CMT311 Principles of Machine Learning

GM Inference (part 2)

Angelika Kimmig
KimmigA@cardiff.ac.uk

29.11.2019

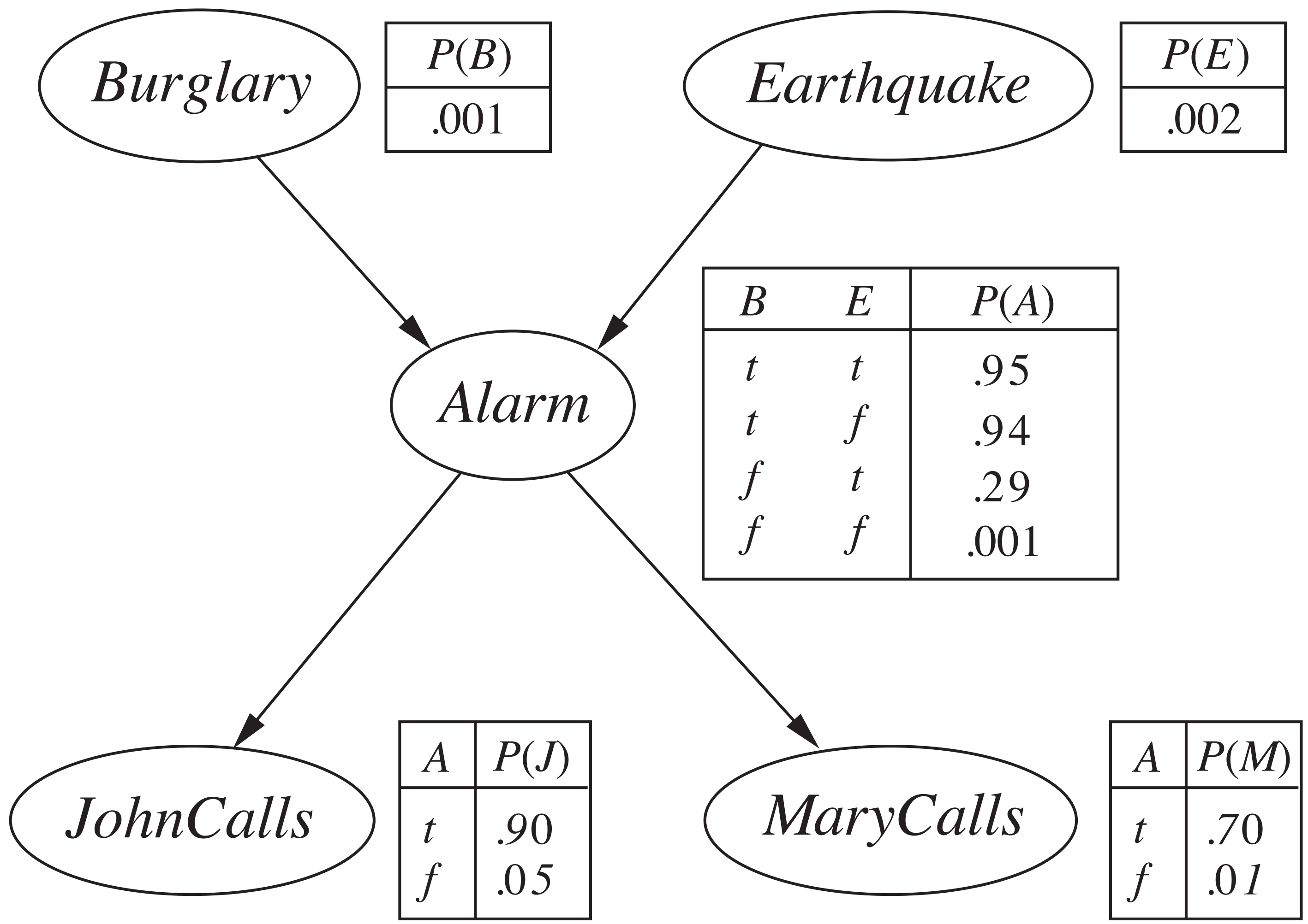
Coursework

- 30% of module marks
- now available on Learning Central under “Assessment”
- **STRICT deadline: 28.02.2020, 9:30am**
- You can **update** your submission as often as you want. Only your **last submission counts**: its timestamp determines whether you submitted on time, and only that submission will be marked.
- **Questions**: **only** via the “Coursework” forum on Learning Central (under “Discussions”)

Last week

Judea Pearl's Alarm network

You have a new burglary alarm that is fairly reliable at detecting a burglary, but also responds to earthquakes. Your neighbours, Mary and John, promise to call you if they hear the alarm sounding.



[figure: Russell& Norvig]

Variable Elimination

$$P(B,J=t,M=t) = \underbrace{\underbrace{P(B)}_{f_B(B)} \underbrace{\sum_E \underbrace{P(E)}_{f_E(E)} \underbrace{\sum_A \underbrace{P(A\,|\,E,B)}_{f_A(A,B,E)} \underbrace{P(J=t\,|\,A)}_{f_J(A)} \underbrace{P(M=t\,|\,A)}_{f_M(A)}}_{f_{\overline{A}JM}(B,E)}}_{f_{\overline{E}\overline{A}JM}(B)}}_{P(B,J=t,M=t)}$$

$$f_{\overline{A}JM}(B,E) = \sum_a f_A(A=a,B,E) \times f_J(A=a) \times f_M(A=a)$$

$$= f_A(A=t,B,E) \times f_J(A=t) \times f_M(A=t) + f_A(A=f,B,E) \times f_J(A=f) \times f_M(A=f)$$

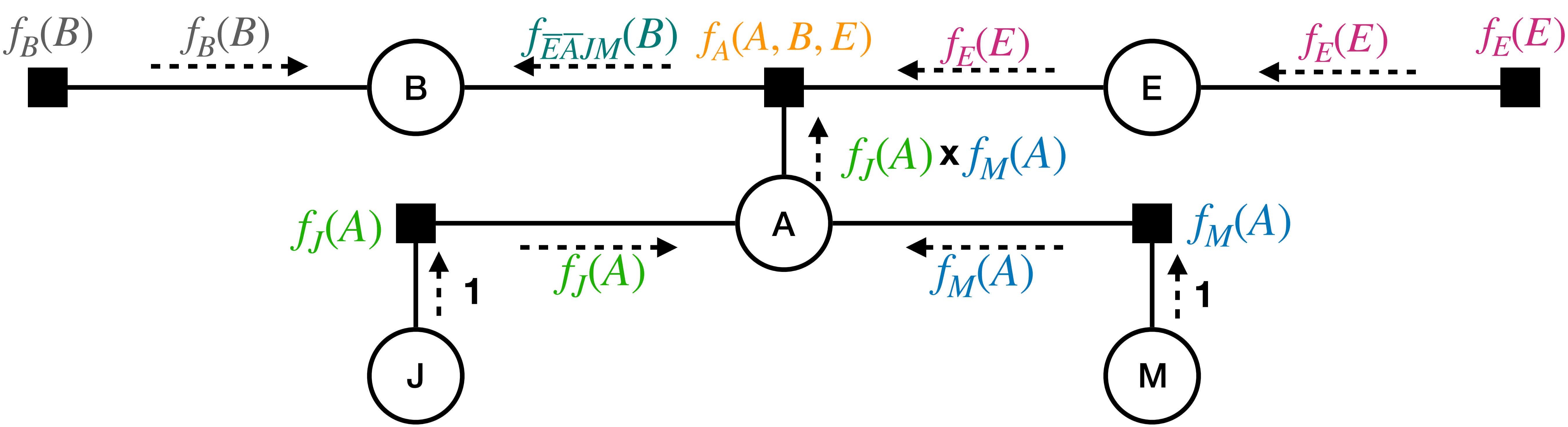
$$f_{\overline{E}\overline{A}JM}(B) = f_E(E=t) \times f_{\overline{A}JM}(B,E=t) + f_E(E=f) \times f_{\overline{A}JM}(B,E=f)$$

$$P(B,J=t,M=t) = f_B(B) \times f_{\overline{E}\overline{A}JM}(B)$$

Variable Elimination as Message Passing

$$P(B, J=t, M=t) = \underbrace{P(B)}_{f_B(B)} \underbrace{\sum_E \underbrace{P(E)}_{f_E(E)} \sum_A \underbrace{P(A \mid E, B)}_{f_A(A,B,E)} \underbrace{P(J=t \mid A)}_{f_J(A)} \underbrace{P(M=t \mid A)}_{f_M(A)}}_{f_{\overline{E}\overline{A}JM}(B)}$$

$$P(B, J=t, M=t) = f_B(B) \times f_{\overline{E}\overline{A}JM}(B)$$



Sum-product algorithm

- FGs can have **loops**, which cause a problem for message passing: eliminating a variable may introduce a factor that isn't in the graph yet
- what to do?
  - option 1: **loopy belief propagation**
    - use propagation rules anyways, hoping that messages will converge
    - no guarantees, but often works well enough in practice
  - option 2: **bucket elimination**
    - guaranteed to produce correct answers

Bucket Elimination

- computes the marginal of one variable only
- multi-variable messages need storage exponential in number of their variables
- for graphs without loops, computational complexity depends on ordering: there is an order with linear complexity, but others are much worse

Today

- Loopy structures revisited: junction trees
- Approximate inference using sampling

The cost of BN inference

- Time and space requirements of variable elimination are dominated by the size of the largest factor constructed.

- If the DAG of a BN is **singly connected** (no loops), then the time and space complexity of exact inference is **linear** in the size of the network (=number of parameters).

    - If the number of parents per node is bounded, also linear in the number of variables.

- If the DAG has **loops**, VE can take **exponential** time and space, even for bounded number of parents.

The cost of BN inference

- As inference in propositional logic is a special case of BN inference, BN inference is (at least) NP-hard.

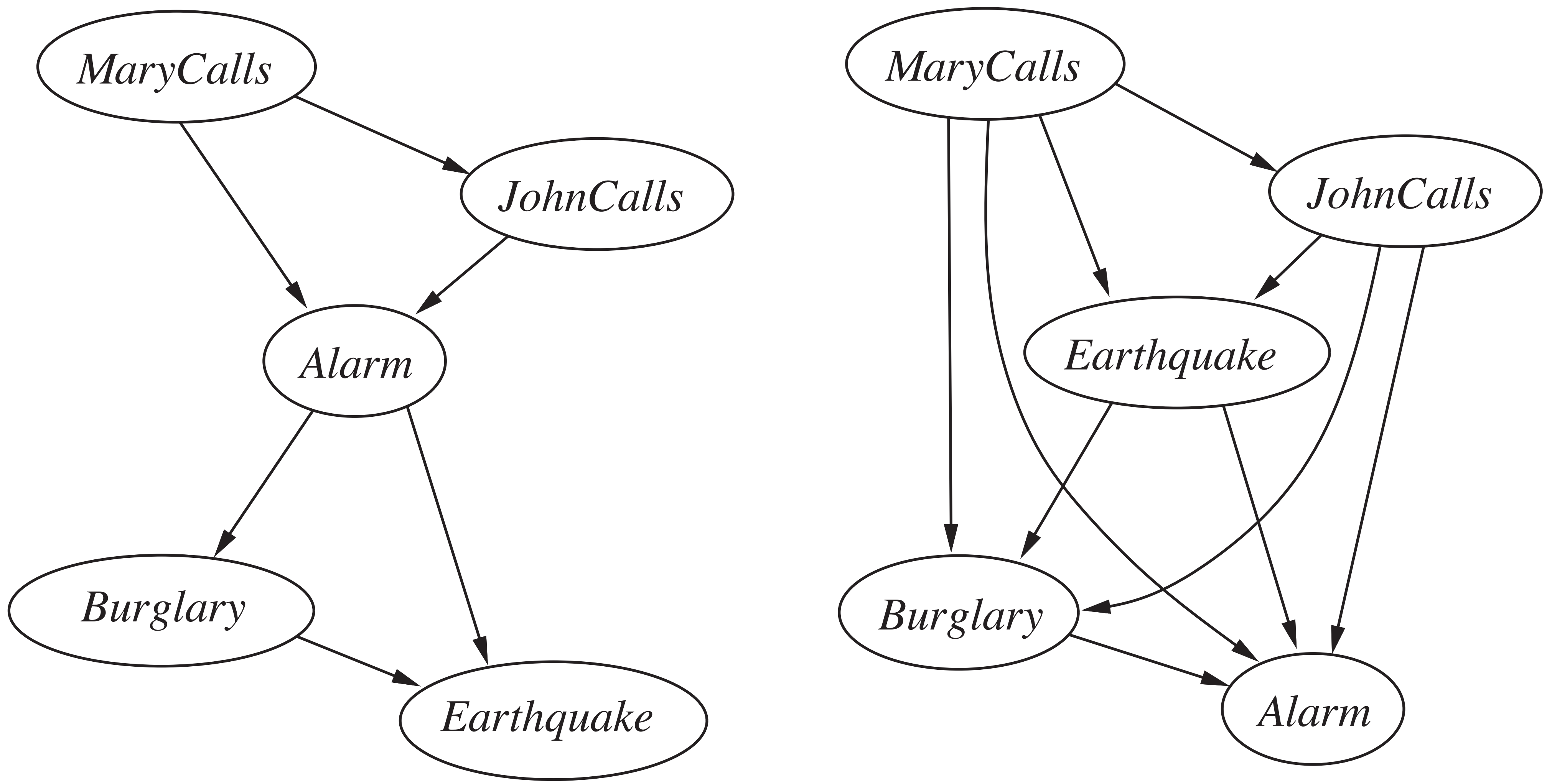
- More precisely, inference in BNs is as hard as computing the number of satisfying assignments for a propositional formula, which is #P-hard (strictly harder than NP-complete problems)

- Yet another reason for careful modelling…

Judea Pearl's Alarm network

order: M,J,A,B,E

order: M,J,E,B,A



(a) (b)

[figure: Russell& Norvig]

The junction tree algorithm

- Loops cannot always be avoided.

- Bucket elimination handles loops, but needs a separate run for each variable's marginal.

- The **junction tree algorithm** computes marginals for all variables:

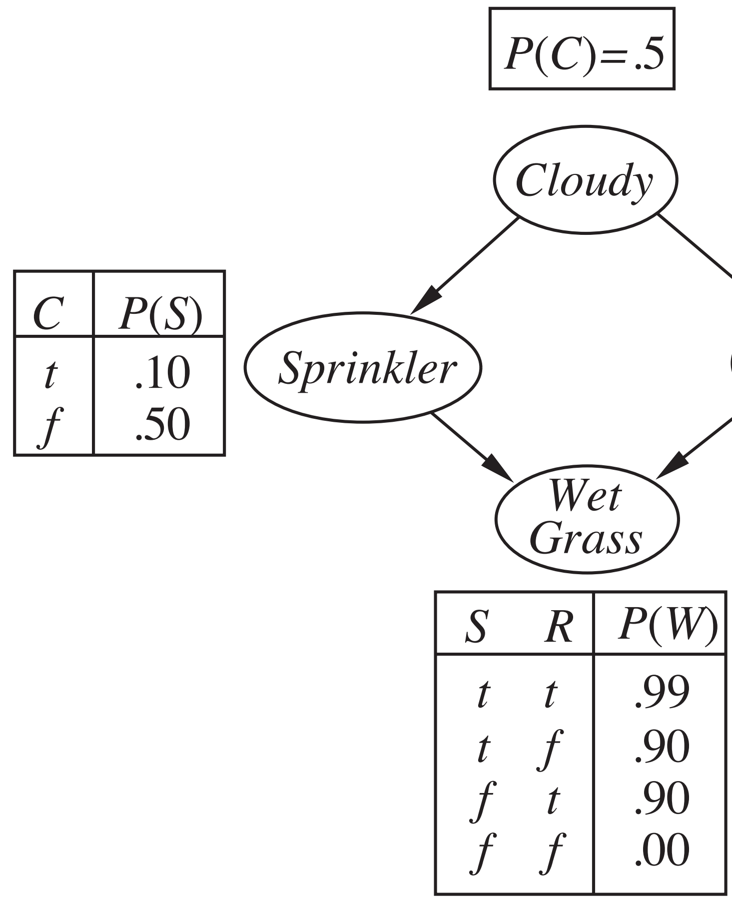
  - **construct a new graph** without loops, but with more complicated nodes (called **junction tree**)

  - perform a form of **message passing** on the junction tree (**linear** in the size of the junction tree)

doesn't make the problem less hard!

- the result explicitly represents marginals

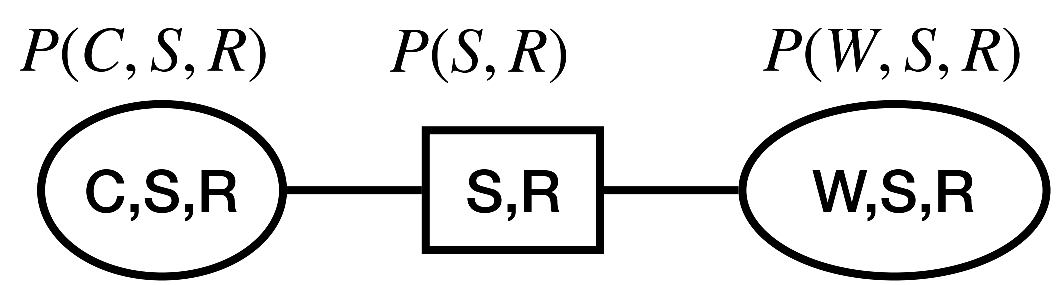
Idea



$$P(C,S,R,W) = P(C)P(S\,|\,C)P(R\,|\,C)P(W\,|\,S,R)$$

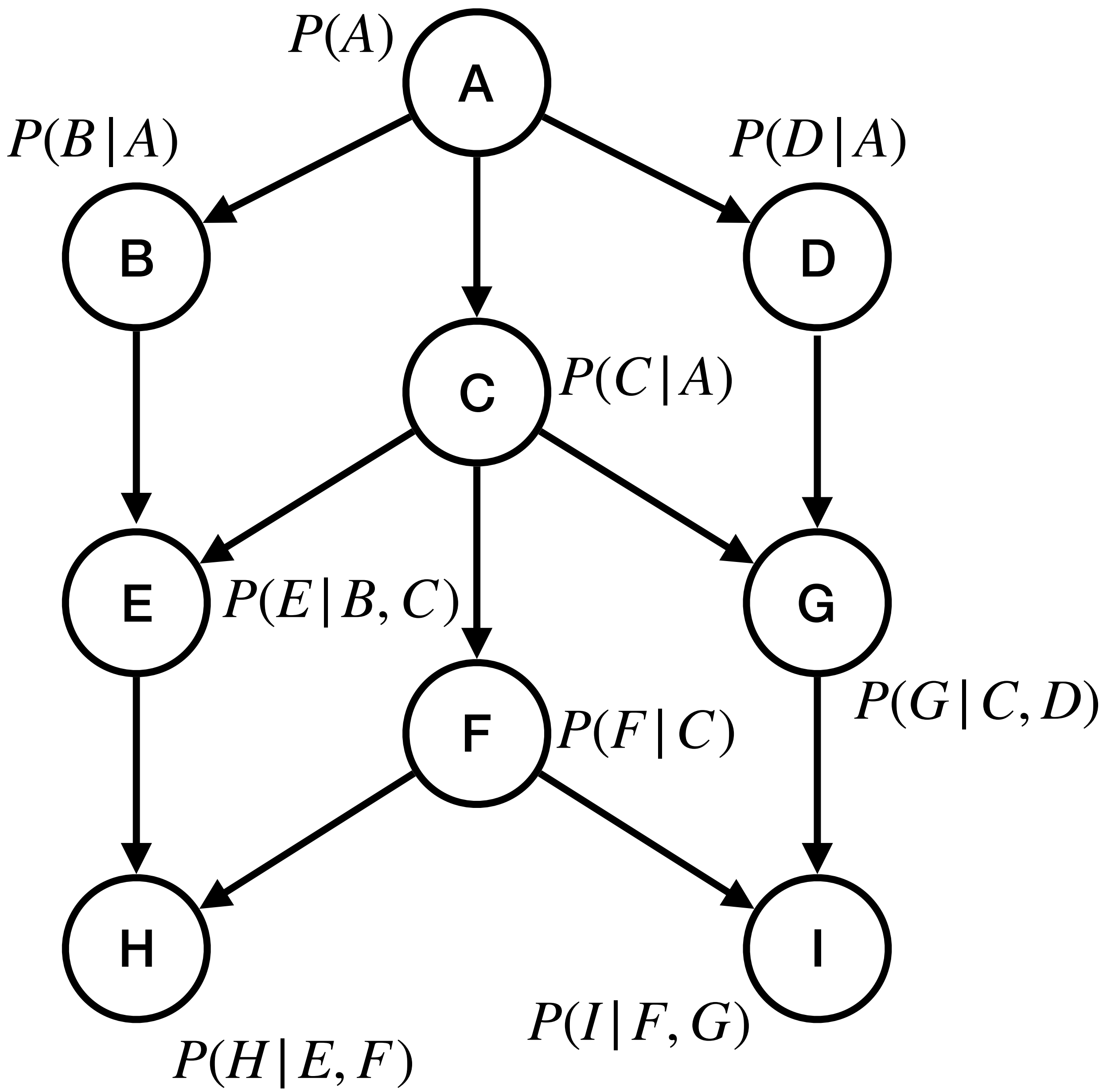
$$= P(C,S,R)\frac{P(W,S,R)}{P(S,R)}$$

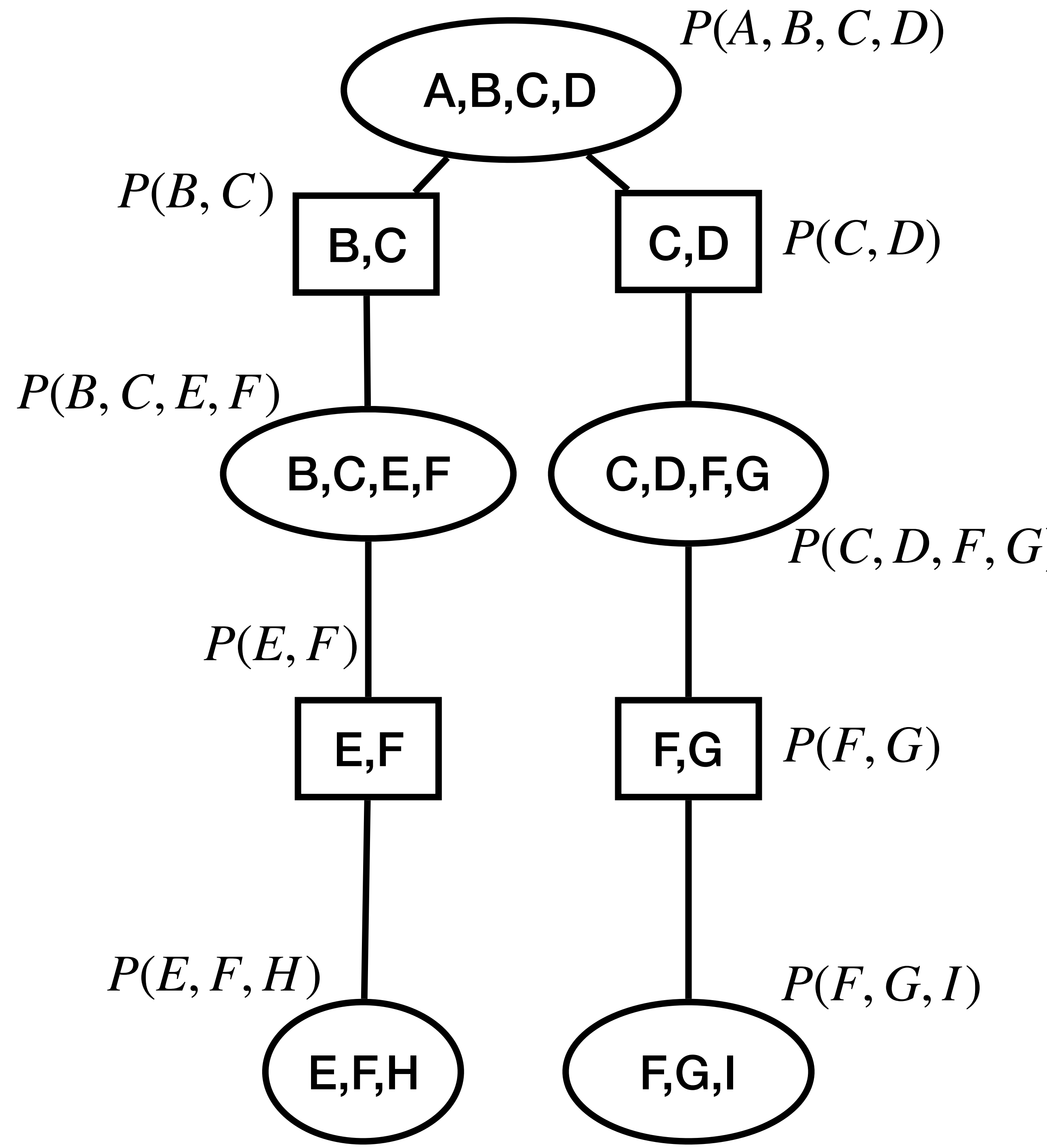
if we can transform the BN into a representation of this factorisation, then we can compute marginals as local summation operators



[figure: Russell& Norvig]

$$P(A)P(B|A)P(C|A)P(D|A)P(E|B,C)P(F|C)P(G|C,D)P(H|E,F)P(I|F,G)$$





$$\frac{P(A,B,C,D)P(B,C,E,F)P(C,D,F,G)P(E,F,H)P(F,G,I)}{P(B,C)P(C,D)P(E,F)P(F,G)}$$

[cf Barber Fig 6.11]

Junction Tree Algorithm

[see Barber ch 6 for correctness]

1. **Moralise**: if input is BN, turn it into an MN (marry all unmarried parents + drop edge directions)

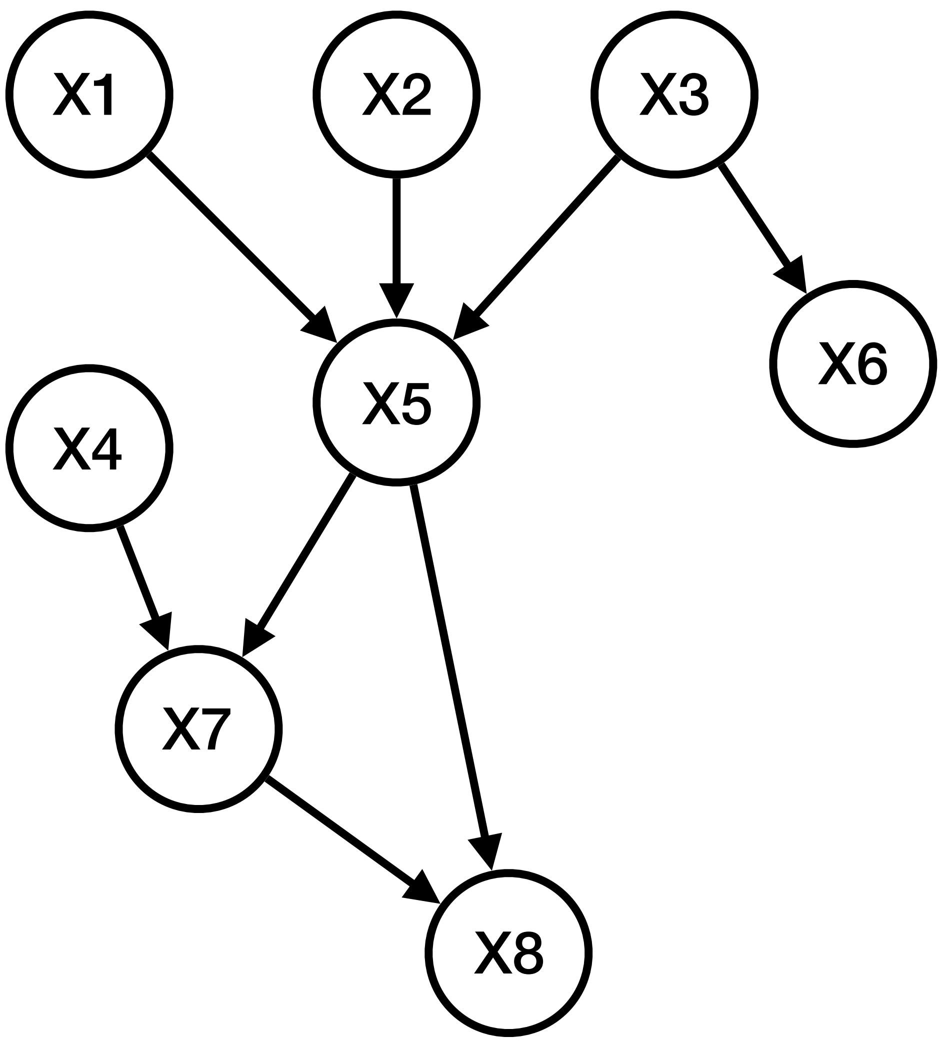
2. **Triangulate**: ensure that every loop involving four or more edges has a chord (a shortcut edge)

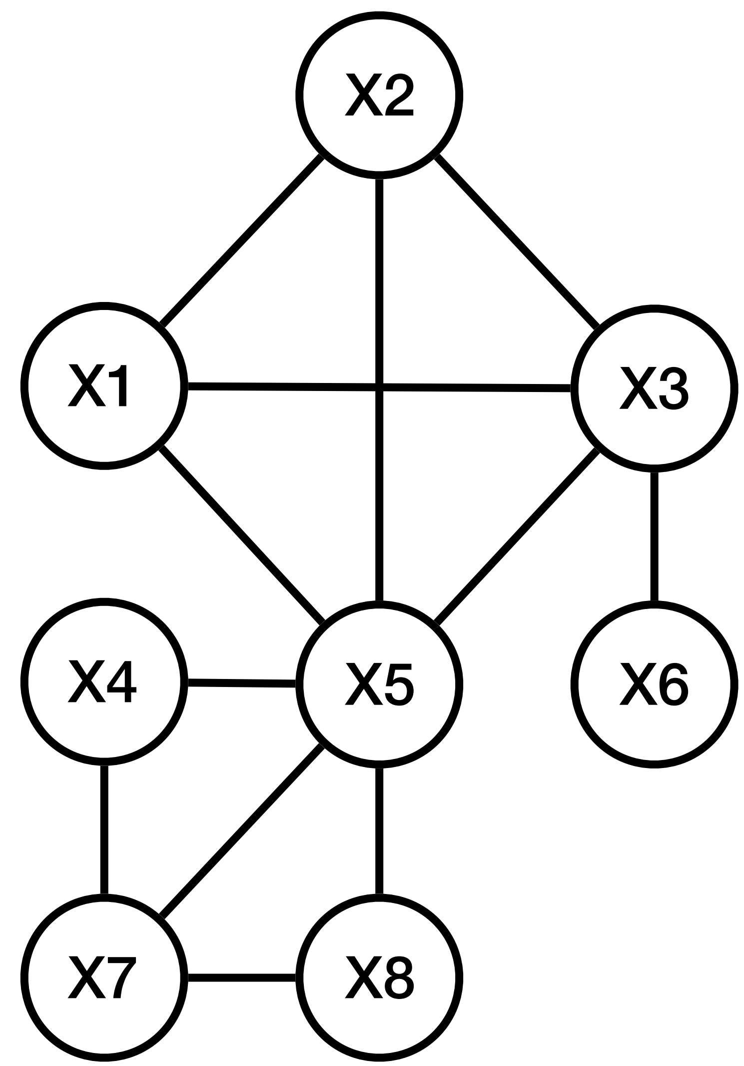
3. **Construct** junction tree: turn the triangulated graph into a so-called **clique graph** and find a maximal weighted **spanning tree** of that graph

4. Assign potentials to nodes

5. Pass messages

Moralisation

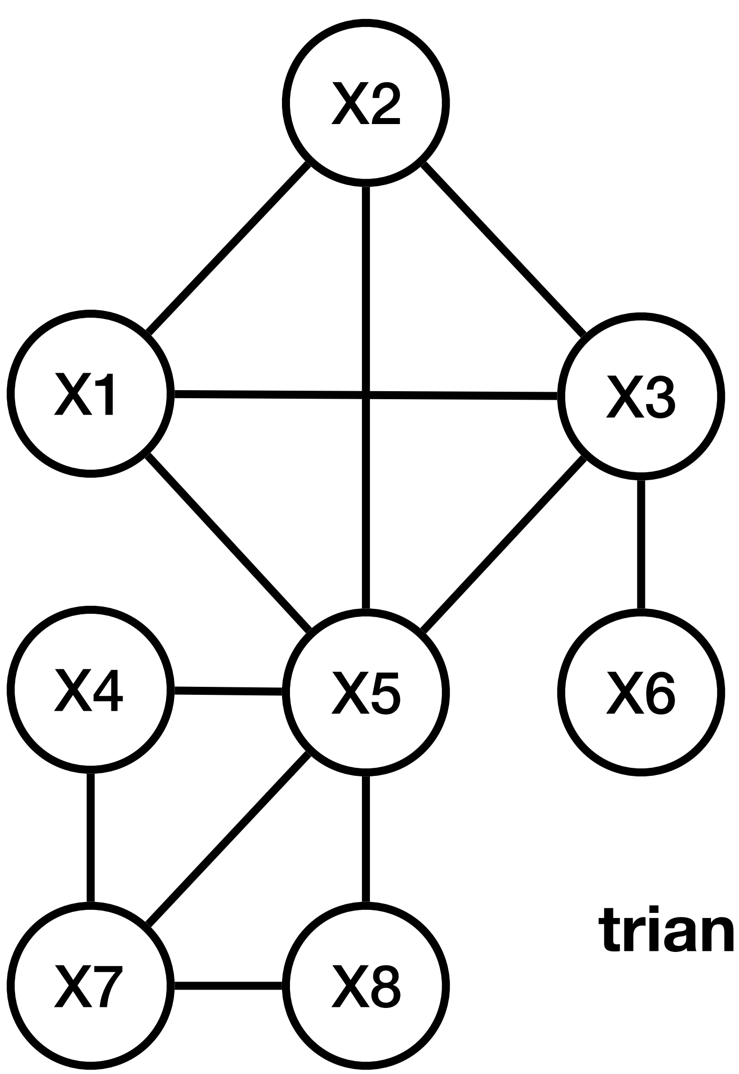




Triangulation

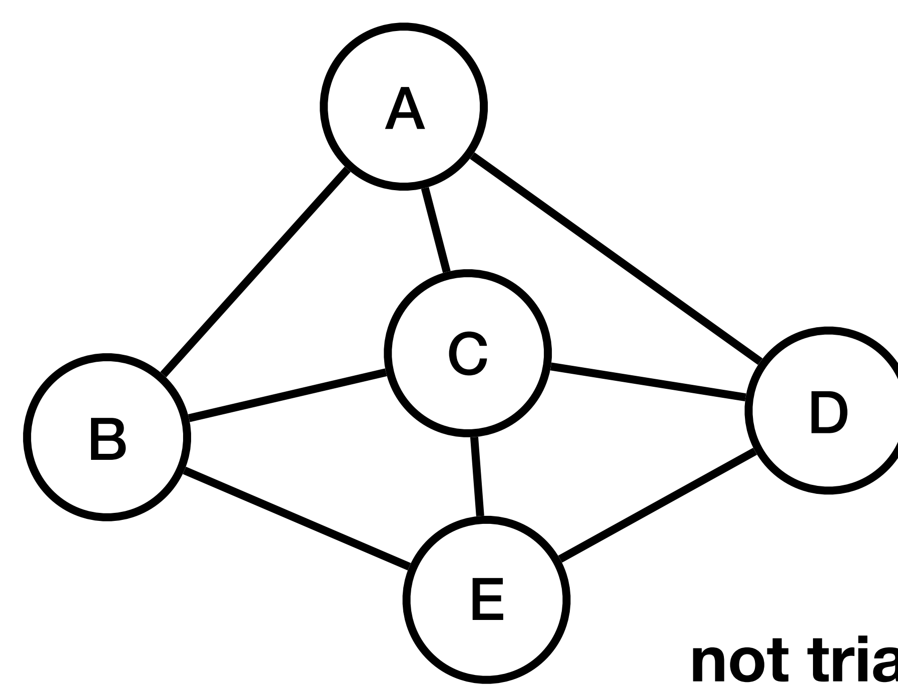
An undirected graph is **triangulated** if every loop of length four or more has a chord.

- choose an arbitrary node and label it 1
- for i = 2 to n
  - choose a node with the highest number of labeled neighbours and label it i
  - if i has two labeled neighbours without an edge between them, FAIL



triangulated

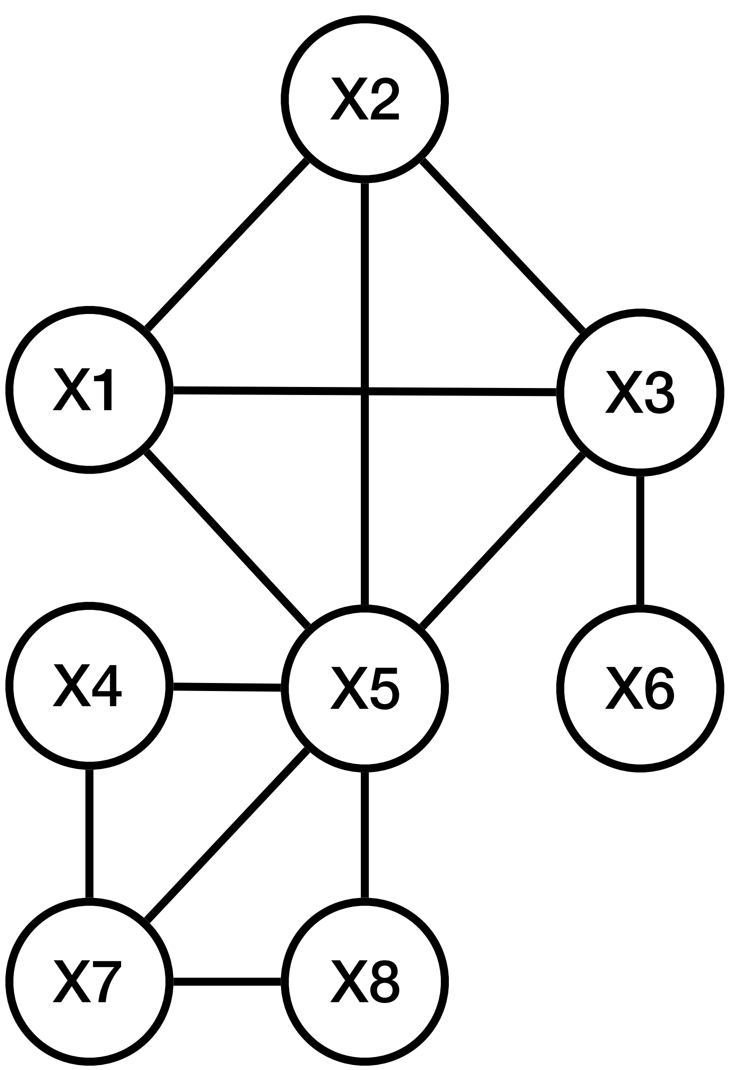
can be used to triangulate: in case of FAIL, add the edge(s) causing the failure & restart

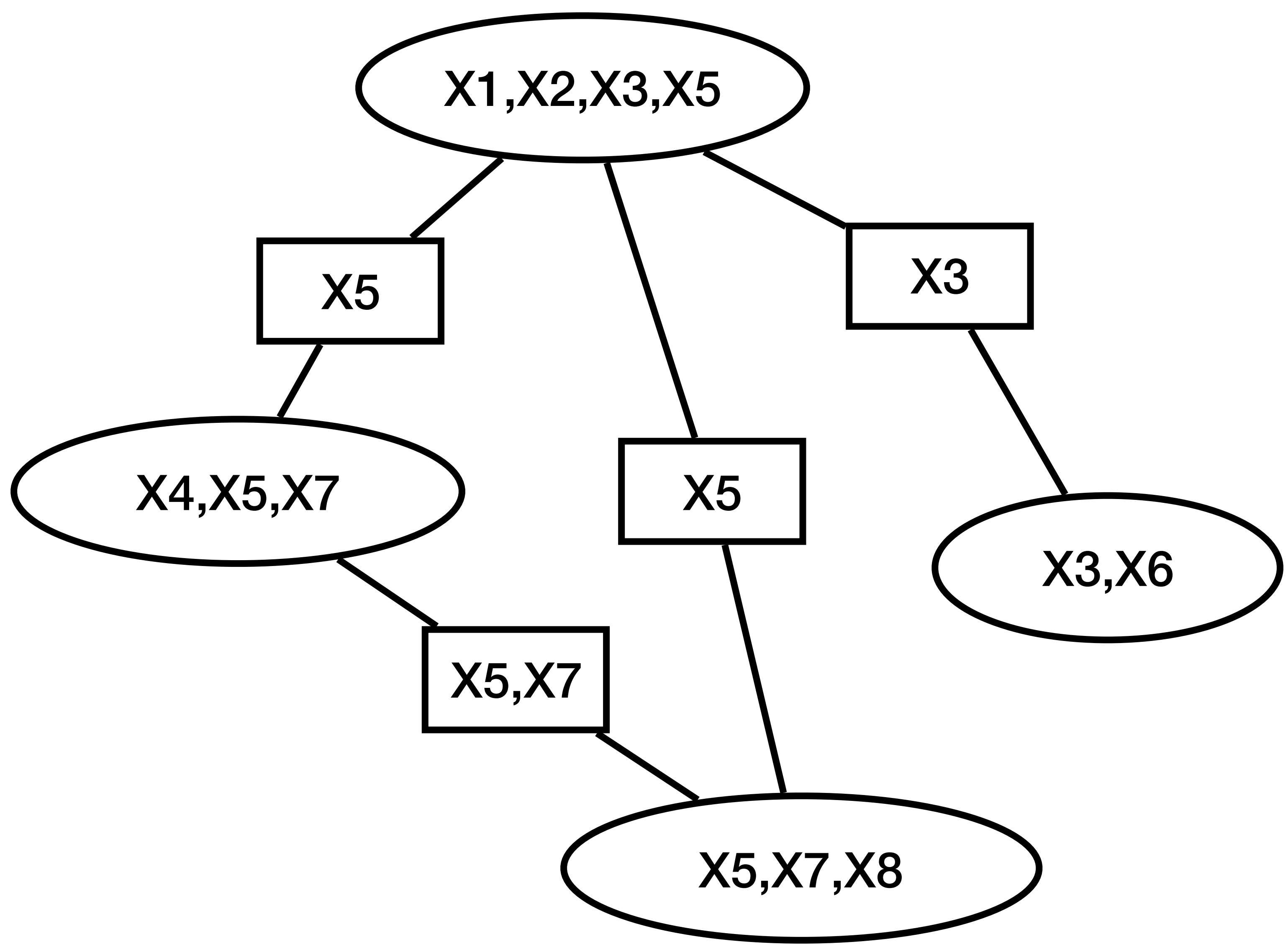


not triangulated

Clique Graph

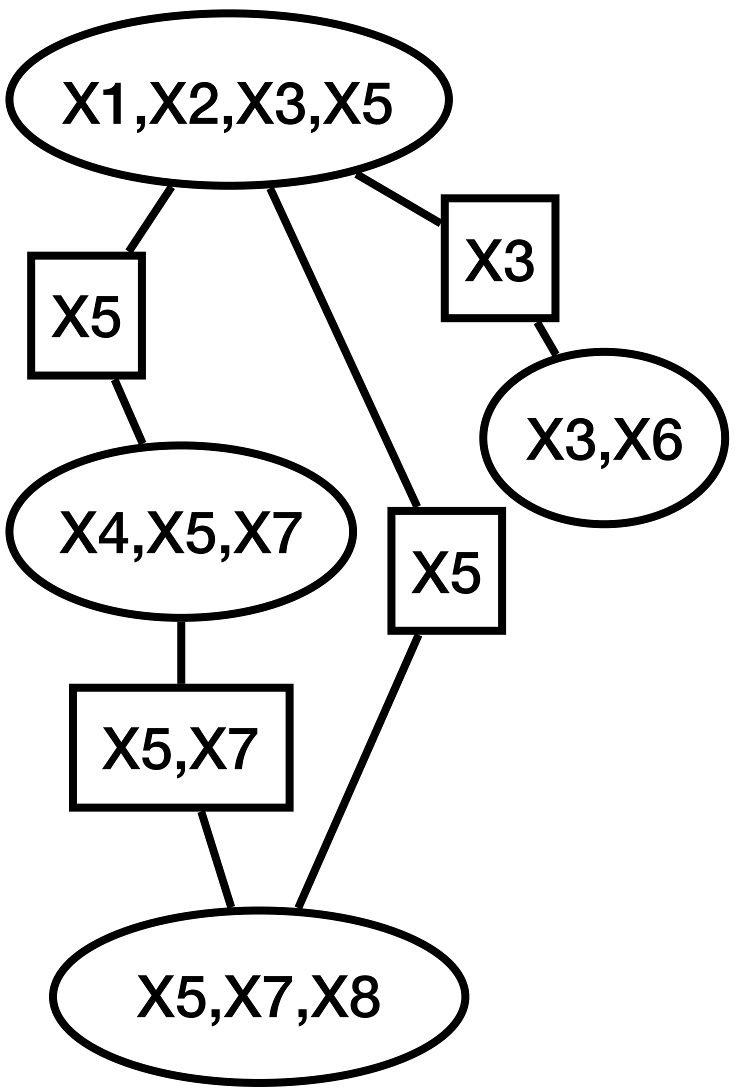
A **clique graph** contains a **clique node** for the set of variables of each potential, a **separator node** for each pair of neighbouring cliques' shared variables, and edges between the separators and their cliques.

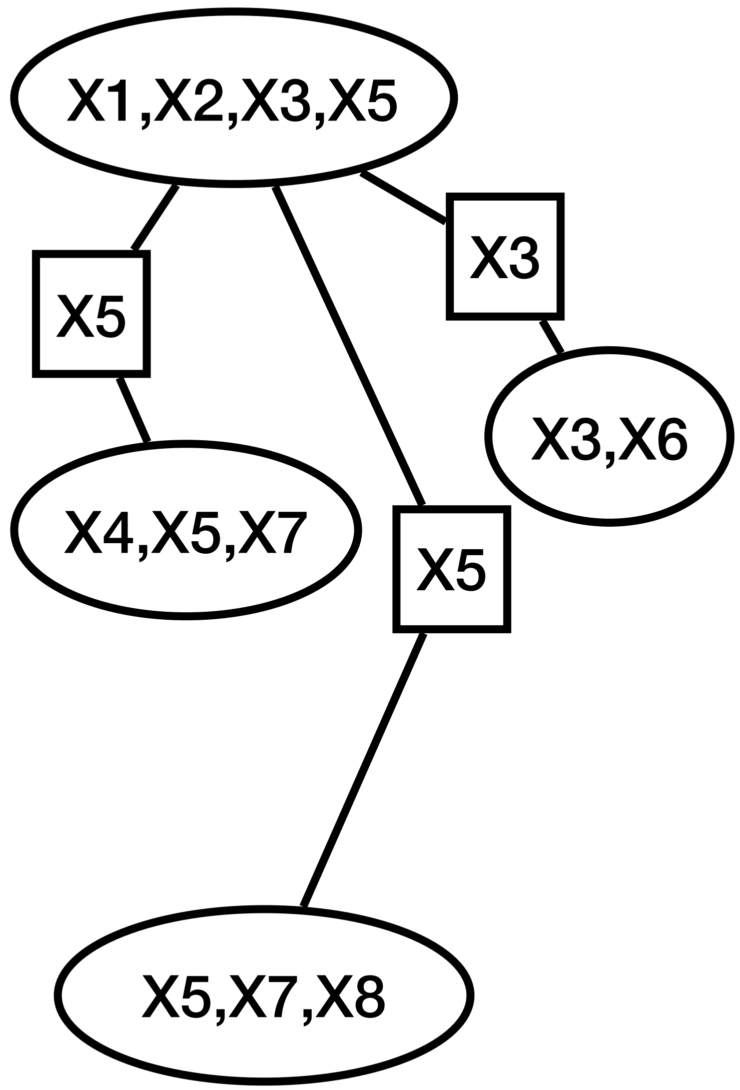




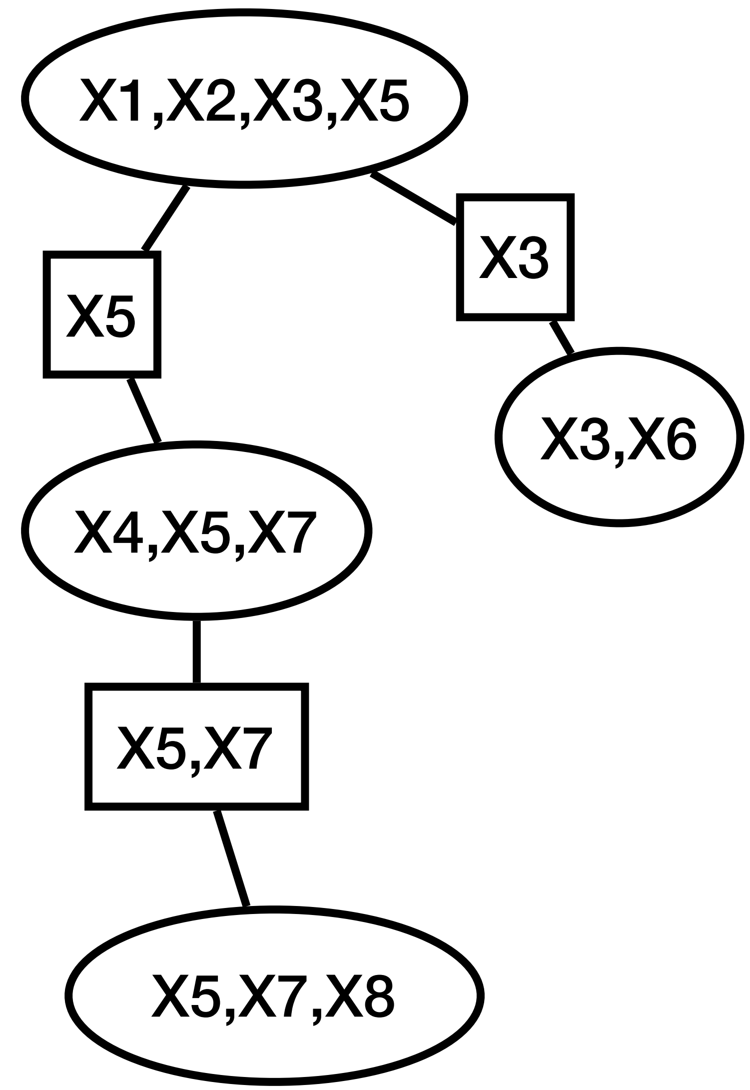
Junction Tree

A **junction tree** is a **clique graph without loops** that satisfies the **running intersection property**, i.e., for all pairs of nodes $\mathscr{X}$ and $\mathscr{Y}$, all nodes on the path between $\mathscr{X}$ and $\mathscr{Y}$ contain $\mathscr{X} \cap \mathscr{Y}$



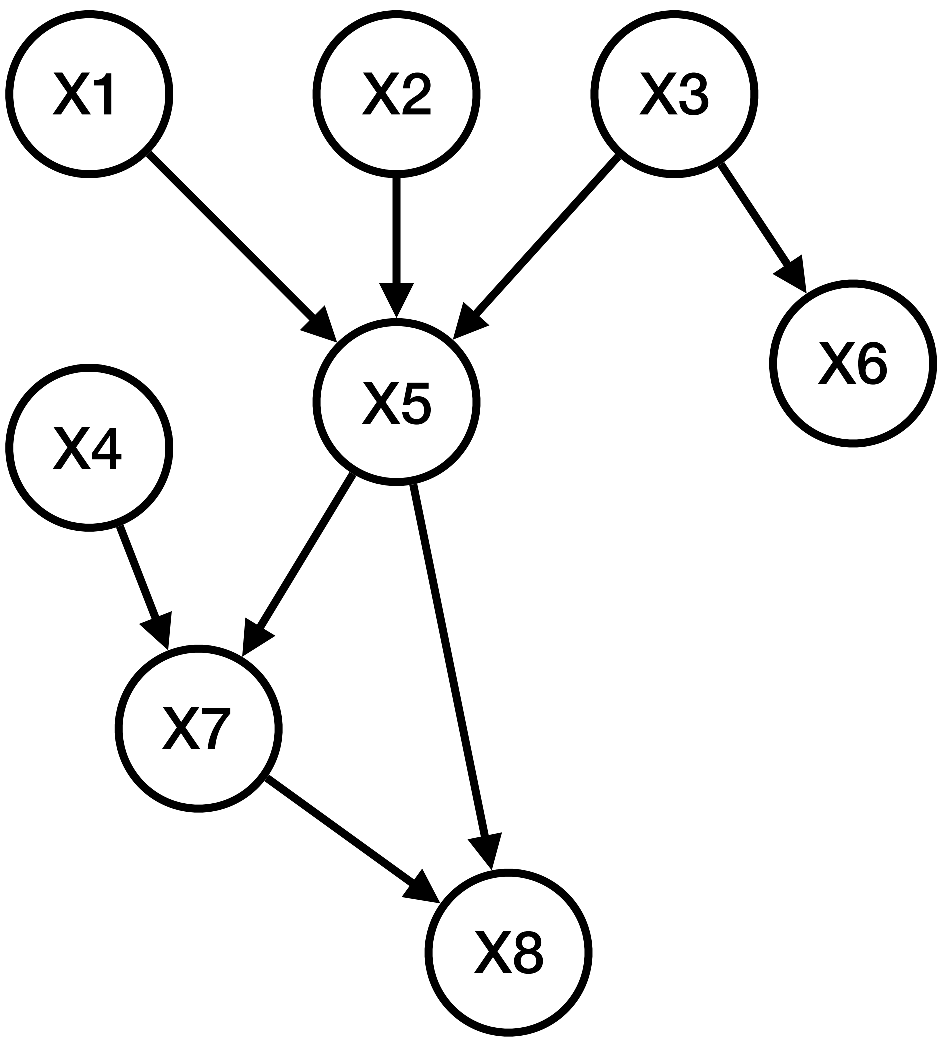


not a junction tree

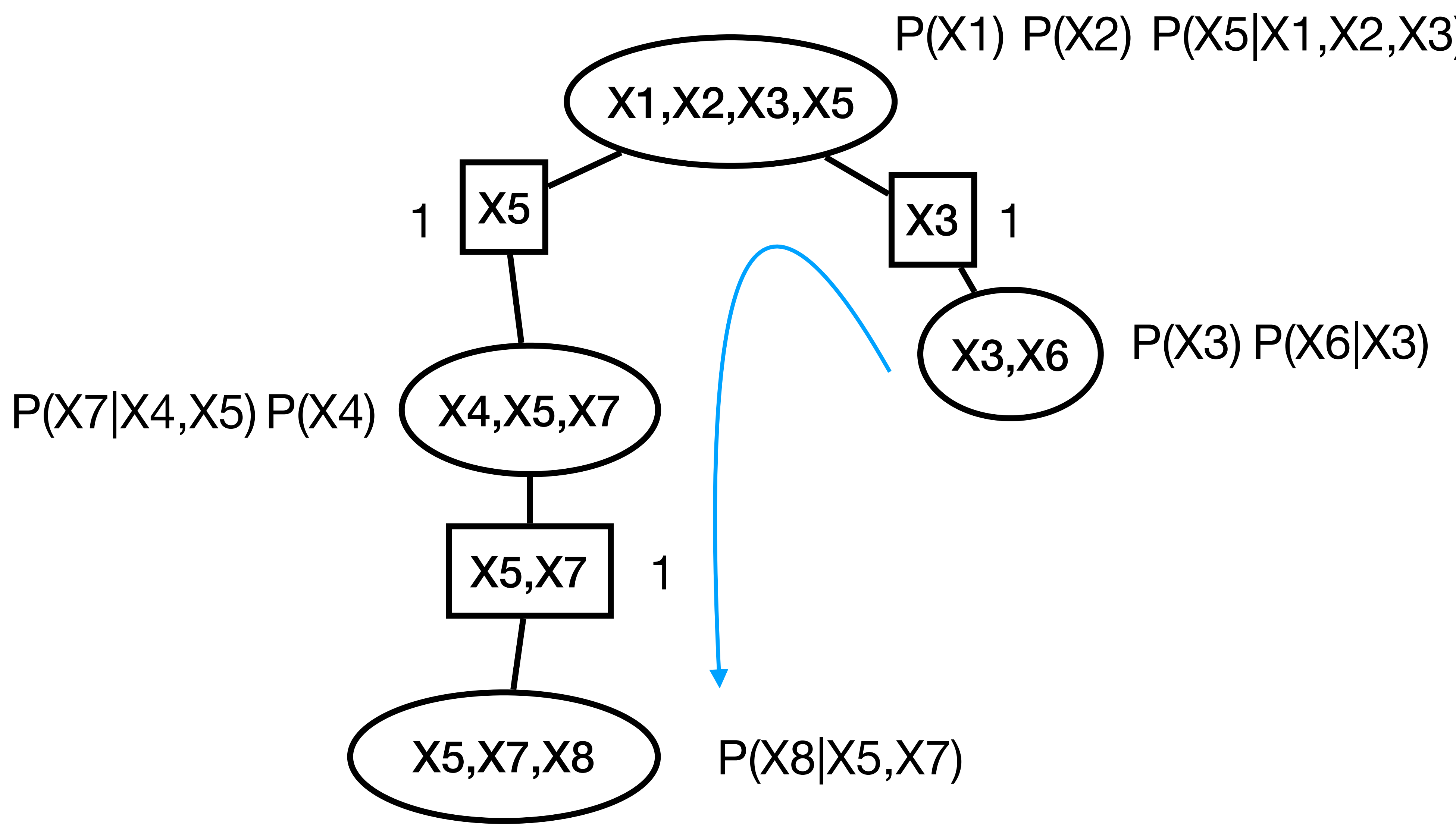


a junction tree

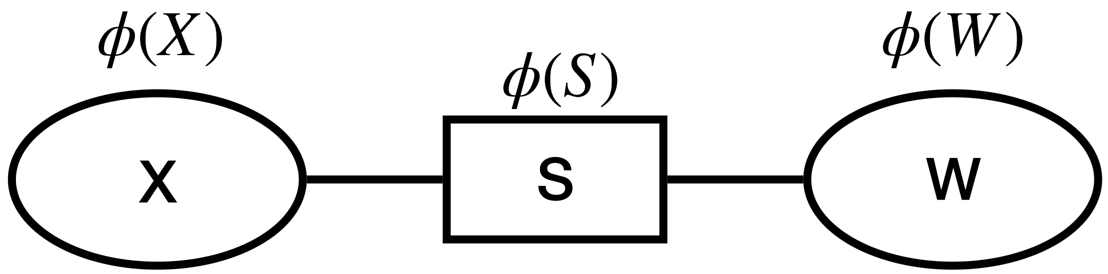
Junction Tree Potentials



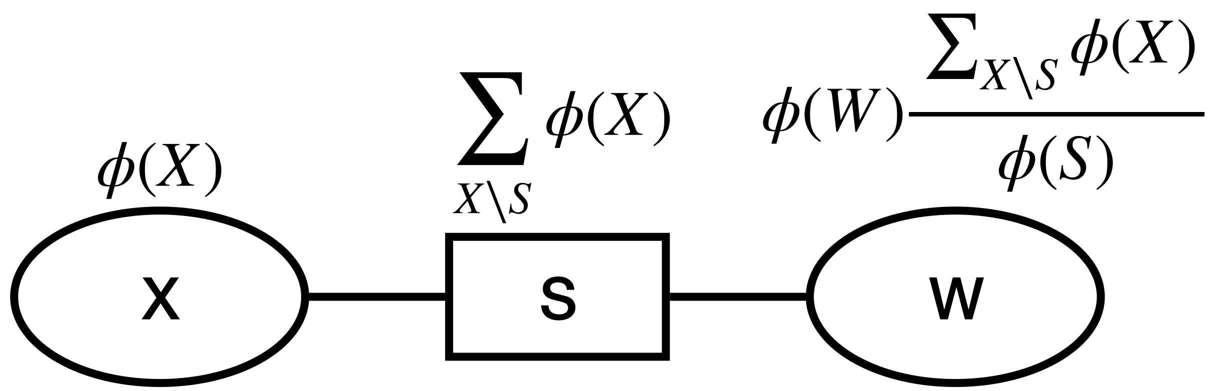
fix an order of clique nodes,
assign each potential $\phi(\mathscr{X})$ to the first clique node in the order containing all variables in $\mathscr{X}$,
set node potentials to the product of the assigned potentials (or 1 if none)

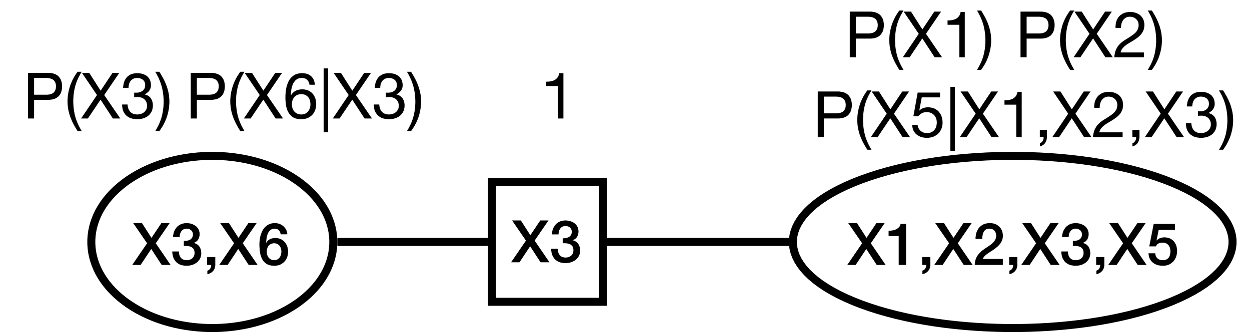


JT Message Passing



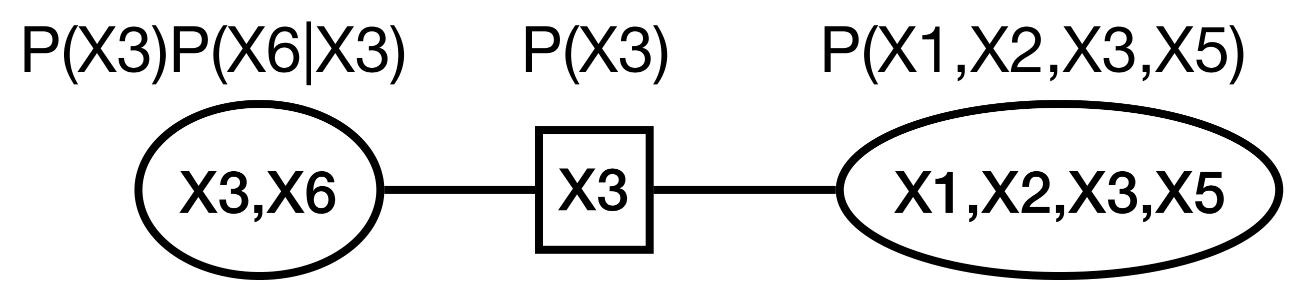
after sending a message from X to W, we get:

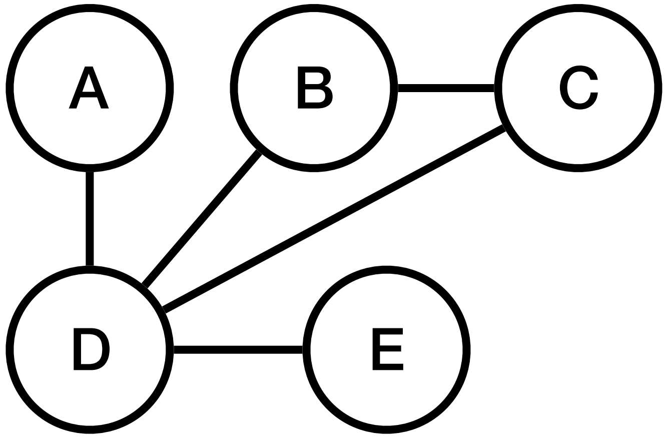




$$\sum_{X6} P(X3)P(X6 \mid X3) = P(X3)$$

$$P(X1)P(X2)P(X5 \mid X1,X2,X3)\frac{P(X3)}{1} = P(X1,X2,X3,X5)$$



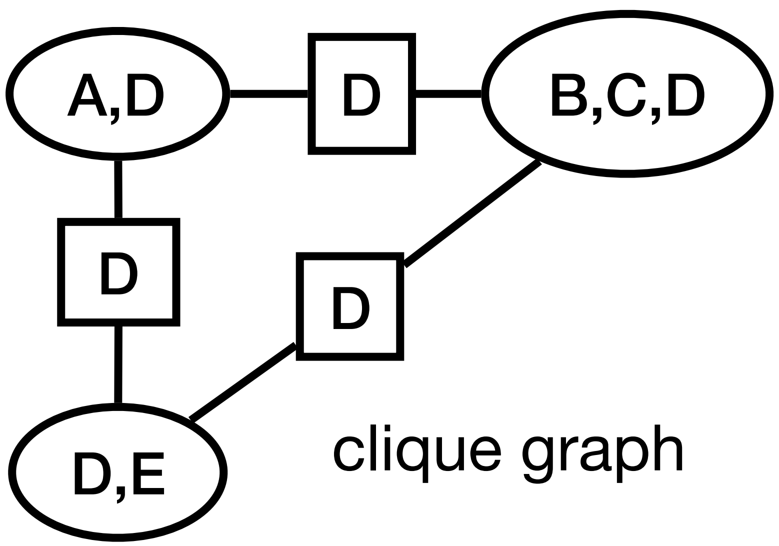


is triangulated

A	D	f_1(A,D)
0	0	5
0	1	2
1	0	1
1	1	1

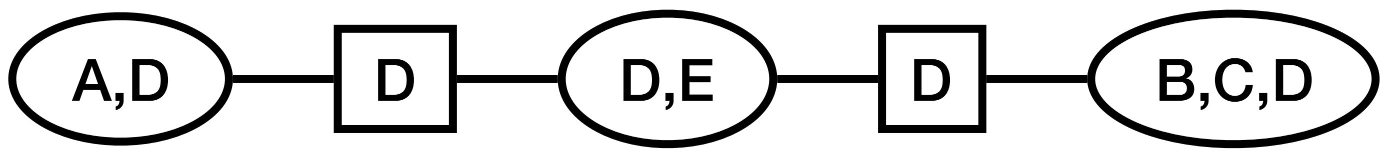
D	E	f_3(D,E)
0	0	1
0	1	0
1	0	0
1	1	1

B	C	D	f_2(B,C,D)
0	0	0	10
0	0	1	1
0	1	0	1
0	1	1	5
1	0	0	1
1	0	1	5
1	1	0	5
1	1	1	10



clique graph

junction tree

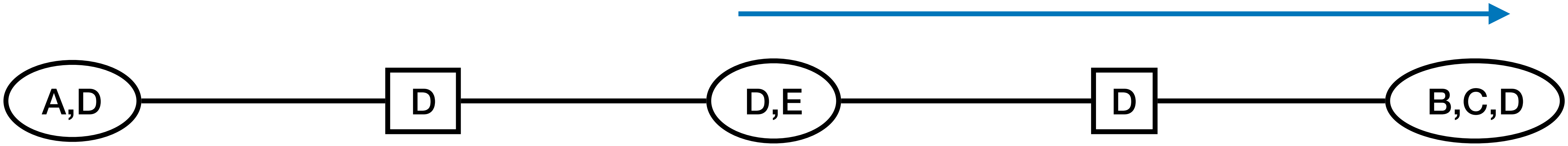


$f_1(A,D)$ 1 $f_3(D,E)$ 1 $f_2(B,C,D)$

$$\sum_A f_1(A,D) \qquad f_3(D,E)\frac{f_4(D)}{1}$$

D	f_4(D)
0	6
1	3

D	E	f_5(D,E)
0	0	6
0	1	0
1	0	0
1	1	3



A	D	f_1(A,D)
0	0	5
0	1	2
1	0	1
1	1	1

D	f_4(D)
0	6
1	3

D	E	f_5(D,E)
0	0	6
0	1	0
1	0	0
1	1	3

1

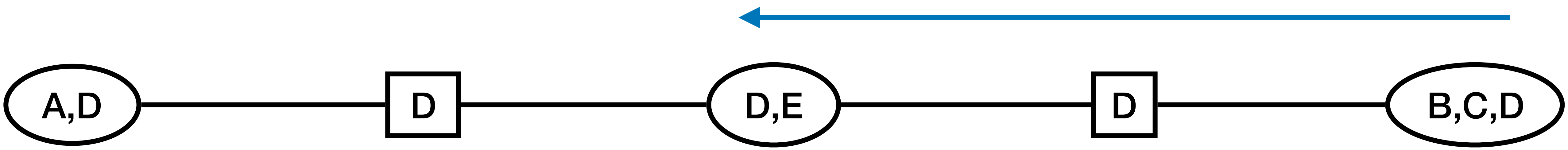
$$\sum_E f_5(D,E)$$

D	f_6(D)
0	6
1	3

B	C	D	f_2(B,C,D)
0	0	0	10
0	0	1	1
0	1	0	1
0	1	1	5
1	0	0	1
1	0	1	5
1	1	0	5
1	1	1	10

$$f_2(B,C,D)\frac{f_6(D)}{1}$$

B	C	D	f_7(B,C,D)
0	0	0	60
0	0	1	3
0	1	0	6
0	1	1	15
1	0	0	6
1	0	1	15
1	1	0	30
1	1	1	30



A	D	f_1(A,D)
0	0	5
0	1	2
1	0	1
1	1	1

D	f_4(D)
0	6
1	3

D	E	f_5(D,E)
0	0	6
0	1	0
1	0	0
1	1	3

D	f_6(D)
0	6
1	3

B	C	D	f_7(B,C,D)
0	0	0	60
0	0	1	3
0	1	0	6
0	1	1	15
1	0	0	6
1	0	1	15
1	1	0	30
1	1	1	30

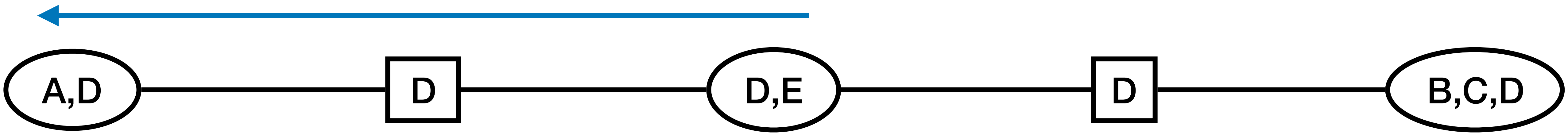
$$\sum_{B,C} f_7(B,C,D)$$

$$f_5(D,E)\frac{f_8(D)}{f_6(D)}$$

D	f_8(D)
0	102
1	63

D	E	f_9(D,E)
0	0	102
0	1	0
1	0	0
1	1	63

D	f_8(D)/f_6(D)
0	102/6=17
1	63/3=21



A	D	f_1(A,D)
0	0	5
0	1	2
1	0	1
1	1	1

D	f_4(D)
0	6
1	3

D	E	f_9(D,E)
0	0	102
0	1	0
1	0	0
1	1	63

D	f_8(D)
0	102
1	63

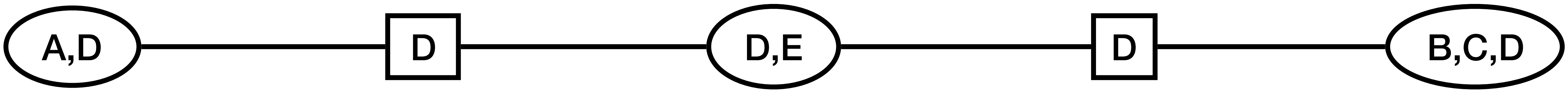
B	C	D	f_7(B,C,D)
0	0	0	60
0	0	1	3
0	1	0	6
0	1	1	15
1	0	0	6
1	0	1	15
1	1	0	30
1	1	1	30

$$\sum_E f_9(D,E)$$

D	f_{10}(D)
0	102
1	63

$$f_1(A,D)\frac{f_{10}(D)}{f_4(D)}$$

A	D	f_{11}(A,D)
0	0	85
0	1	42
1	0	17
1	1	21



A	D	$f_{11}(A,D)$
0	0	85
0	1	42
1	0	17
1	1	21

D	$f_{10}(D)$
0	102
1	63

D	E	$f_9(D,E)$
0	0	102
0	1	0
1	0	0
1	1	63

D	$f_8(D)$
0	102
1	63

B	C	D	$f_7(B,C,D)$
0	0	0	60
0	0	1	3
0	1	0	6
0	1	1	15
1	0	0	6
1	0	1	15
1	1	0	30
1	1	1	30

these are unnormalised marginals: to compute the normalisation constant Z, pick one of the tables and sum out all variables

$$Z = \sum_D f_{10}(D) = 102 + 63 = 165$$

to compute the marginal P(A,D), normalise $f_{11}(A, D)$:

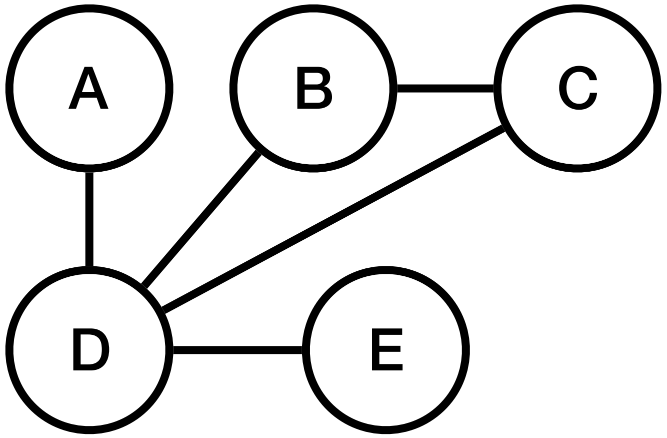
A	D	P(A,D)
0	0	85/165=0.515
0	1	42/165=0.255
1	0	17/165=0.103
1	1	21/165=0.127

to compute the marginal P(A), sum out D:

A	P(A)
0	85/165+42/165=0.7697
1	17/165+21/165=0.2303

similar for marginals of other variables

Homework



A	D	$f_1(A,D)$
0	0	5
0	1	2
1	0	1
1	1	1

D	E	$f_3(D,E)$
0	0	1
0	1	0
1	0	0
1	1	1

B	C	D	$f_2(B,C,D)$
0	0	0	10
0	0	1	1
0	1	0	1
0	1	1	5
1	0	0	1
1	0	1	5
1	1	0	5
1	1	1	10

- From the information on the previous slide, compute the marginals of the other variables, P(B), P(C), P(D) and P(E), as well.

- Compute the marginal of each variable using the original Markov network (repeated above) and the definition of the marginals, i.e.,
  $$P(A) = \frac{1}{Z} \sum_{B,C,D,E} f_1(A,D) f_2(B,C,D) f_3(D,E)$$ etc, where
  $$Z = \sum_{A,B,C,D,E} f_1(A,D) f_2(B,C,D) f_3(D,E)$$

- Construct the factor graph for the Markov network and use the sum-product algorithm (seen last week) to compute the marginal of each variable.

Where are we?

- **Variable elimination**: use the definition of marginal probability, but exploit distributivity and caching of intermediate results to save steps
- **Sum-product algorithm**: message passing on singly connected factor graph to compute all single variable marginals
- **Bucket elimination**: compute the marginal of one variable on any graph, using buckets to organise computation
- **Junction tree algorithm**: compute all single variable marginals on any graph
- All of these perform **exact** inference & can get expensive
- Alternative: **approximate** inference by **sampling**

Sampling

- sampling = randomly selecting elements from Ω according to a distribution P over Ω
- remember data generation in the statistical learning framework
- a large number of i.i.d. (independently and identically distributed) samples from P can be used to estimate the probability of any event $E \subseteq \Omega$

Example

C	P(C)
England	0.9
Scotland	0.1
Wales	0.0

E,E,E,E,E,S,E,E,E,E

C	P(C)
England	0.85
Scotland	0.05
Wales	0.10

E,E,E,E,E,S,E,E,E,E,W,W,E,E,E,E,E,E,E,E

C	P(C)
England	0.83
Scotland	0.07
Wales	0.10

E,E,E,E,E,S,E,E,E,E,W,W,E,E,E,E,E,E,E,E,E,E,E,S,E,E,E,E,E,W

C	P(C)
England	0.85
Scotland	0.075
Wales	0.075

E,E,E,E,E,S,E,E,E,E,W,W,E,E,E,E,E,E,E,E,E,E,E,S,E,E,E,E,E,W,E,E,E,E,E,E,E,E,S

E,E,E,E,E,S,E,E,E,E,W,W,E,E,E,E,E,E,E,E,E,E,E,S,E,E,E,E,E,W,E,E,E,E,E,E,E,E,S,
E,S,E,E,E,E,E,E,E,E

C	P(C)
England	0.86
Scotland	0.08
Wales	0.06

...

C	P(C)
England	0.88
Scotland	0.08
Wales	0.04

Sampling in practice

- **Assumption**: we have a **random number generator** that generates i.i.d. samples from the uniform distribution over the interval [0,1], e.g., {0.432,0.100,0.773,0.310,0.130,0.004,0.055,0.931,0.445,0.723}

- **hardware random number generators** harvest “entropy” to generate random numbers (measuring atmospheric noise, thermal noise or other external phenomena, see also https://en.wikipedia.org/wiki//dev/random)

- **pseudo random number generators** generate “random” numbers deterministically from an initial “seed” number

Sampling for a single RV

- to sample from P(C), create the **cumulant:**

$$t_0 = 0$$
$$t_1 = t_0 + P(C = England) = 0.88$$
$$t_2 = t_1 + P(C = Scotland) = 0.96$$
$$t_3 = t_2 + P(C = Wales) = 1.00$$

C	P(C)
England	0.88
Scotland	0.08
Wales	0.04

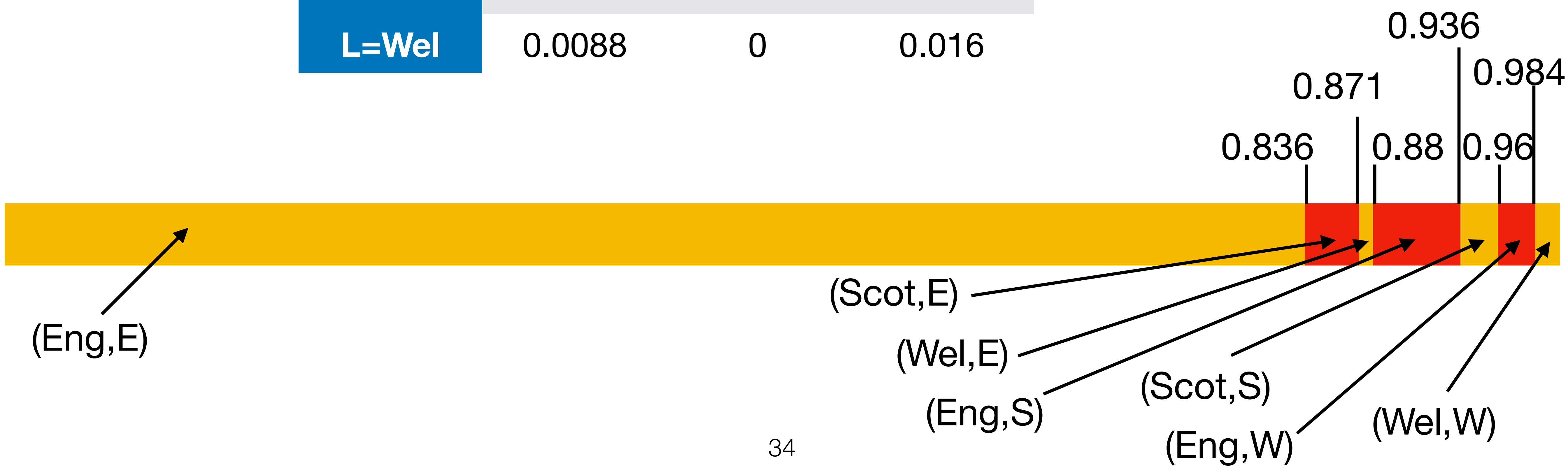
random numbers {0.432, 0.100, 0.773, 0.310, 0.130, 0.004, 0.055, 0.931, 0.445, 0.723}

sample E E E E E E E S E E

Sampling from a joint distribution

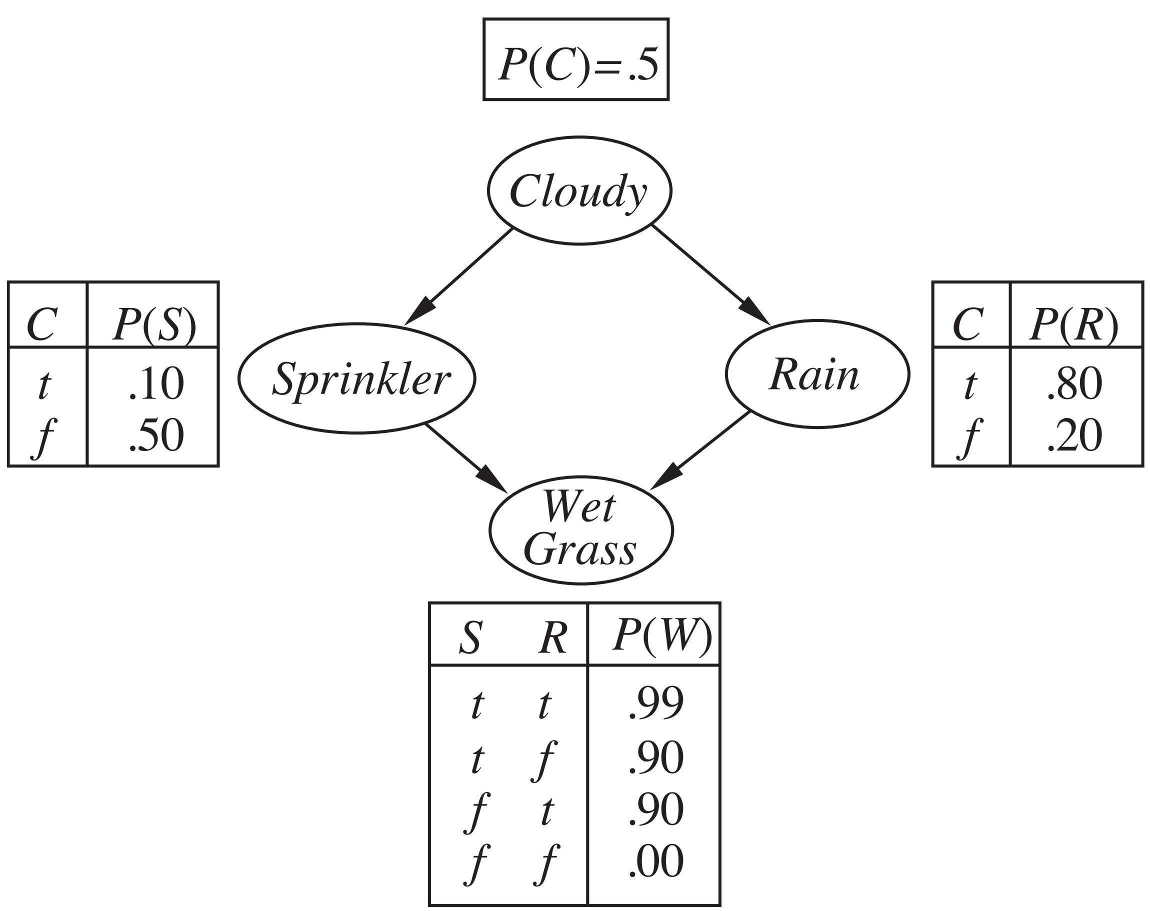
- the same idea can in principle be used for joint distributions: we simply get a more fine grained cumulant

P(C,L)	C=E	C=S	C=W
L=Eng	0.836	0.056	0.024
L=Scot	0.0352	0.024	0
L=Wel	0.0088	0	0.016



Sampling from a BN

- If the joint distribution is given as BN, we can use the structure:
  - fix an order of the variables such that every child comes after its parents
  - sample each node in order from its conditional distribution



C,S,R,W

sample C from P(C) -> t

sample S from P(S|C=t) -> f

sample R from P(R|C=t) -> t

sample W from P(W|S=f,R=t) -> t

full sample (C=t,S=f,R=t,W=t)

[figure: Russell& Norvig]

Estimating probabilities from samples

- Given a set of samples from the joint distribution, we can estimate answers to general probability questions

(C=1,S=0,R=1,W=1)
(C=0,S=0,R=0,W=0)
(C=1,S=0,R=1,W=0)
(C=0,S=0,R=1,W=1)
(C=1,S=1,R=1,W=1)
(C=1,S=0,R=1,W=1)
(C=0,S=0,R=1,W=0)
(C=1,S=1,R=1,W=1)
(C=1,S=0,R=1,W=1)
(C=1,S=1,R=1,W=0)

marginal of S?

$P(S=0) = 7/10$
$P(S=1) = 3/10$

conditional distribution of C given W=1?

$P(C=0 \mid W=1) = 5/6$
$P(C=1 \mid W=1) = 1/6$

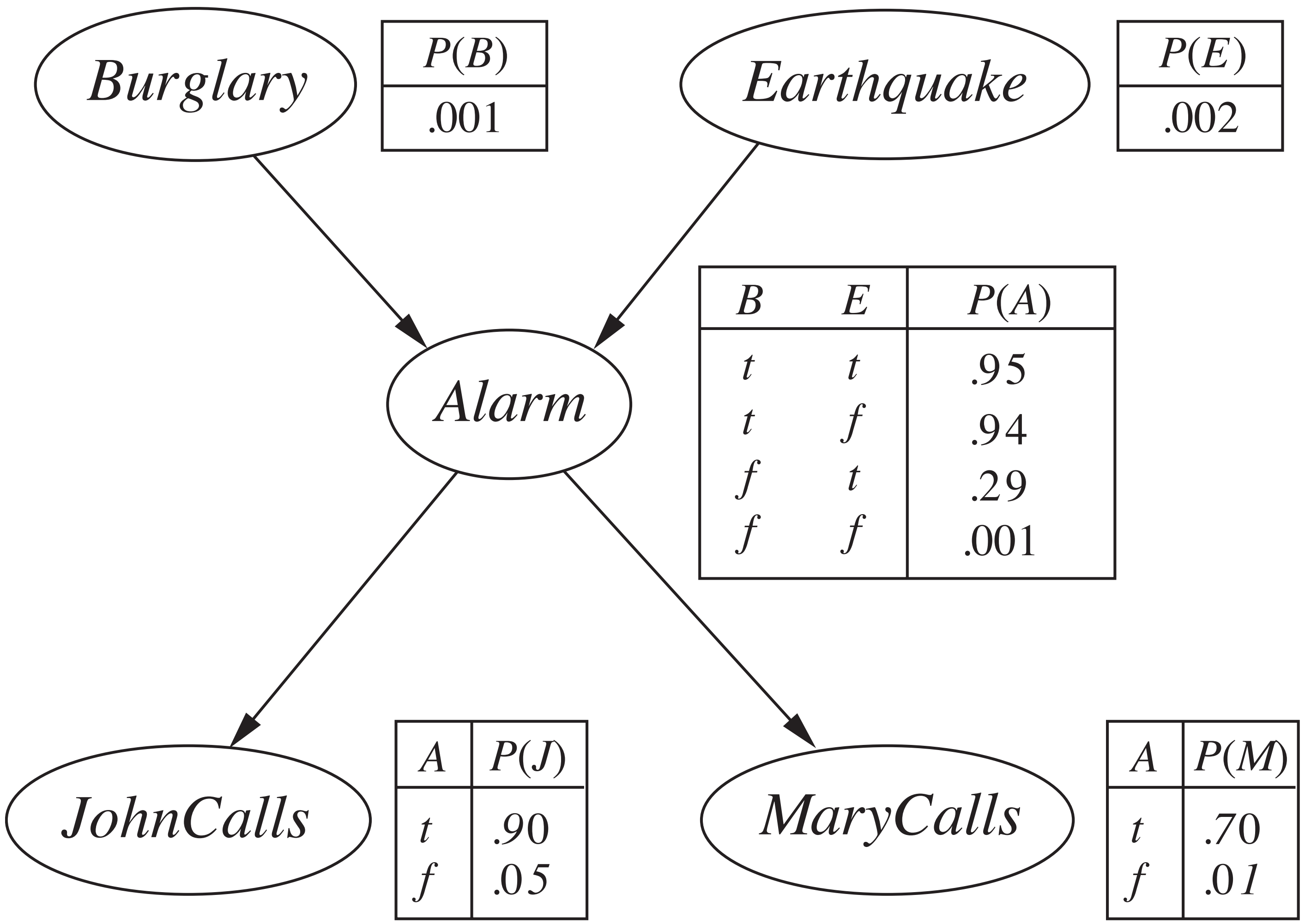
Sampling from a BN

- How to sample from a conditional distribution, say P(C|W=t)?
- **Rejection sampling**: sample from the joint distribution as before, but throw away all samples where W≠t
- Problem: the more observations we condition on, the more samples are thrown away
  - observing N Boolean variables rules out all but one of their 2^N possible states

Sampling from a BN

- **Likelihood weighting** avoids this problem by only generating samples that are consistent with the evidence
- Sample unobserved variables as in the case of the joint distribution, but set observed ones to their given value
- no longer i.i.d.: we cannot just compute frequencies
- instead: each sample is weighted by its likelihood, i.e., the product of the conditional probabilities of the observations given their parents in the sample

Example



P(B | J = t, E = t)

B = f	1
E = t	1*0.002
A = f	0.002
J = t	0.002*0.05
M = t	0.0001

[figure: Russell& Norvig]

Likelihood weighting

- can be much more efficient than rejection sampling
- performance can suffer as number of observations increases, especially if evidence variables are late in the order the variables are sampled in

Gibbs sampling

- Gibbs sampling generates a chain of samples by sampling each variable in turn, conditioned on the current values of all other variables
- given current state $(X_1 = v_1, \ldots, X_n = v_n)$, sample new value for X_i from $P(X_i \mid X_1 = v_1, \ldots, X_{i-1} = v_{i-1}, X_{i+1} = v_{i+1}, \ldots, X_n = v_n)$
- often easier to sample from a distribution over just one variable

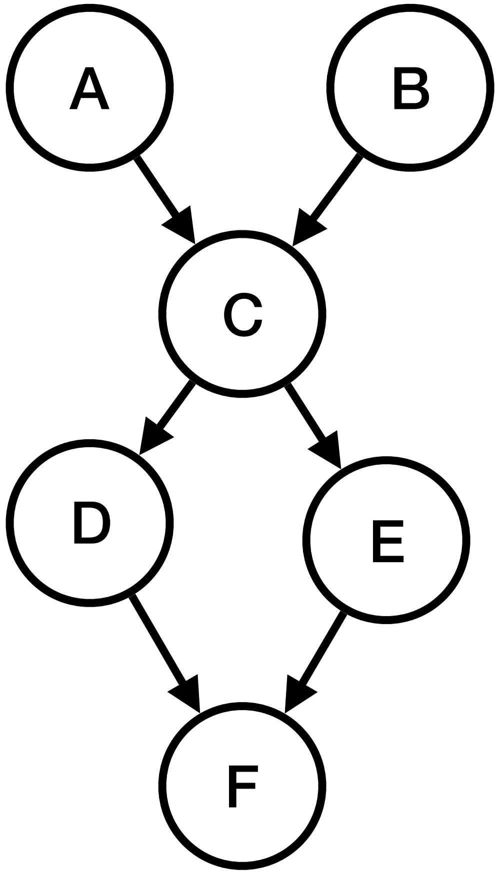
Gibbs sampling

- In a **Bayesian network**, each variable is conditionally independent of all others given its Markov blanket

- thus, need to sample from $P(X_i \mid MB(X_i))$ knowing the values of the variables in the Markov blanket, where $P(X_i \mid MB(X_i)) \propto P(X_i \mid parents(X_i)) \prod_{j \in ch(X_i)} P(X_j \mid parents(X_j))$

- easy to obtain (and normalise)

- observed variables can simply be fixed and excluded from sampling



P(A=1)	P(B=1)
0.3	0.5

A	B	P(C=1|A,B)	P(C=0|A,B)
0	0	0.9	0.1
0	1	0.3	0.7
1	0	0.2	0.8
1	1	0.2	0.8

C	P(D=1|C)	P(D=0|C)
0	0.1	0.9
1	0.2	0.8

C	P(E=1|C)	P(E=0|C)
0	0.3	0.7
1	0.6	0.4

D	E	P(F=1|D,E)	P(F=0|D,E)
0	0	0.5	0.5
0	1	0.9	0.1
1	0	0.5	0.5
1	1	0.6	0.4

start with arbitrary sample:
(A=1,B=0,C=0,D=1,E=1,F=0)

sample from $P(A \mid B=0, C=0) \propto P(A)P(C=0 \mid A, B=0)$

$$P(A=1)P(C=0 \mid A=1, B=0) = 0.3 \cdot 0.7 = 0.21$$
$$P(A=0)P(C=0 \mid A=0, B=0) = 0.7 \cdot 0.1 = 0.07$$

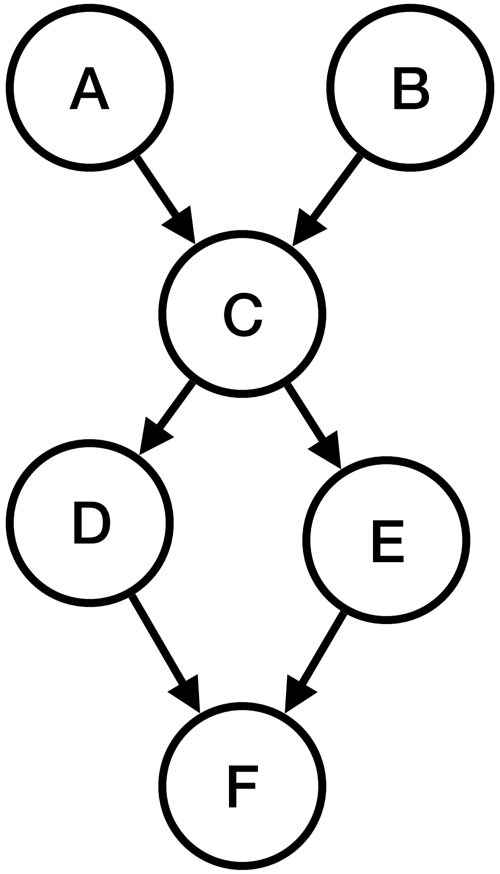
$$P(A=1 \mid B=0, C=0) = 0.75$$
$$P(A=0 \mid B=0, C=0) = 0.25$$

say, A=1, i.e., next sample: (A=1,B=0,C=0,D=1,E=1,F=0)

sample from $P(B \mid A=1, C=0) \propto P(B)P(C=0 \mid A=1, B)$

$$P(B=1 \mid A=1, C=0) = 0.5$$
$$P(B=0 \mid A=1, C=0) = 0.5$$

say, B=1, i.e., next sample: (A=1,B=1,C=0,D=1,E=1,F=0)



P(A=1)
0.3

P(B=1)
0.5

A	B	P(C=1|A,B)	P(C=0|A,B)
0	0	0.9	0.1
0	1	0.3	0.7
1	0	0.2	0.8
1	1	0.2	0.8

C	P(D=1|C)	P(D=0|C)
0	0.1	0.9
1	0.2	0.8

C	P(E=1|C)	P(E=0|C)
0	0.3	0.7
1	0.6	0.4

D	E	P(F=1|D,E)	P(F=0|D,E)
0	0	0.5	0.5
0	1	0.9	0.1
1	0	0.5	0.5
1	1	0.6	0.4

current sample: (A=1,B=1,C=0,D=1,E=1,F=0)

sample from $P(C \mid A=1,B=1,D=1,E=1) \propto P(C \mid A=1,B=1)P(D=1 \mid C)P(E=1 \mid C)$

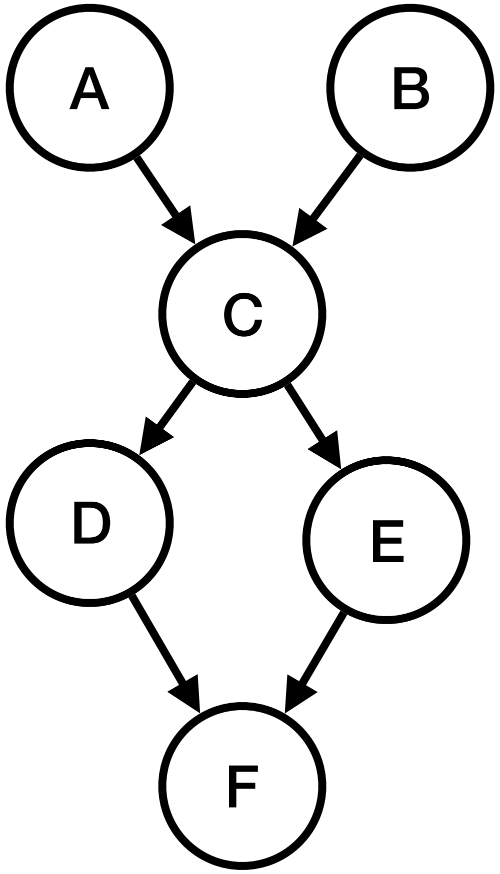
$$P(C=1 \mid A=1,B=1)P(D=1 \mid C=1)P(E=1 \mid C=1) = 0.2 \cdot 0.2 \cdot 0.6$$
$$P(C=0 \mid A=1,B=1)P(D=1 \mid C=0)P(E=1 \mid C=0) = 0.8 \cdot 0.1 \cdot 0.3$$

say, C=1, i.e., next sample: (A=1,B=1,C=1,D=1,E=1,F=0)

sample from $P(D \mid C=1,E=1,F=0) \propto P(D \mid C=1)P(F=0 \mid D, E=1)$

$$P(D=1 \mid C=1)P(F=0 \mid D=1,E=1) = 0.2 \cdot 0.4$$
$$P(D=0 \mid C=1)P(F=0 \mid D=0,E=1) = 0.8 \cdot 0.1$$

say, D=0, i.e., next sample:
(A=1,B=1,C=1,D=0,E=1,F=0)



P(A=1)	P(B=1)
0.3	0.5

A	B	P(C=1|A,B)	P(C=0|A,B)
0	0	0.9	0.1
0	1	0.3	0.7
1	0	0.2	0.8
1	1	0.2	0.8

C	P(D=1|C)	P(D=0|C)
0	0.1	0.9
1	0.2	0.8

C	P(E=1|C)	P(E=0|C)
0	0.3	0.7
1	0.6	0.4

D	E	P(F=1|D,E)	P(F=0|D,E)
0	0	0.5	0.5
0	1	0.9	0.1
1	0	0.5	0.5
1	1	0.6	0.4

current sample: (A=1,B=1,C=1,D=0,E=1,F=0)

sample from $P(E \mid C=1, D=0, F=0) \propto P(E \mid C=1) P(F=0 \mid D=0, E)$

$$P(E=1 \mid C=1) P(F=0 \mid D=1, E=1) = 0.6 \cdot 0.4$$

$$P(E=0 \mid C=1) P(F=0 \mid D=1, E=0) = 0.4 \cdot 0.5$$

say, E=0, i.e., next sample: (A=1,B=1,C=1,D=0,E=0,F=0)

sample from $P(F \mid D=0, E=0)$

say, F=1, i.e., next sample: (A=1,B=1,C=1,D=0,E=0,F=1)

next, sample A again, etc

(A=1,B=0,C=0,D=1,E=1,F=0)
(A=1,B=0,C=0,D=1,E=1,F=0)
(A=1,B=1,C=0,D=1,E=1,F=0)
(A=1,B=1,C=1,D=1,E=1,F=0)
(A=1,B=1,C=1,D=0,E=1,F=0)
(A=1,B=1,C=1,D=0,E=0,F=0)
(A=1,B=1,C=1,D=0,E=0,F=1)
…

Gibbs sampling

- straightforward to implement, but samples are **strongly dependent**
- therefore, it is common to
    - discard a fixed number of samples at the start (burn-in)
    - only use every k-th sample to estimate answers
- Gibbs sampling belongs to the wider class of Markov Chain Monte Carlo (MCMC) algorithms

MCMC algorithms

- MCMC algorithms generate the next sample by random modification of the current one
- if the random modification satisfies certain properties, in the long run, this generates each possible state with the correct frequency
- however, can be difficult to detect when this has happened, i.e., when estimates converge

Probabilistic Inference

- Inference uses a distribution to **answer questions**, e.g., to compute the conditional distribution of a variable given observations

- **Exact** algorithms manipulate formulas to obtain these answers; this can be done efficiently for singly-connected BNs or MNs, but is hard in general

- **Sampling** resorts to generating large numbers of examples from the distribution and uses these to estimate answers

Looking forward

- How to learn Bayesian networks from data?
  - given the graph, learn the parameters
  - learn both the graph structure & the parameters
  - learning as inference
- Probabilistic models involving time
- Probabilistic models involving objects and relations

Reading Material

- Today:
  - Russell & Norvig: 14.5
  - Barber: 6 & 27 (yes, that’s 27)

- Parts of slides based on
    - David Barber’s slides for the BRML book
    - Tinne De Laet & Luc De Raedt’s slides for the UAI course at KU Leuven