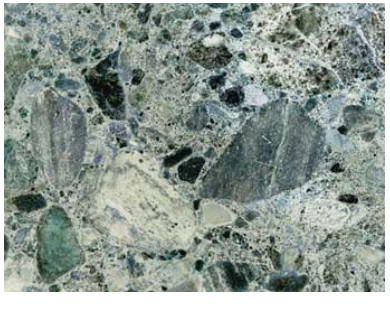
Graphical models

Marcus Frean
School of Mathematics, Statistics and Computer Science
Victoria University of Wellington

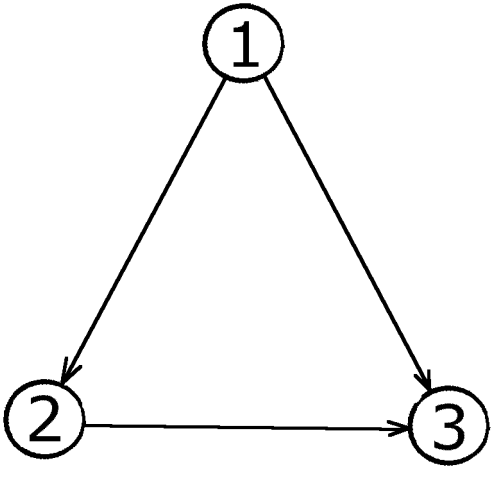
- probability propagation
- tractable inference?
- lines in marble?



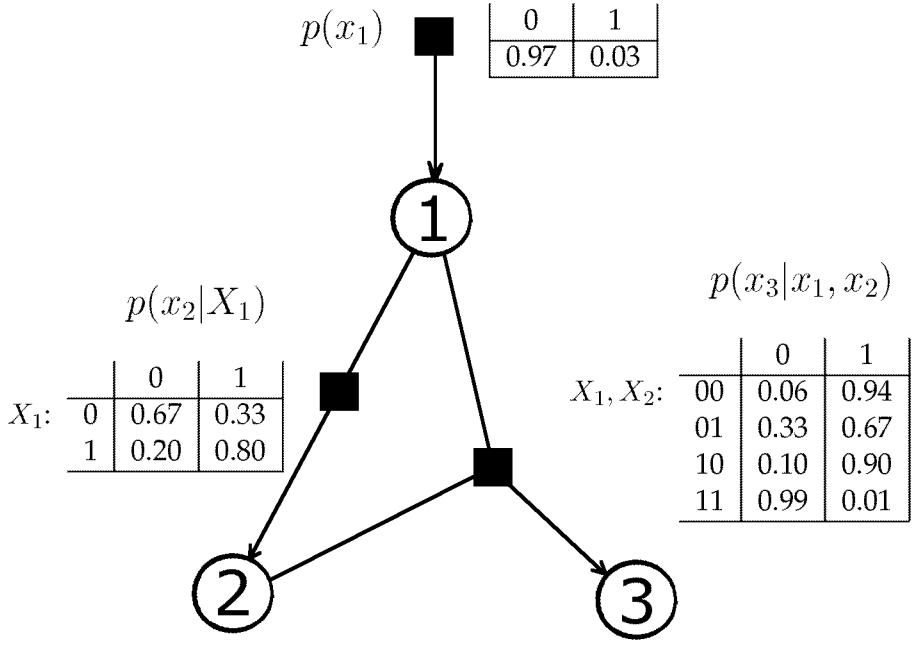
Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga
o te Ūpoko o te Ika a Māui

$$p(x_1, x_2, x_3) = p(x_3|x_1, x_2)\ p(x_2|x_1)\ p(x_1)$$

Belief net:



Factor graph:



another simple graph

Belief net:

another simple graph

Belief net:

Factorisation:

$$p(x_1, x_2, x_3) = p(x_3|x_2)\ p(x_2|x_1)\ p(x_1)$$

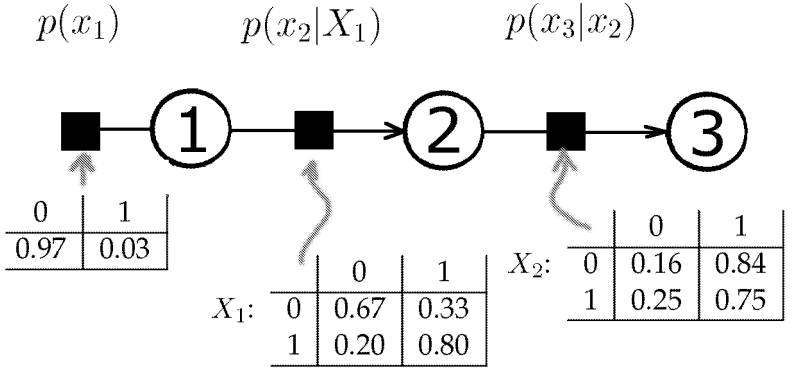
another simple graph

Belief net:

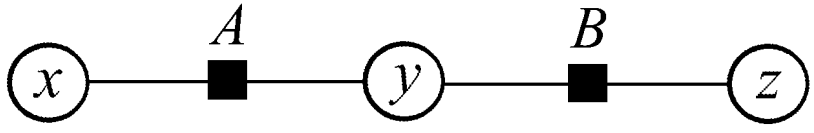
Factorisation:

$$p(x_1, x_2, x_3) = p(x_3|x_2)\ p(x_2|x_1)\ p(x_1)$$

Factor graph:



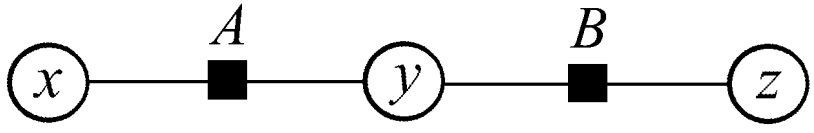
undirected graphs



$$p(x, y, z) = \Phi_A(x, y)\ \Phi_B(y, z)$$

x and z are conditionally independent given y.

undirected graphs



$$p(x, y, z) = \Phi_A(x, y)\ \Phi_B(y, z)$$

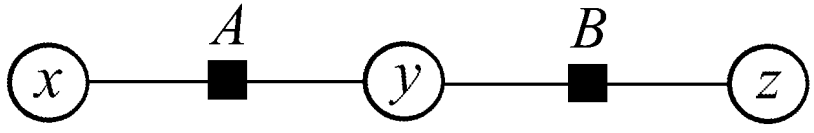
x and z are conditionally independent given y.

With only modest abuse of notation...

$$\begin{aligned} p(x) &= \sum_y \sum_z p(x, y, z) \\ &\propto \sum_y \sum_z \Phi_A(x, y)\ \Phi_B(y, z) \end{aligned}$$

but those sums can be distributed...

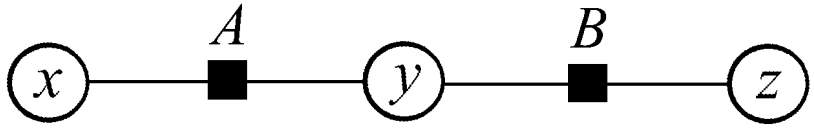
message passing



$$p(x) = \frac{1}{Z} \underbrace{\sum_y \Phi_A(x,y) \underbrace{\sum_z \Phi_B(y,z)}_{\text{msg from } B \text{ to } y}}_{\text{msg from } A \text{ to } x}$$

- probability propagation runs on the factor graph

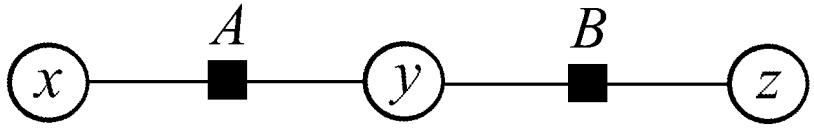
message passing



$$p(x) = \frac{1}{Z} \underbrace{\sum_y \Phi_A(x,y) \underbrace{\sum_z \Phi_B(y,z)}_{\text{msg from } B \text{ to } y}}_{\text{msg from } A \text{ to } x}$$

- probability propagation runs on the factor graph
- consists of messages sent to and from each node (variable or factor)

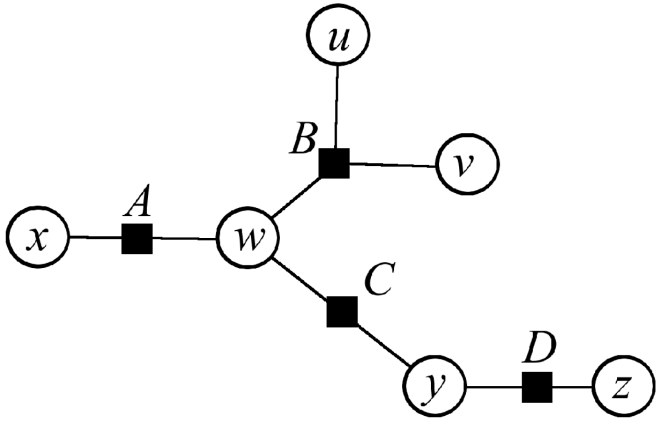
message passing



$$p(x) = \frac{1}{Z} \underbrace{\sum_{y} \Phi_A(x,y) \underbrace{\sum_{z} \Phi_B(y,z)}_{\text{msg from } B \text{ to } y}}_{\text{msg from } A \text{ to } x}$$

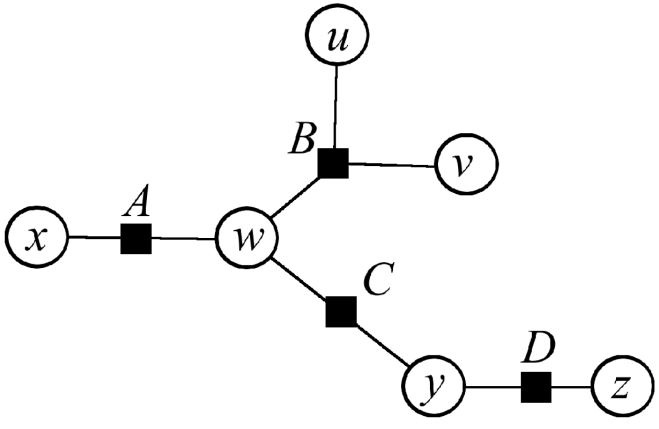
- probability propagation runs on the factor graph
- consists of messages sent to and from each node (variable or factor)
- message always consists of a probability distribution over the variable that the message is associated with

another example



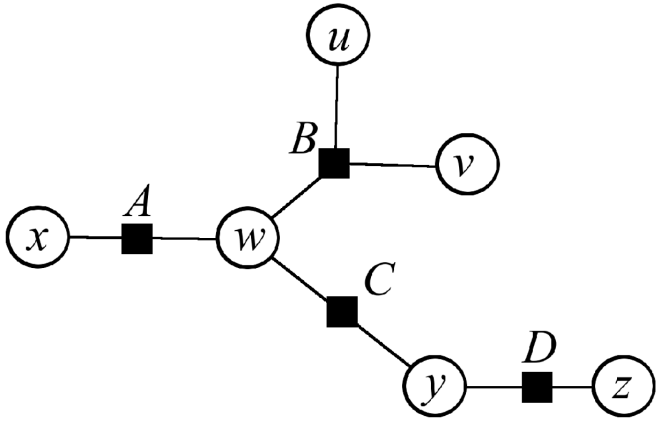
$$
\begin{aligned}
p(x) &= \sum_{u,v,w,y,z} p(u,v,w,x,y,z) \\
&\propto \sum_{u,v,w,y,z} \Phi_A(x,w)\ \Phi_B(u,v,w)\ \Phi_C(w,y)\ \Phi_D(y,z) \\
&\propto \sum_{w} \Phi_A(x,w) \sum_{u,v} \Phi_B(u,v,w) \sum_{y} \Phi_C(w,y) \sum_{z} \Phi_D(y,z)
\end{aligned}
$$

more on message passing



- each outgoing message is a simple function of all the incoming messages on *other* edges

more on message passing



- each outgoing message is a simple function of all the incoming messages on *other* edges
- the function is different for the two types of node

aside: Viterbi and EM work just as in HMMs

variable nodes are MULTIPLIERS

- To generate a message on an edge, they simply multiply the incoming messages on their other edges.

variable nodes are MULTIPLIERS

- To generate a message on an edge, they simply multiply the incoming messages on their other edges.
- If the variable is **observed**, they send a distribution which is zero everywhere except at the observed value, where it is positive.

variable nodes are MULTIPLIERS

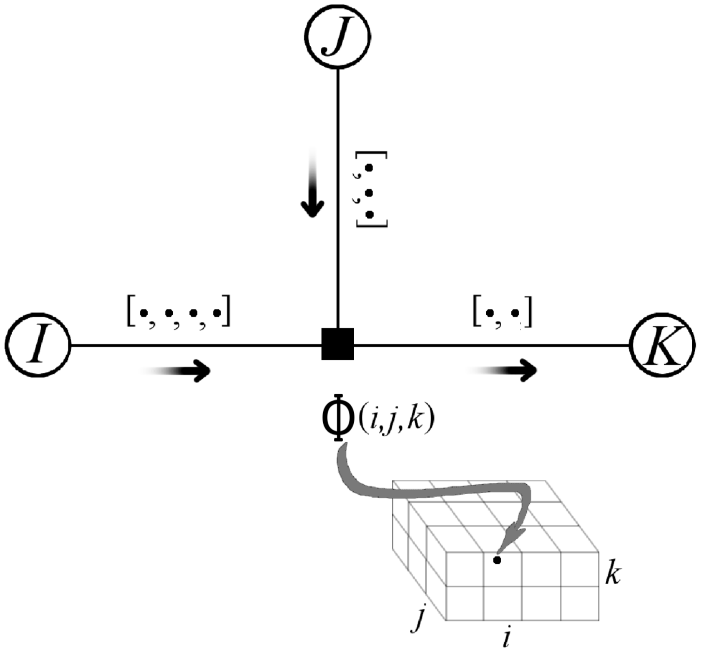
- To generate a message on an edge, they simply multiply the incoming messages on their other edges.
- If the variable is **observed**, they send a distribution which is zero everywhere except at the observed value, where it is positive.

When a variable node has all its incoming messages, it can calculate $p(x|\text{observations})$ by multiplying them together and normalising.

factor nodes are SUMMERS

Each dimension of Φ corresponds to one of its neighbours.
To generate a message on an edge,

1. *form a joint* distribution by taking the product (**) of the incoming messages on their other edges, weighted by the factor
2. *integrate out* the incoming variables from that joint



e.g. `calcMessage`(K):

$$m(i) = \sum_i \sum_j \texttt{msg}_I(i)\ \ \texttt{msg}_J(j)\ \ \Phi(i,j,k)$$

aside: the "meaning" of messages

In the special case of a Belief net the messages have easily interpreted meanings.

- messages passing in the "forward" direction are of the form $p(x, \text{obs})$ where x is the variable associated with the edge and obs is all observations *before* the node in question

aside: the "meaning" of messages

In the special case of a Belief net the messages have easily interpreted meanings.

- messages passing in the "forward" direction are of the form $p(x, \text{obs})$ where x is the variable associated with the edge and obs is all observations *before* the node in question
- messages passing in the "backward" direction are of the form $p(\text{obs}|x)$ where obs is all observations *after* the node in question

put pic in here

tractability

What kinds of distributions can we use, in practice?...

Outgoing messages should be no more complex than the incoming ones it's handy if they're easy to normalise, but not so crucial i.m.h.o...

tractability for MULTIPLIERS

want $f(\mathbf{x})$ such that $f_1(\mathbf{x}) \ * \ f_2(\mathbf{x})$ is of the same family

- **multinomial** - ok (element-wise multiplication)
- **Gaussian** - ok (product of Gaussians is Gaussian)

Others? Exponentiated polynomials:

$$\exp \sum_{i=0}^{n} a_i x^i \ * \ \exp \sum_{i=0}^{n} b_i x^i \quad = \quad \exp \sum_{i=0}^{n} (a_i + b_i) x^i$$

(n should be even)

and that's *it*, right?...

tractability for SUMMERS

- **multinomial** - ok (element-wise multiplication, then summing out)
- **Gaussian** - ok (joint Gaussian, then marginalise out all but one dimension, leaves a Gaussian)

Others?

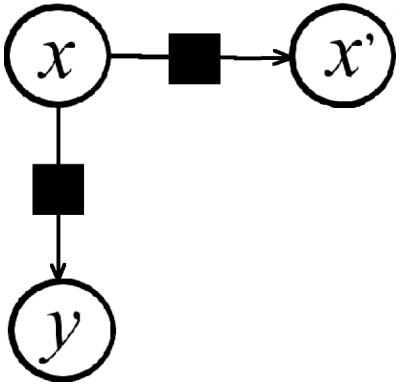
nb. you can give away exactness and go for particle filtering instead - best idea I've seen is to treat as mixture of Gaussians - product is silly, so resample

reinforcement learning problems

In RL we're interested in arriving at near-optimal controllers based only on reinforcement signals: you're not told what you should have done, just how good your action was. This is HARD due to

- hidden state (POMDPs)
- delayed rewards

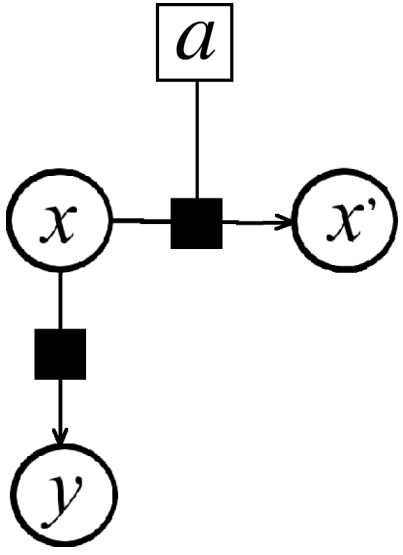
the standard HMM graph



There are just two factors:

- transitions (Markov)
- emissions

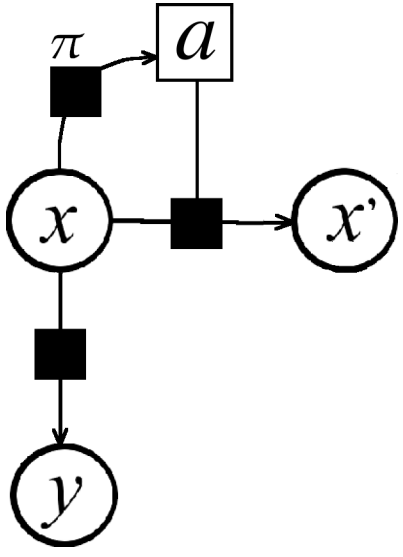
HMM with actions: 'Dynamic decision nets'



transitions now involve actions

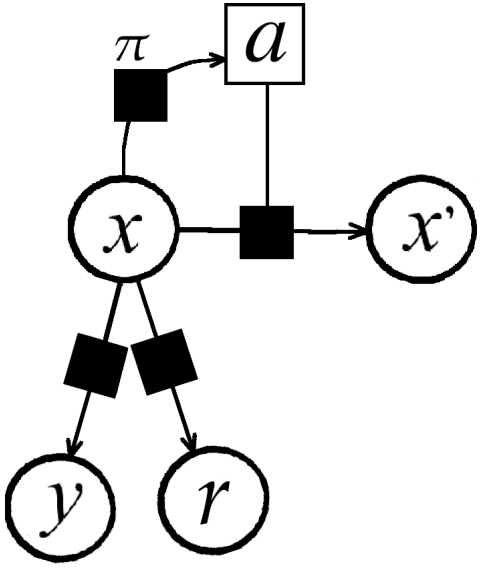
- can now concoct sequences of "optimal" actions

a reactive HMM



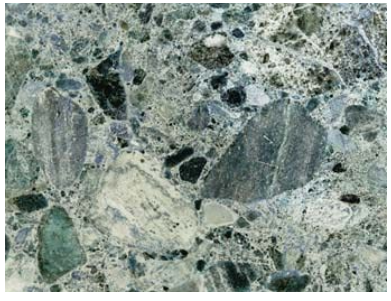
- actions derived from state estimates via policy π

reactive HMM that learns its policy



This model has a stationary distribution over states, and thus an expected value for r can be calculated, as well as it's gradient (*I think*). The policy could be improved by following this gradient...

action-centered representations



Emission and transition models can be trained to predict the sensors (HMM), but *should* be trained to **generate internal states that support optimal decision-making**

- Is there a way to do this *without* entering the Lurid Swamp of Ad Hockery and succumbing to the dark energies?