversing parent links
- Breadth-first search starting from X
  - traverse edges only if they can be part of an active path
    - use "ancestor of shaded" marks to test activity
  - prune when we visit a node for the second time from the same direction (from children or from parents)
- If we reach Y, then X and Y are dependent given $\{Z_1, Z_2, \ldots\}$ — else, conditionally independent

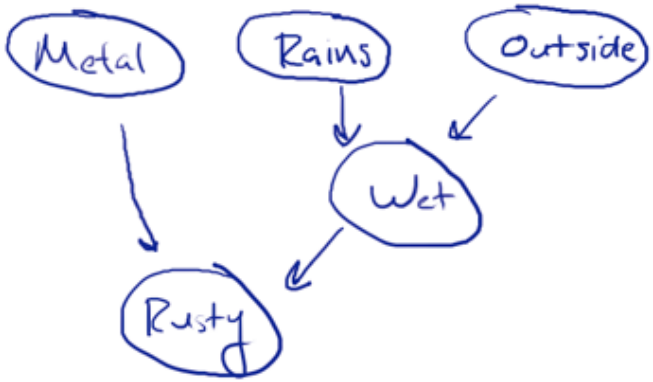
Markov blanket

- Markov blanket of C = minimal set of obs'ns to make C independent of rest of graph

parents
children
co-parents



Learning fully-observed Bayes nets



P(M) = 3/5

P(Ra) = 2/5

P(O) = 4/5

P(W | Ra, O) =

Ra O
F F
F T 1/2
T F
T T
0%!

$\frac{0+\lambda}{0+\lambda+1+\lambda} = \frac{\lambda}{1+2\lambda}$ Laplace smoothing to fit

$\frac{0+\lambda}{0+\lambda+0+\lambda}$

P(Ru | M,W) =

M	Ra	O	W	Ru
T	F	T	T	F
T	T	T	T	T
F	T	T	F	F
T	F	F	F	T
F	F	T	F	T

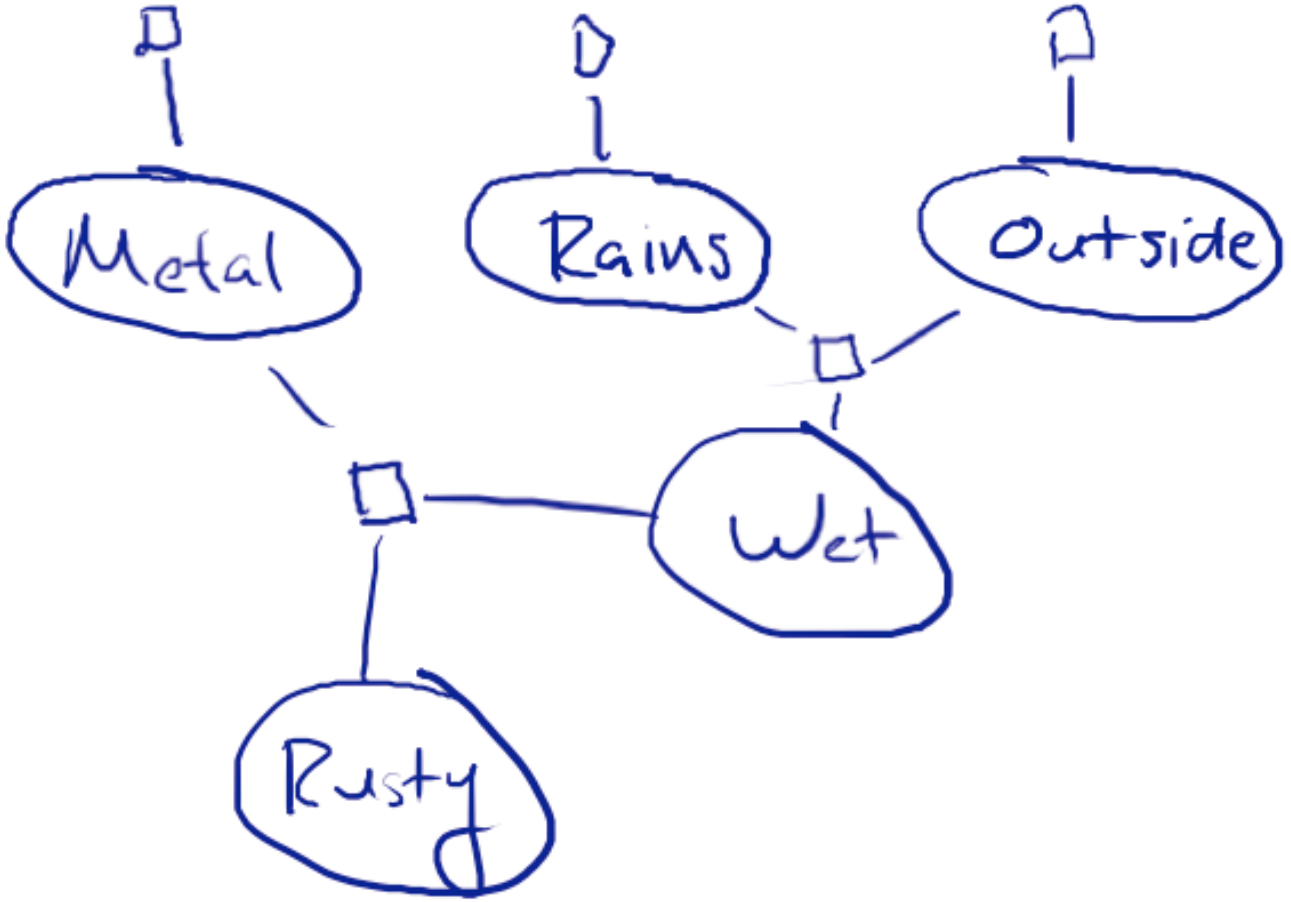
Limitations of counting

- Works ***only*** when all variables are observed in all examples
- If there are ***hidden*** or ***latent*** variables, more complicated algorithm (expectation-maximization or spectral)
  - or use a toolbox!

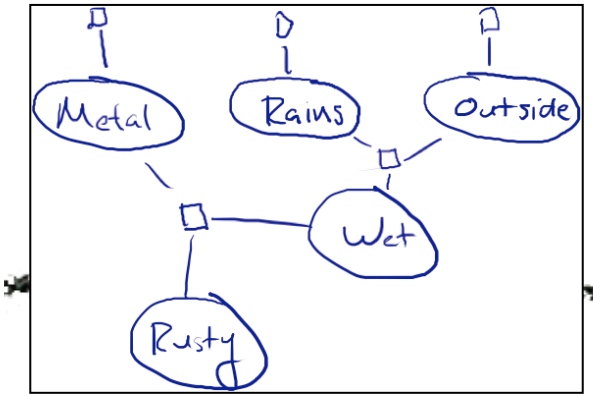
Factor graphs

- Another common type of graphical model
- ***Undirected, bipartite*** graph instead of DAG
- Like Bayes net:
  - can represent any distribution
  - can infer conditional independences from graph structure
  - but some distributions have more faithful representations in one formalism or the other

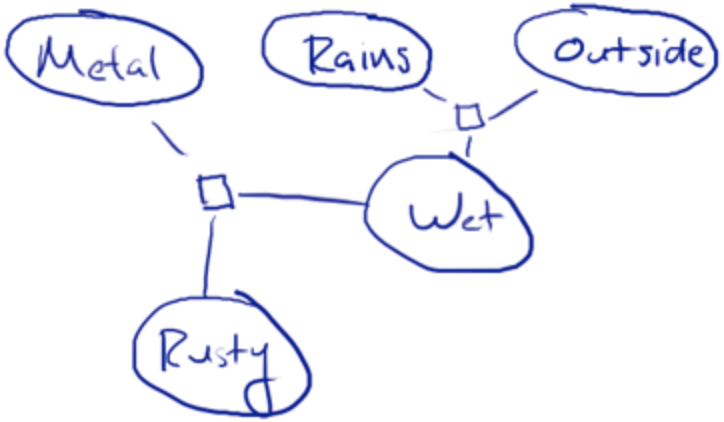
Rusty robot: factor graph

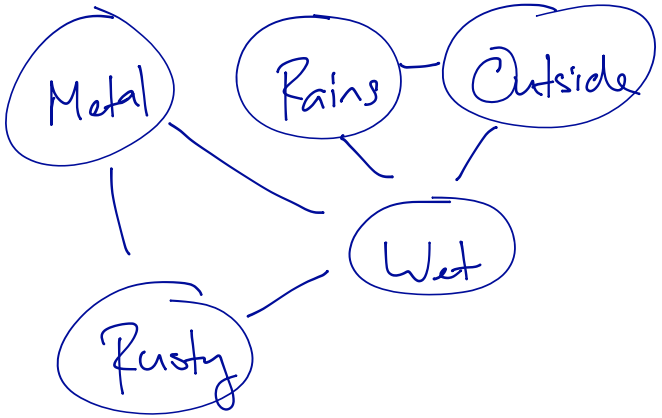


P(M) P(Ra) P(O) P(W|Ra,O) P(Ru|M,W)



Conventions



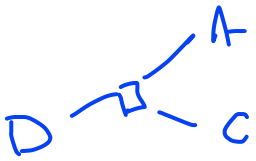


Markov random field

- Don't need to show unary factors—why?
  - can usually be collapsed into other factors
  - don't affect structure of dynamic programming
- Show factors as cliques

Non-CPT factors

- Just saw: easy to convert Bayes net → factor graph
- In general, factors need not be CPTs: any nonnegative #s allowed
  - higher # → this combination more likely



- In general, $P(A, B, \ldots) = \prod_i \phi_i(x_{\{i\}}) / Z$

$\{A, C, D\}$

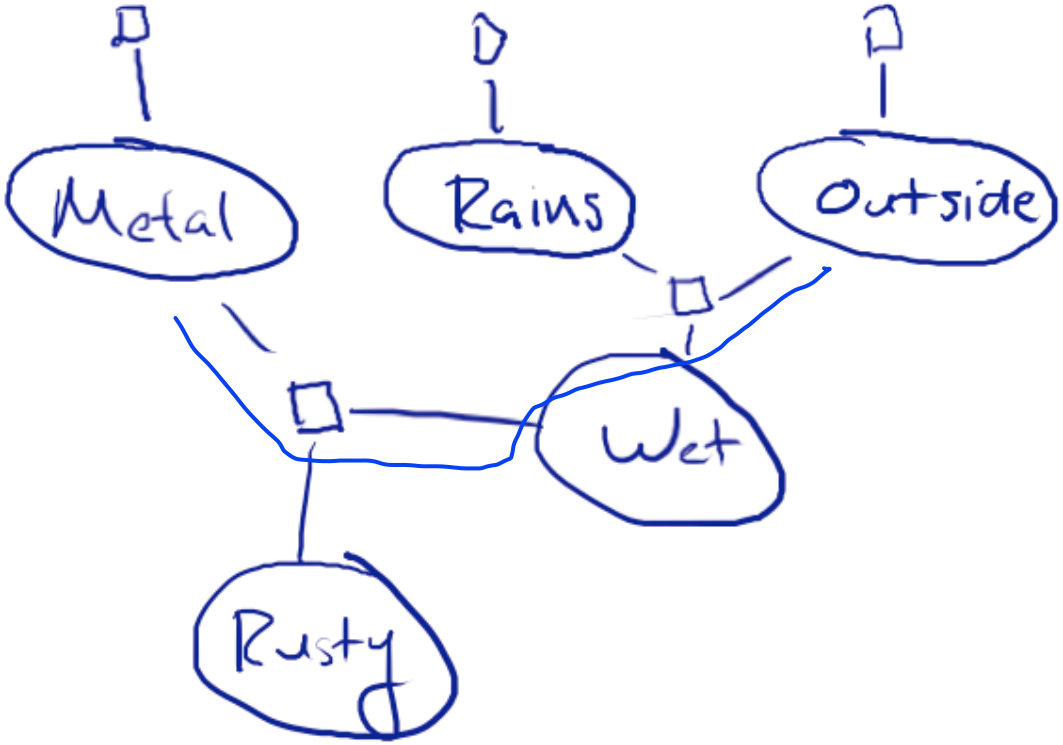
- $Z = \sum_X \prod_i \phi_i(x_{\{i\}})$

$\hookrightarrow \sum_A \sum_B \sum_C \cdots$

Independence

- Just like Bayes nets, there are graphical tests for independence and conditional independence
- Simpler, though:
  - Cover up all observed nodes
  - Look for a path

Independence example



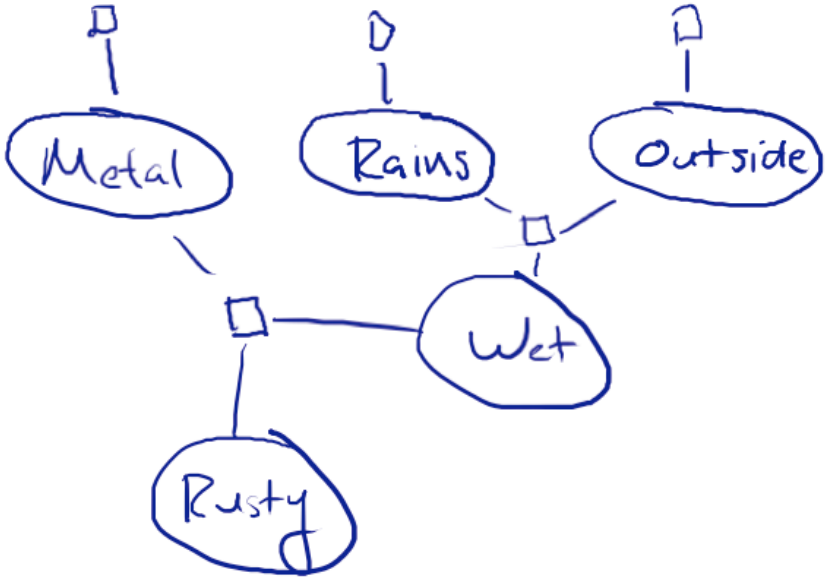
What gives?

- Take a Bayes net, list (conditional) independences
- Convert to a factor graph, list (conditional) independences
- Are they the same list?
- What happened?

accidental indep.

Inference: same kind of DP as before

$$P(M, Ra, O, W, Ru) = \phi_1(M)\,\phi_2(Ra)\,\phi_3(O)\,\phi_4(Ra, O, W)\,\phi_5(M, W, Ru)/Z$$



$\phi_1(M)$: T 0.9, F 0.1

$\phi_2(Ra)$: T 0.7, F 0.3

$\phi_3(O)$: T 0.2, F 0.8

$\phi_4(Ra, O, W)$:

TTT	0.9
TTF	0.1
TFT	0.1
TFF	0.9
FTT	0.1
FTF	0.9
FFT	0.1
FFF	0.9

$\phi_5(M, W, Ru)$:

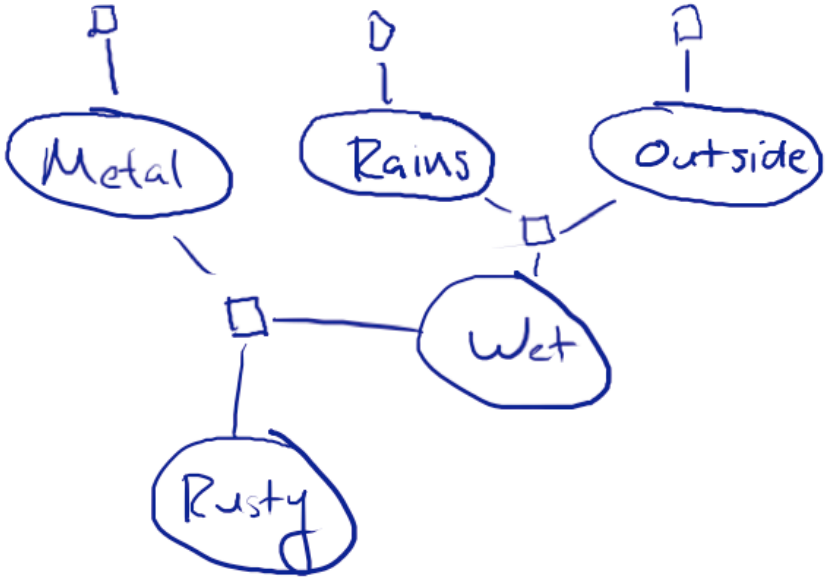
TTT	0.8
TTF	0.2
TFT	0.1
TFF	0.9
FTT	0
FTF	1
FFT	0
FFF	1

- Typical Q: given Ra=F, Ru=T, what is P(W)?

Incorporate evidence

$P(M, O, W \mid Ra = F, Ru = T)$

$P(M, Ra, O, W, Ru) = \phi_1(M)\phi_2(Ra)\phi_3(O)\phi_4(Ra, O, W)\phi_5(M, W, Ru)/Z$



$\phi_1(M)$: T 0.9, F 0.1

$\phi_2(Ra)$: T 0.7, F 0.3

$\phi_3(O)$: T 0.2, F 0.8

$\phi_4(Ra, O, W)$:

TTT 0.9
TTF 0.1
TFT 0.1
TFF 0.9
FTT 0.1
FTF 0.9
FFT 0.1
FFF 0.9

$\phi_5(M, W, Ru)$:

TTT 0.8
TTF 0.2
TFT 0.1
TFF 0.9
FTT 0
FTF 1
FFT 0
FFF 1

Condition on Ra=F, Ru=T

Eliminate nuisance nodes

$$P(M, O, W \mid R_A = T, R_M = F) = \phi_1(M)\,\phi_2(R_A)\,\phi_3(O)\,\phi_4(R_A, O, W)\,\phi_5(M, W, R_M)\,/\,Z$$

- Remaining nodes: M, O, W
- Query: P(W) | stuff
- So, O&M are nuisance—marginalize away
- Marginal = $\sum_O \sum_M \phi_1(M)\,\phi_3(O)\,\phi_4(O, W)\,\phi_5(M, W)\,/\,Z$

Elimination order

$$\sum_M \sum_O \phi_1(M)\phi_3(O)\phi_4(O,\omega)\phi_5(M,\omega)/Z$$

- Sum out nuisance variables in turn
- Can do it in any order, but some orders may be easier than others—do O then M

$\phi_3(O)$: T 0.2, F 0.8

$\phi_4(\cancel{R_4}, O, \omega)$:

~~T~~	T	0.1
~~T~~	F	0.9
~~F~~	T	0.1
~~F~~	F	0.9

$\phi_1(M)$: T 0.9, F 0.1

$\phi_6(\omega)$: T 0.1, F 0.9

$\phi_5(M, \omega, \cancel{R_5})$:

T	T	0.8
T	F	0.1
F	T	0
F	F	0

$\phi_7(\omega)$ =

T: $0.1 \times 0.9 \times 0.8 + 0.1 \times 0.1 \times 0 = .072$

F: $0.9 \times 0.9 \times 0.1 + 0.9 \times 0.1 \times 0 = .081$

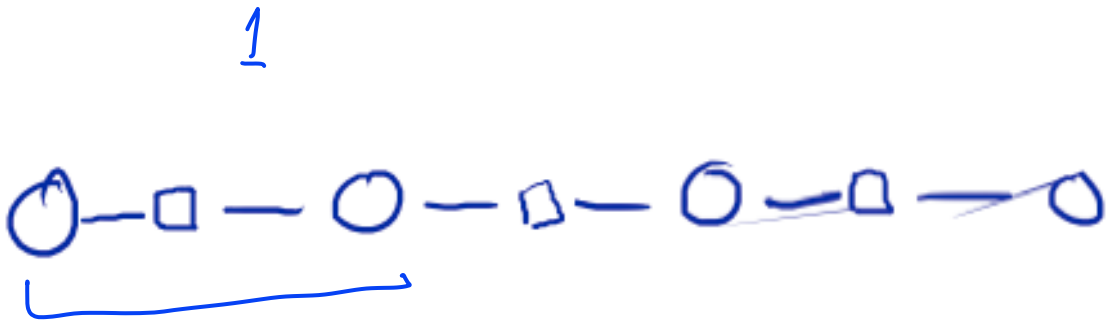
$$P(\omega \mid R_9 = T, R_4 = F) = \frac{.072}{.072 + .081} = 8/17$$

Discussion

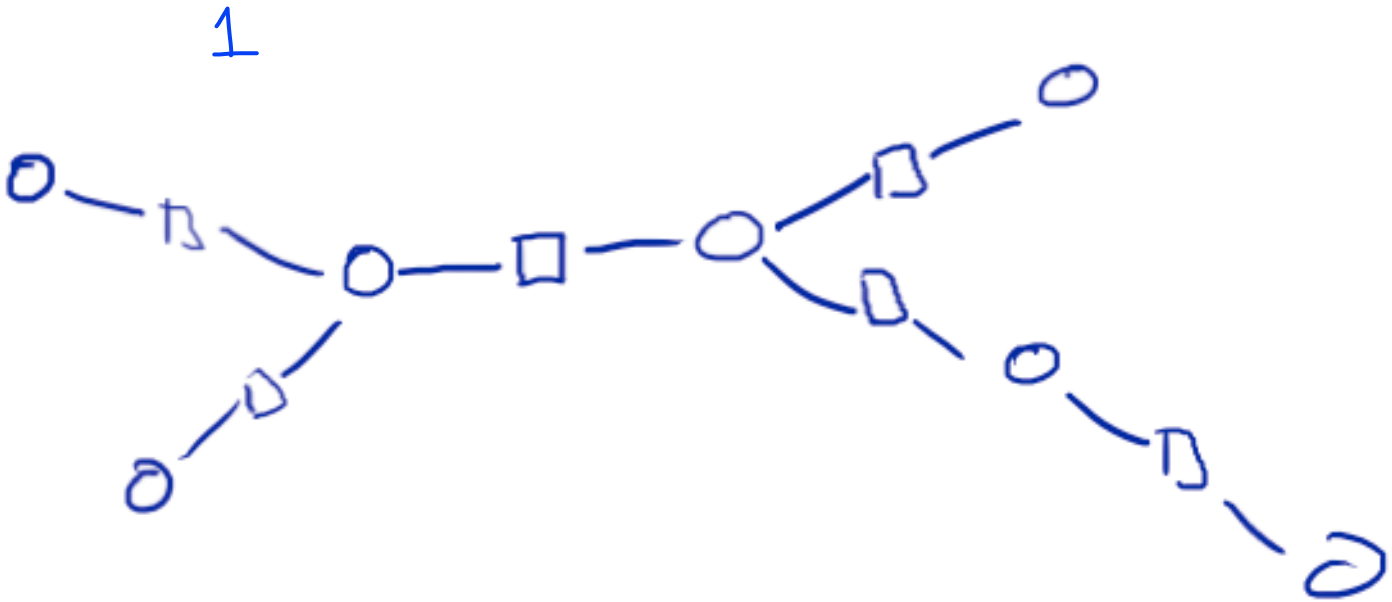
- Directed v. undirected: advantages to both
- Normalization
- Each elimination introduces a new table (all current neighbors of eliminated variable), makes some old tables irrelevant
- Each elim. order introduces different tables
- Some tables bigger than others
  - FLOP count; treewidth

Treewidth examples

Chain



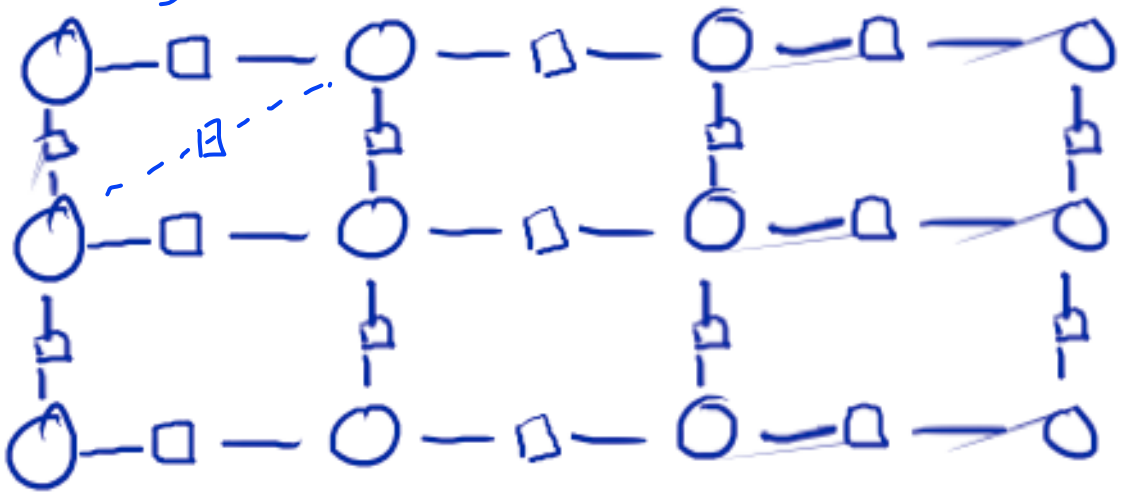
Tree



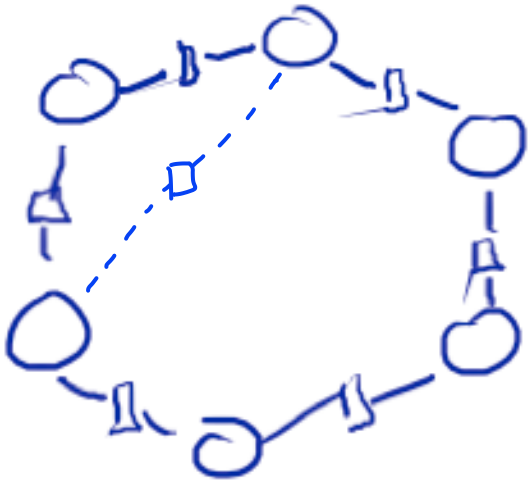
Treewidth examples

3 (# chains)

Parallel chains



Cycle 2



Inference in general models

- Prior + evidence → (marginals of) posterior
  - several examples so far, but no general algorithm
- General algorithm: ***message passing***
  - aka ***belief propagation***
  - build a *junction tree*, instantiate evidence, pass messages (*calibrate*), read off answer, eliminate nuisance variables
- Share work of building JT among multiple queries
  - there are many possible JTs; different ones are better for different queries, so might want to build several

Better than variable elimination

- Suppose we want all 1-variable marginals
  - Could do N runs of variable elimination
  - Or: BP simulates N runs for the price of 2
- Further reading: Kschischang et al., "Factor Graphs and the Sum-Product Algorithm"

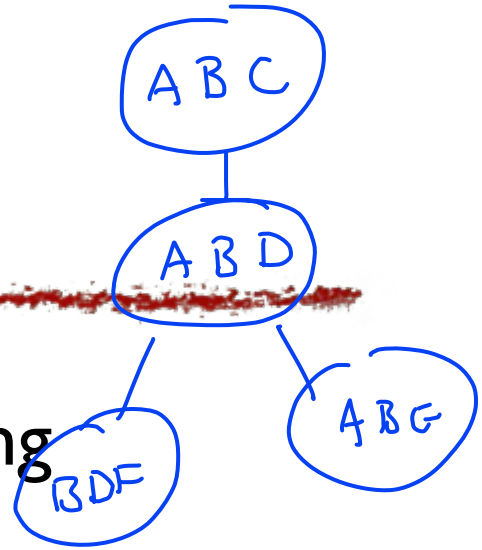
  www.comm.utoronto.ca/frank/papers/KFL01.pdf
- Or, Daphne Koller's book

What you need to understand

- How expensive will inference be?
  - what tables will be built and how big are they?
- What does a message represent and why?

Junction tree

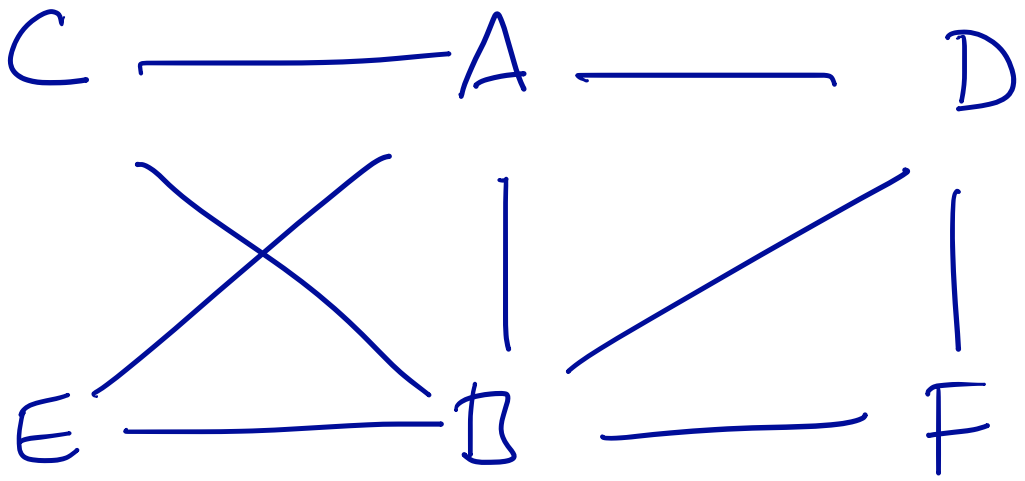
(aka clique tree, aka join tree)



- Represents the tables that we build during elimination
  - many JTs for each graphical model
  - many-to-many correspondence w/ elimination orders
- A junction tree for a model is:
  - a tree
  - whose nodes are sets of variables ("cliques")
  - that contains a node for each of our factors
  - that satisfies ***running intersection property*** (below)

factor ⊆ clique

Example network



- Elimination order: CEABDF
- Factors: ABC, ABE, ABD, BDF

Building a junction tree

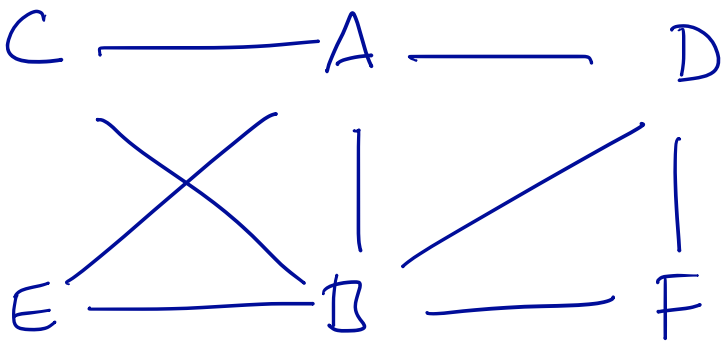
(given an elimination order)

- $S_0 \leftarrow \varnothing$, $V \leftarrow \varnothing$ *[S = table args; V = visited]*
- For i = 1…n: *[elimination order]*
  - $T_i \leftarrow S_{i-1} \cup (\text{nbr}(X_i) \setminus V)$ *[extend table to unvisited nbrs]*
  - $S_i \leftarrow T_i \setminus \{X_i\}$ *[marginalize out X_i]*
  - $V \leftarrow V \cup \{X_i\}$ *[mark X_i visited]*
- Build a junction tree from values S_i, T_i:
  - nodes: local maxima of T_i ($T_i \not\subseteq T_j$ for $j \neq i$)
  - edges: local minima of S_i (after a run of marginalizations without adding new nodes)

Example

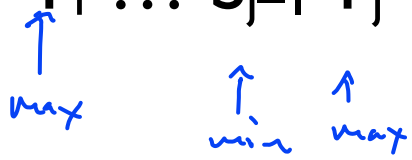
CEABDF

	S	T	V
0	∅		
1	AB	ABC	C
2	AB	ABG	CE
3	BD	ABD	CEA
4	DF	BDF	CEAB
5	F	DF	
6	∅	F	



Edges, cont'd

- Pattern: $T_i \ldots S_{j-1} T_j \ldots S_{k-1} T_k \ldots$



- Pair each T with its following S (e.g., T_i w/ S_{j-1})
- Can connect T_i to T_k iff $k>i$ and $S_{j-1} \subseteq T_k$
- Subject to this constraint, free to choose edges
  - always OK to connect in a line, but may be able to skip

Running intersection property

- Once a node X is added to T, it stays in T until eliminated, then never appears again
- In JT, this means all sets containing X form a connected region of tree
  - true for all X = running intersection property

Moralize & triangulate