Lecture 8

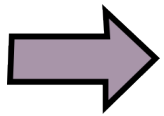
Approximate Inference

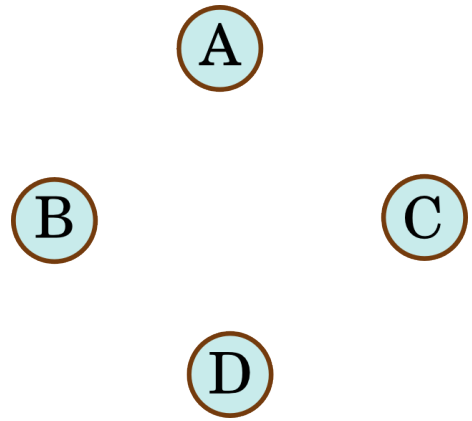
Highly Dependent Data

Approach 1: Model all the dependencies:

Data Distribution

a_1	b_3	c_2	d_1
a_3	b_2	c_4	d_2
a_5	b_1	c_1	d_3
.	.	.	.
.	.	.	.
.	.	.	.



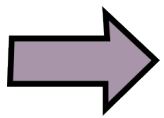


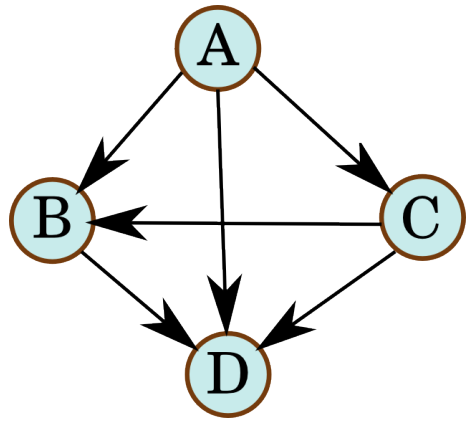
Highly Dependent Data

Approach 1: Model all the dependencies:

Data
Distribution

a_1	b_3	c_2	d_1
a_3	b_2	c_4	d_2
a_5	b_1	c_1	d_3
.	.	.	.
.	.	.	.
.	.	.	.



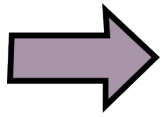


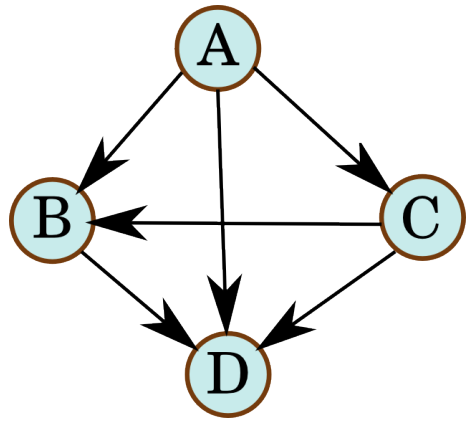
Highly Dependent Data

Approach 1: Model all the dependencies:

Data
Distribution

a_1	b_3	c_2	d_1
a_3	b_2	c_4	d_2
a_5	b_1	c_1	d_3
.	.	.	.
.	.	.	.
.	.	.	.





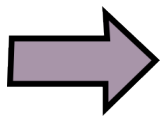
Propagating probabilities is difficult or infeasible!

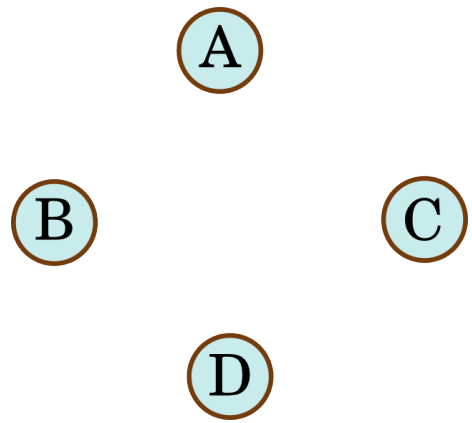
Highly Dependent Data

Approach 2: Find the maximally weighted spanning tree:

Data Distribution

a_1	b_3	c_2	d_1
a_3	b_2	c_4	d_2
a_5	b_1	c_1	d_3
.	.	.	.
.	.	.	.
.	.	.	.



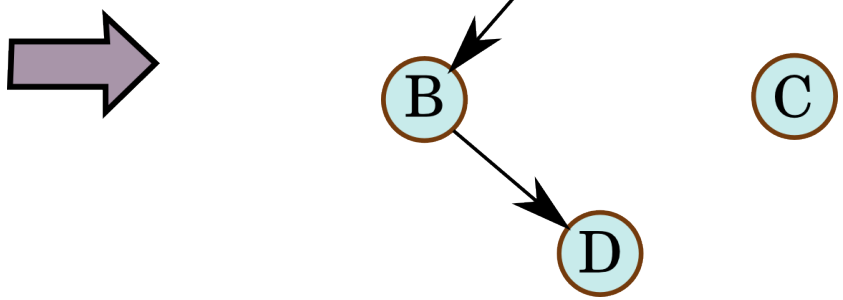


Highly Dependent Data

Approach 2: Find the maximally weighted spanning tree:

Data Distribution

a_1	b_3	c_2	d_1
a_3	b_2	c_4	d_2
a_5	b_1	c_1	d_3
.	.	.	.
.	.	.	.
.	.	.	.

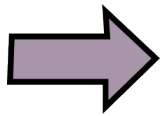


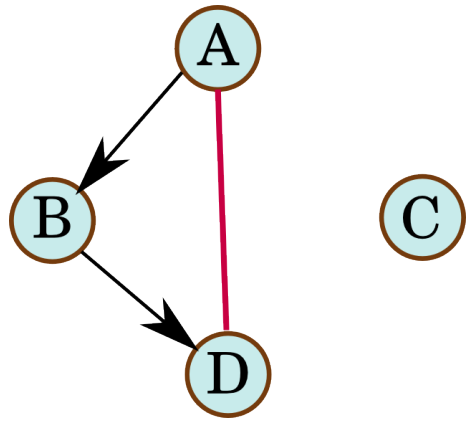
Highly Dependent Data

Approach 2: Find the maximally weighted spanning tree:

Data Distribution

a_1	b_3	c_2	d_1
a_3	b_2	c_4	d_2
a_5	b_1	c_1	d_3
.	.	.	.
.	.	.	.
.	.	.	.





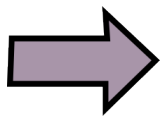
Loops are not allowed

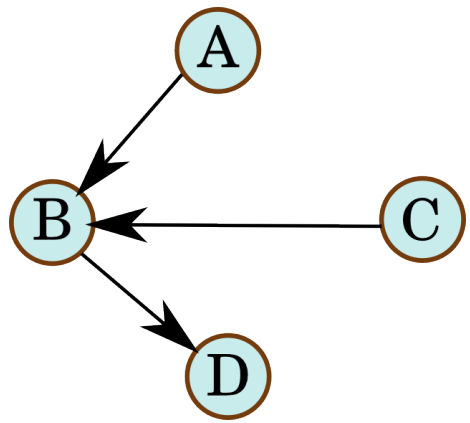
Highly Dependent Data

Approach 2: Find the maximally weighted spanning tree:

Data Distribution

a_1	b_3	c_2	d_1
a_3	b_2	c_4	d_2
a_5	b_1	c_1	d_3
.	.	.	.
.	.	.	.
.	.	.	.



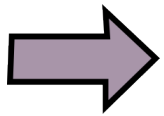


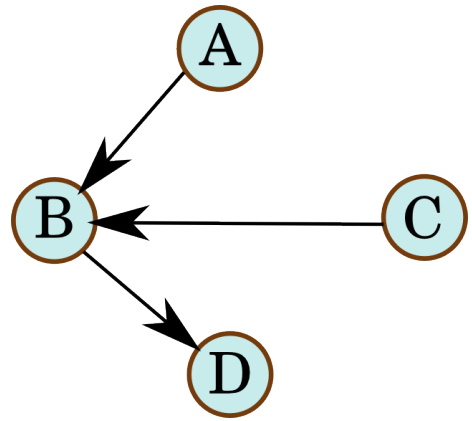
Highly Dependent Data

Approach 2: Find the maximally weighted spanning tree:

Data Distribution

a_1	b_3	c_2	d_1
a_3	b_2	c_4	d_2
a_5	b_1	c_1	d_3
.	.	.	.
.	.	.	.
.	.	.	.





Now the network does not model the dependencies accurately

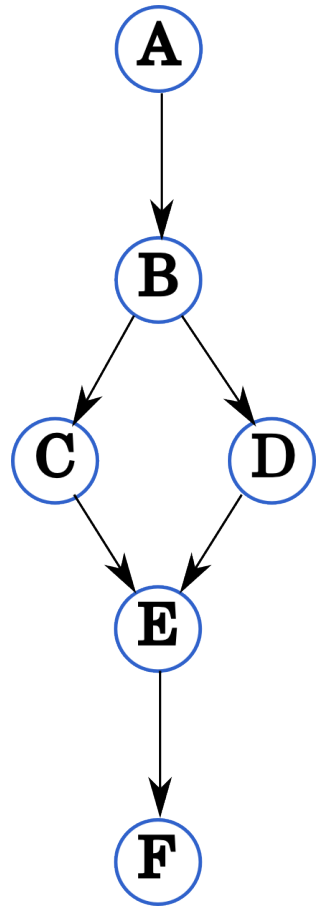
Exact and Approximate Inference

- If we include all dependencies then computation is exact, but can be computationally infeasible for large sized networks and large data sets. We will look at exact computation algorithms later in the course.

- If we choose a spanning tree then message passing terminates in one pass and is very fast, but the inference is only approximate.

Problems with Loops in Networks

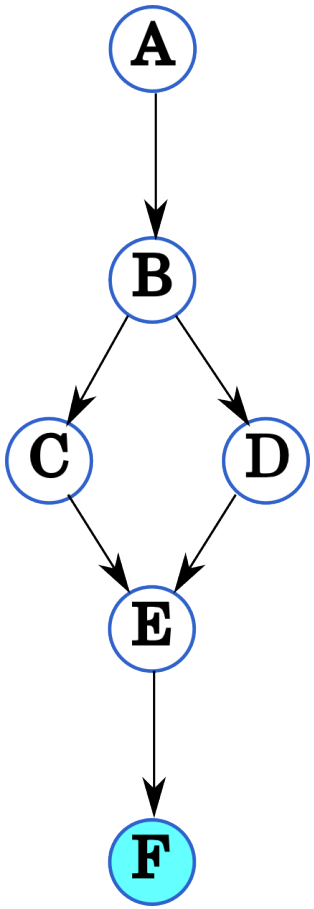
Issue 1: Looping Messages.



Problems with Loops in Networks

Issue 1: Looping Messages.

When only Node F is instantiated there is no condition that stops the messages travelling round the loop B C D E.

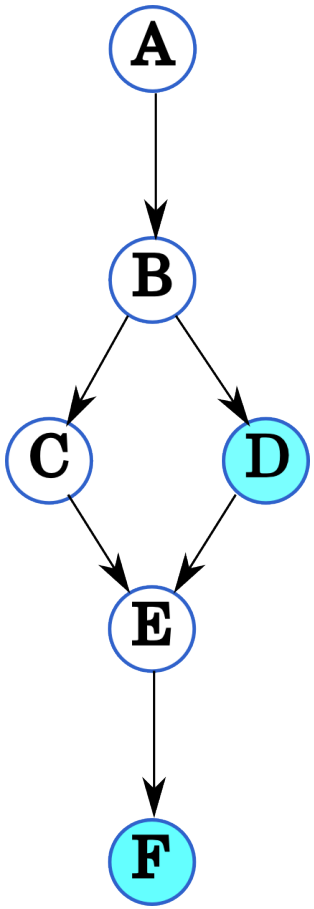


Problems with Loops in Networks

Issue 1: Looping Messages.

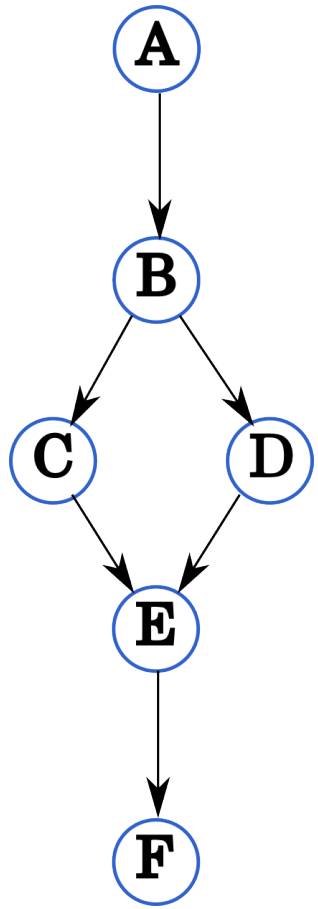
When only Node F is instantiated there is no condition that stops the messages travelling round the loop B C D E.

Exact propagation can still be carried out if one of nodes B C or D is instantiated



Problems with Loops in Networks

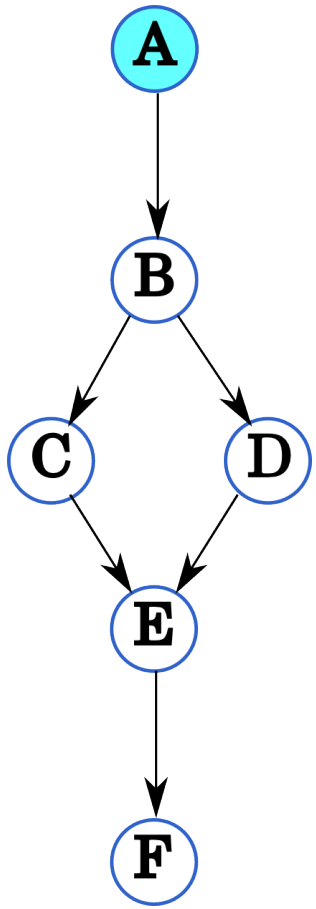
Issue 2: Independence of Multiple Parents



Problems with Loops in Networks

Issue 2: Independence of Multiple Parents

When only Node A is instantiated propagation terminates. However C and D are not independent and so the π evidence at E and F is not correct.

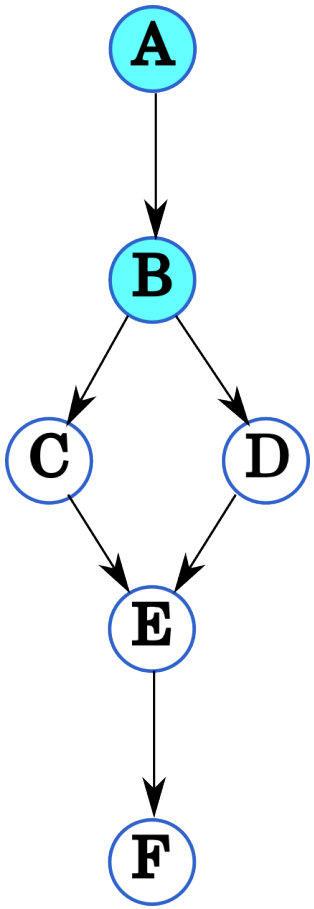


Problems with Loops in Networks

Issue 2: Independence of Multiple Parents

When only Node A is instantiated propagation terminates. However C and D are not independent and so the π evidence at E and F is not correct.

Exact propagation can still be carried out if one of nodes B C or D is instantiated. This will make C and D independent.



Approximate Inference Methods

Spanning trees and Naive Bayesian networks can be considered approximate inference methods They model the most important dependencies, though not all.

Their performance can be improved by a number of techniques including:

1. Node Deletion
2. Allowing Loopy belief Propagation
3. Hidden (or Latent) Node Placement

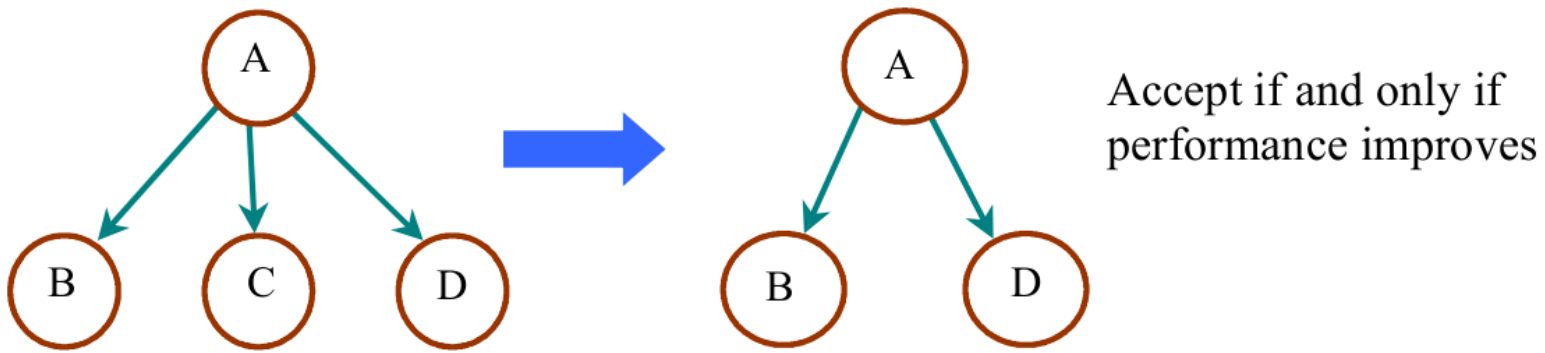
Node Deletion

- Given a pair of highly dependent nodes it has been found that deleting one sometimes improves a networks predictive performance.
- This is a surprising result from which it is difficult to infer any general rule.
- Node deletion is something that can be tested experimentally.

Selective Bayesian Network

The idea here is to use only a subset of the variables.

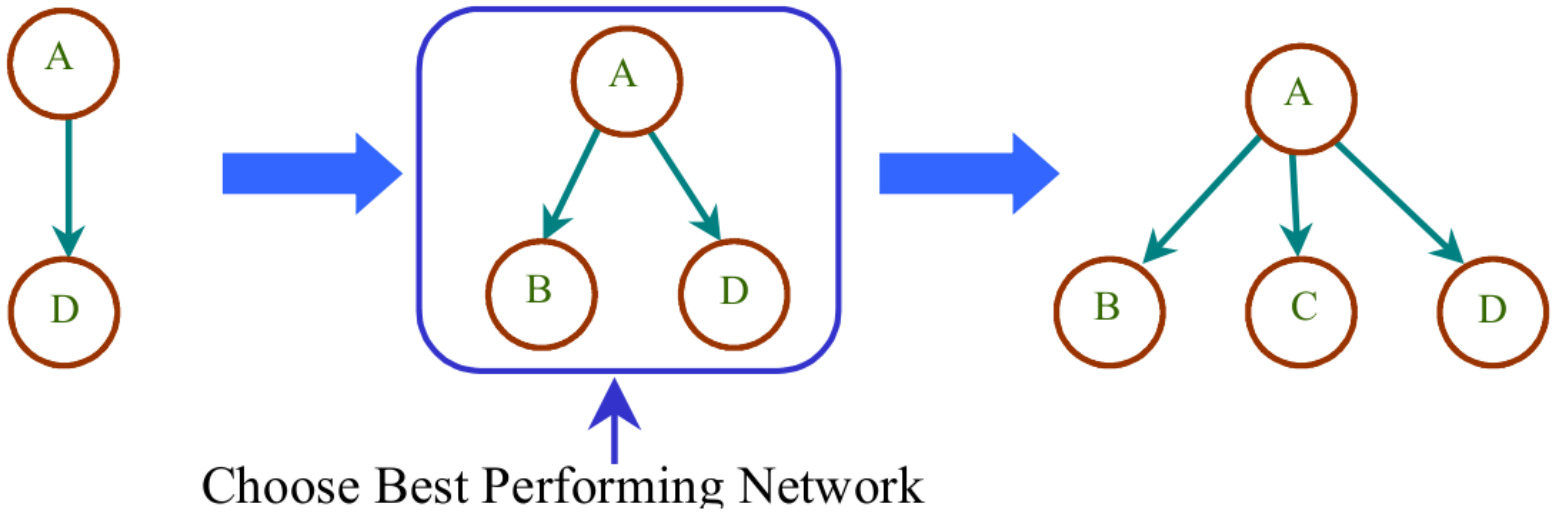
This can be done by starting with all the variables then deleting any suspect variable and testing for improvement in performance. Deletion continues until no further improvement can be found.



We can find suspect variables by testing pairs of children for high conditional dependence.

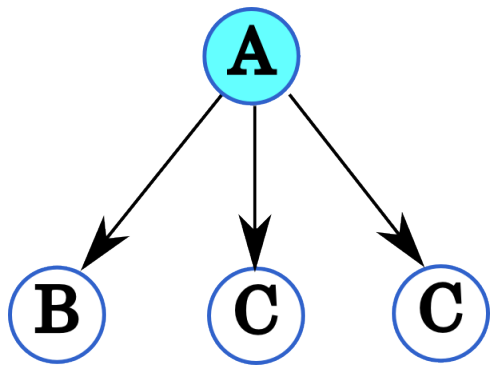
Selective Bayesian Networks

Equivalently we can add variables incrementally (in dependency order) and test the performance of each network.



Why can removing variables improve the performance

- Consider a Bayesian network where the same variable appears twice.
- Clearly conditional independence doesn't hold
- The network will be biased in favour of C
- Deleting C will improve performance



The improvement will depend on the quantity of unaccounted dependency between variables.

Loopy Belief Propagation

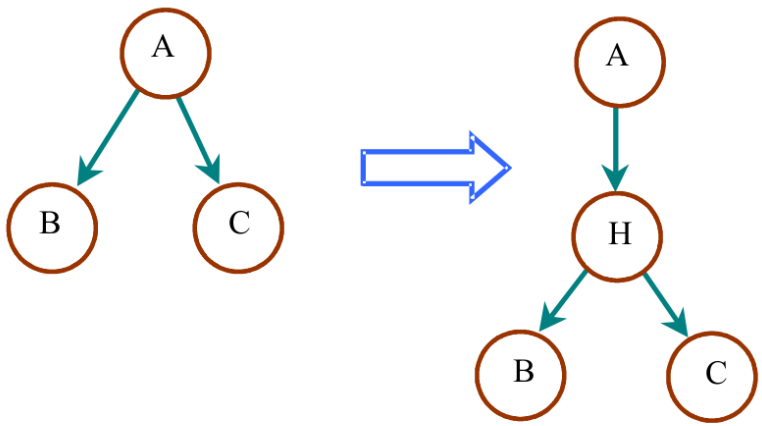
Another approximate method is to include all arcs expressing significant dependency, and allow propagation to continue until:

- The probability distributions reach a stable state, or
- A limiting number of iterations has occurred (there may be no termination)

Loopy belief propagation has been shown to be equivalent to a multivariate optimisation problem. It will most likely find a local optimum. We cannot say anything about its accuracy.

Hidden Nodes (or Latent Variables)

If any two children of a parent node are not conditionally independent, they can be separated by a hidden node:



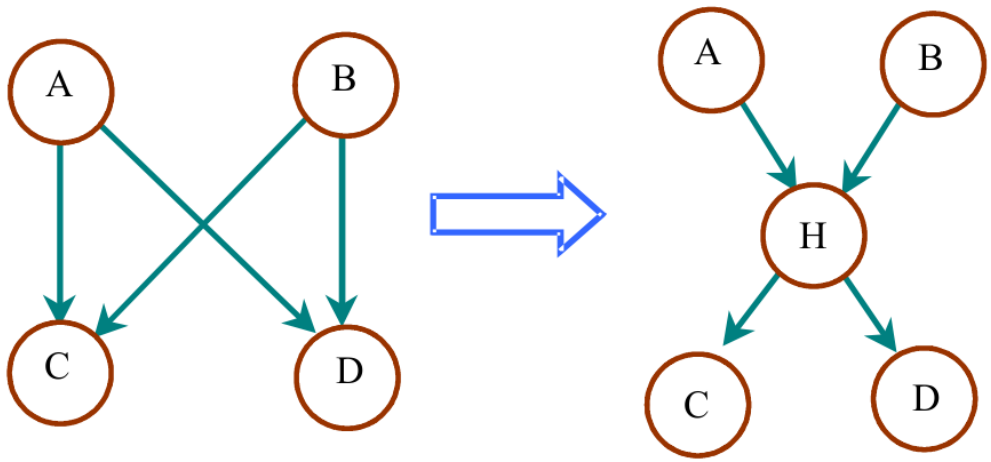
The new node represents a common cause that relates B and C. It is called hidden because we have no corresponding measured variable.

Now we look at how to obtain it statistically.

Switch Nodes

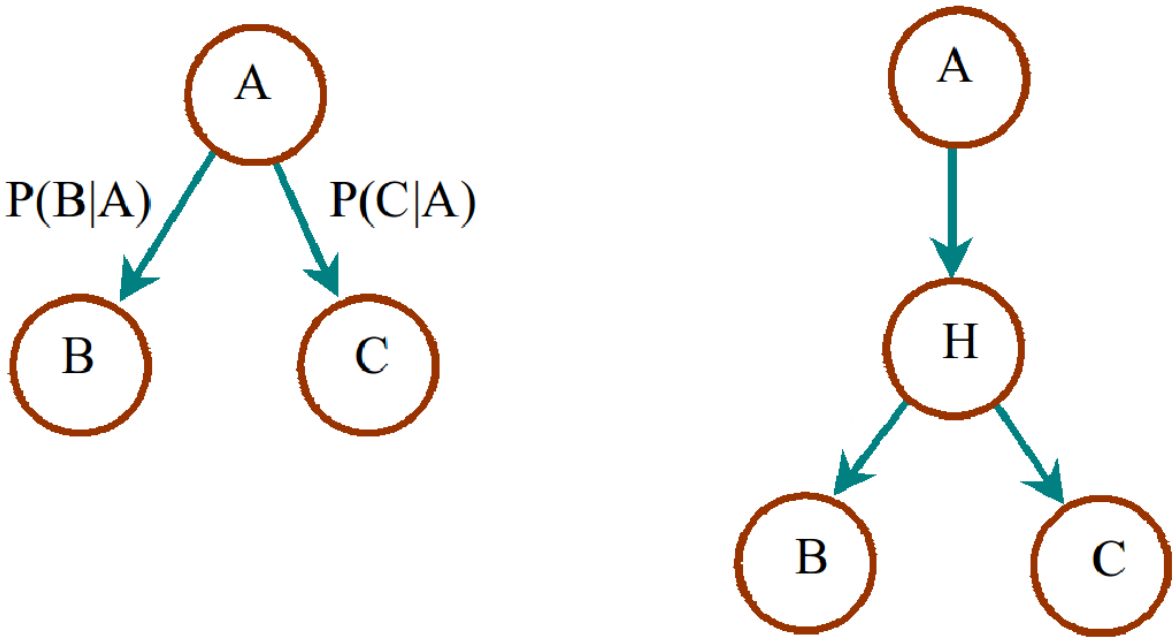
Adding hidden nodes which act as switches can simplify complex networks.

Example from Neopolitan:



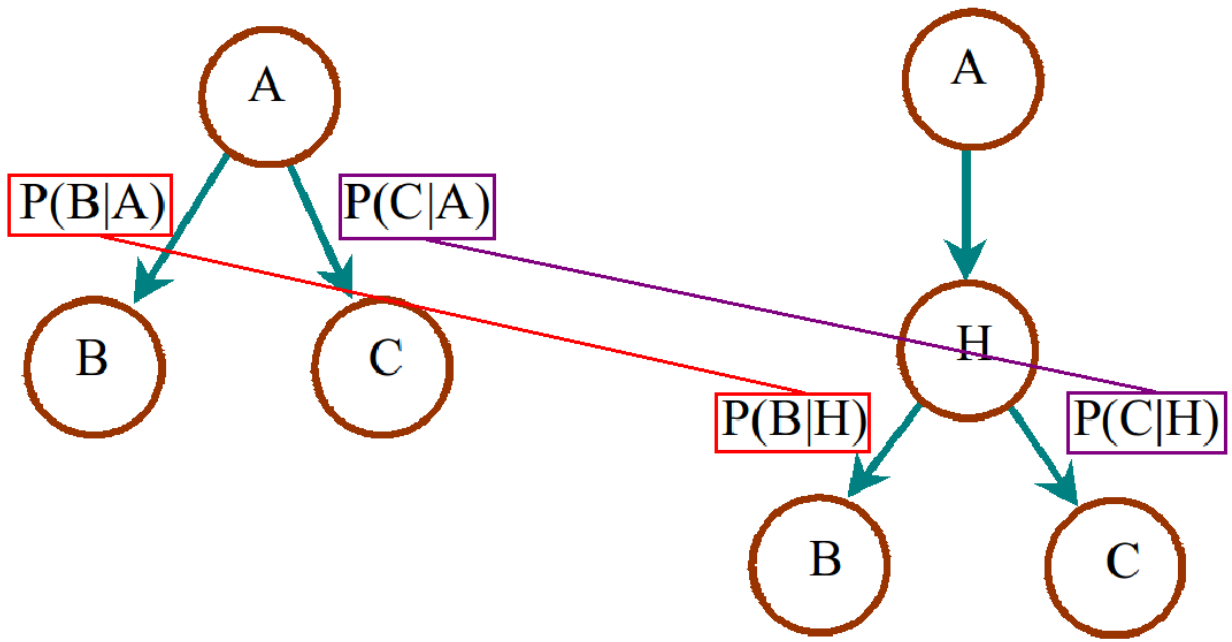
Advantages of Adding Hidden Nodes

A network can always perform as well with a hidden node as it can without:



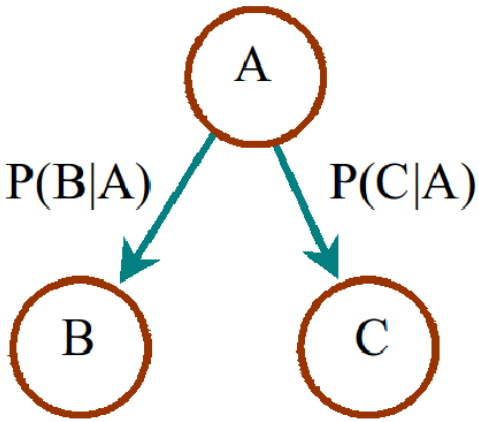
Advantages of Adding Hidden Nodes

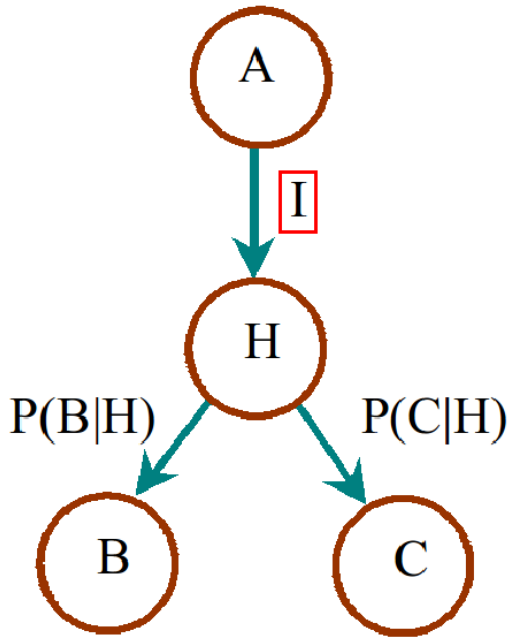
A network can always perform as well with a hidden node as it can without:



Advantages of Adding Hidden Nodes

A network can always perform as well with a hidden node as it can without:





Using Hidden Nodes

In order to create a hidden node we need to:

1. decide how many states the hidden node is to have;
2. identify values for the three new link matrices introduced.

It may be possible to obtain hidden node information from an expert (eg the eyes example from lecture 2). For example an expert may:

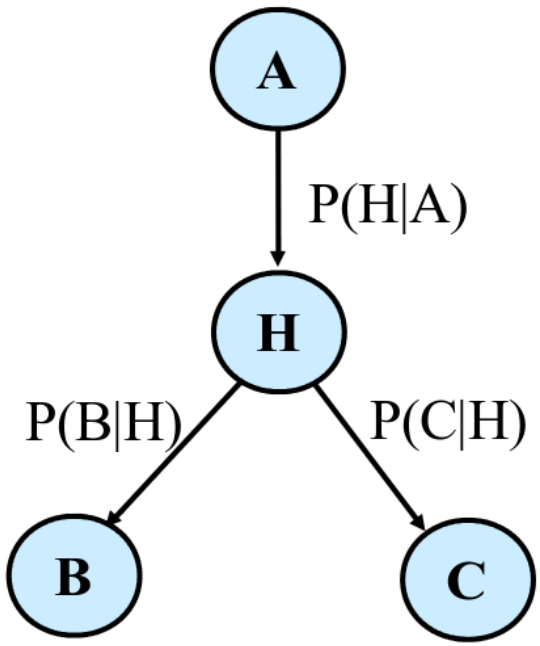
1. identify a variable corresponding to the hidden node;
2. provide data for training (ie calculating the link matrices).

In general however this is not often possible.

How Many States?

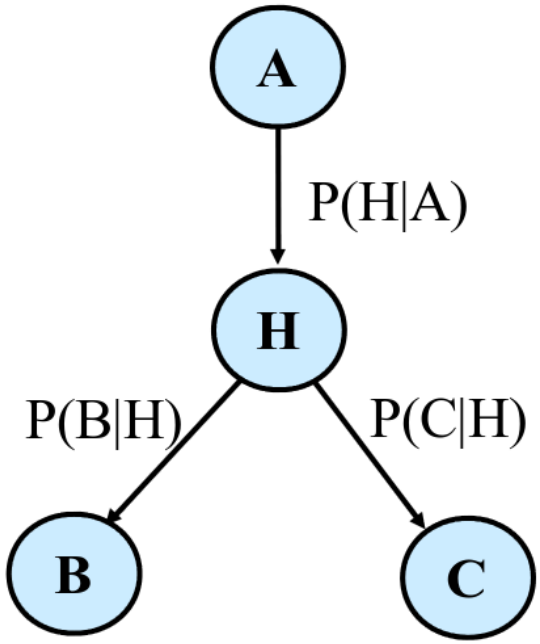
- We expect that the number of states of a hidden node will be comparable to the number of states of the nodes its is separating.
- From the previous slides we would expect the hidden node to have at least the same number of states as its parent.
- Link matrices with too many states will have very low probabilities for some states, so a possible approach is to sart with a large number of states and reduce the number depending on how many low probability states we have.

Calculating the Conditional Probabilities



1. Given estimates of: $P(H|A), P(B|H), P(C|H)$ and a set of data points $[a_i, b_j, c_k]$
2. Use each b_j, c_k to compute $P'(A)$ from the network, calculate and accumulate an error:
$$E = (P'(A) - P(a_i))^2$$
3. Minimise E over the data set by adjusting the elements of $P(H|A), P(B|H), P(C|H)$

Calculating the Conditional Probabilities



For each conditional probability $P(c_j|h_k)$ we need to find a value for:

$$\frac{\partial E}{\partial P(c_j|h_k)}$$

Then in each epoch we update the conditional probabilities using:

$$P(c_j|h_k) \Rightarrow P(c_j|h_k) - \mu \frac{\partial E}{\partial P(c_j|h_k)}$$

Gradients may be calculated analytically or numerically. A closed form equation for the gradients was developed by Chee Keong Kwoh.

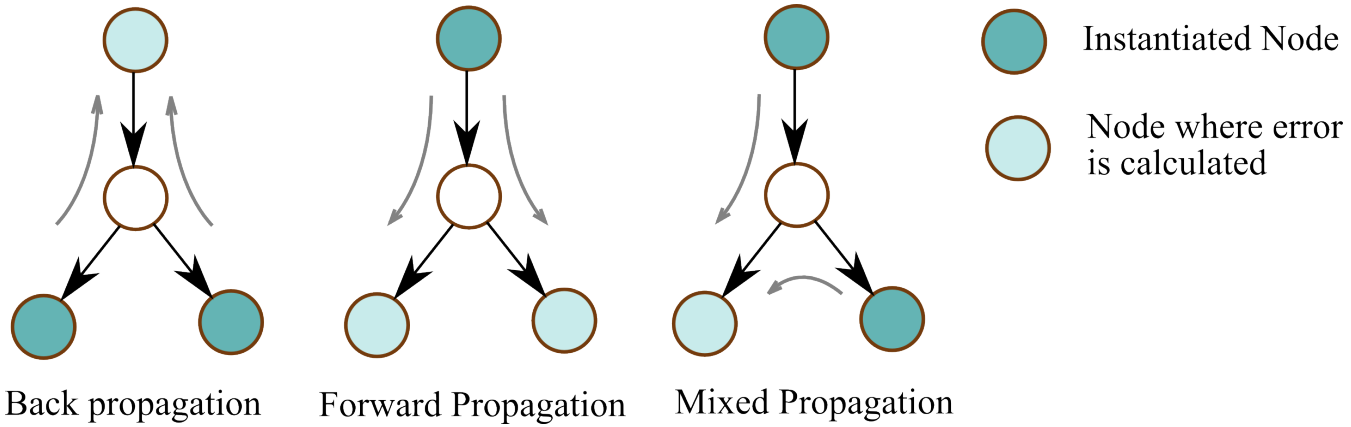
Gradient Descent and Probabilities

Gradient descent has problems when applied to probability distributions. After one cycle of updating:

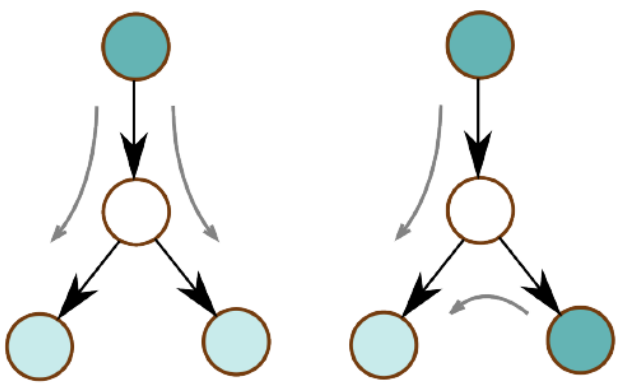
- Distributions will no longer sum to 1
- Individual probability values may be greater than 1 or less than 0

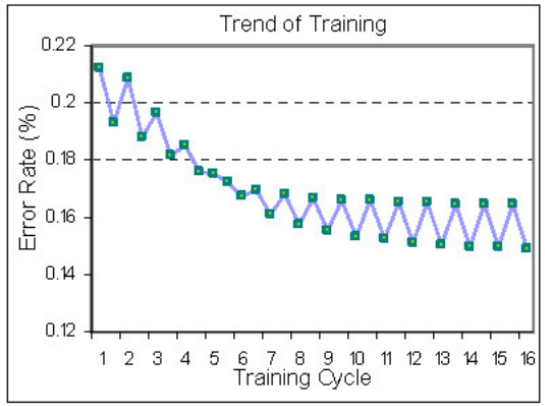
The conditional probability matrices must be normalised so that the columns sum to 1. This may compromise finding an optimal solution.

Propagation Strategies for calculating errors



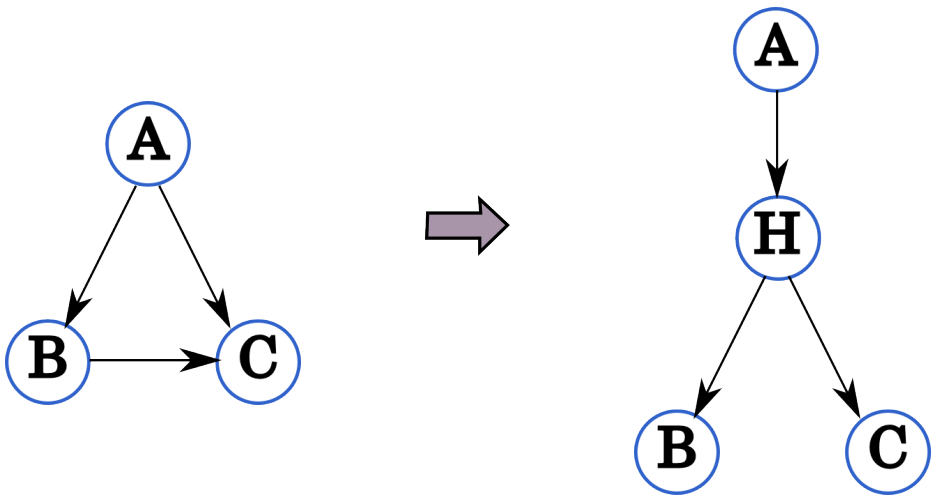
Strategies may be alternated during the optimisation and this produces annealing behaviour:





Hidden Nodes for Removing Loops

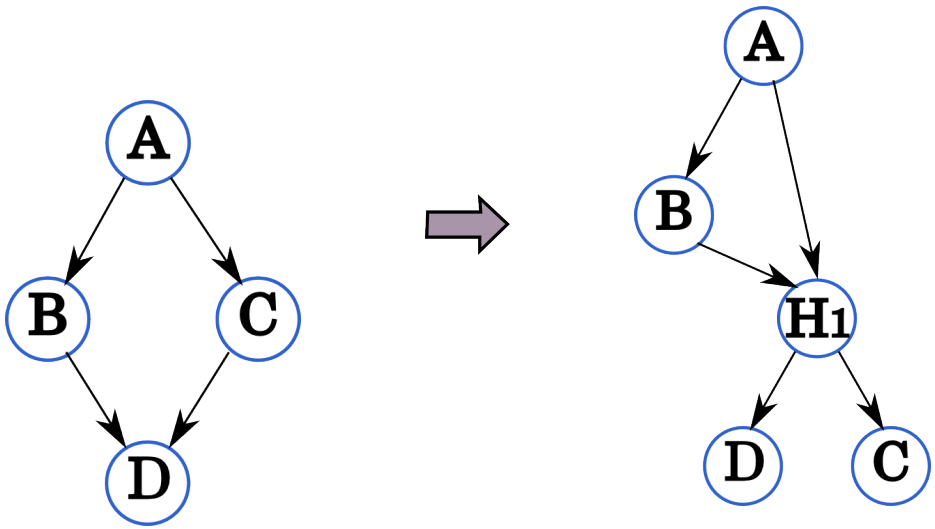
Suppose we build a network including all the dependencies, we can then use hidden nodes to remove any loops that were formed. In the case of the simple triple we have seen that:



The process is to remove the least dependent link of a multiple parent.

Reducing bigger loops

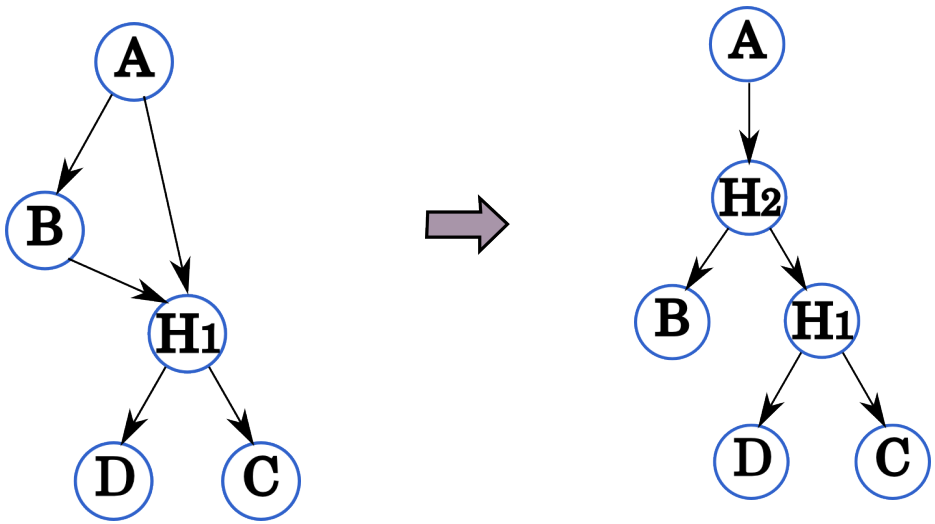
We can apply the same process to bigger loops. Modelling the dependency between C and D we get:



The training methods still work since for any instantiation of A or B the probability propagation will finish.

Reducing bigger loops

We can continue by modelling the dependency between B and H_1:



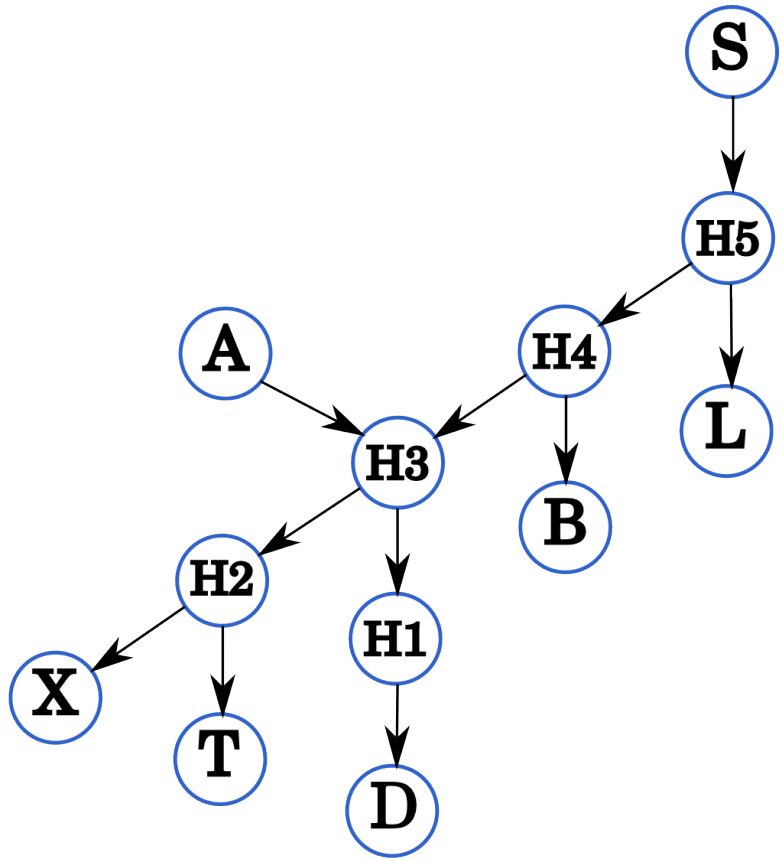
This results in a singly connected network but with two hidden nodes.

Reducing bigger loops

We can always reduce any network a singly connected form by this method. One possible form of the Asia network is:

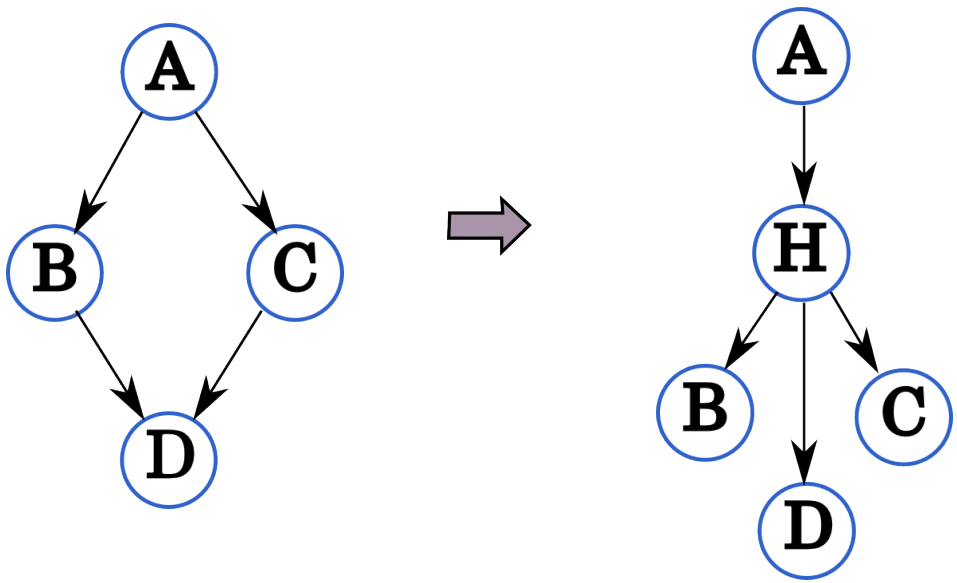
However, the large number of hidden nodes makes the method look less attractive.

The performance will become increasingly dependent on the training data and training process.



Reducing bigger loops

Could we simplify things by combining our two hidden nodes into one?:



The answer to this is very much data dependent. Clearly the hidden node now has to model the dependency between *B* and *C* that comes through both the common parent *A* and the common child *D*.

Limitations of the Hidden Node Method

There is clearly going to be a limit to the degree to which we can model dependencies through hidden nodes. As the dependencies become more complex, either:

1. We will need many hidden nodes, or
2. The number of states in the hidden node will become very large

In either case we may not have enough data to train the new network.

Criteria for Introducing Hidden Nodes

Given a network we can measure the conditional dependency of each pair of children given the parents.

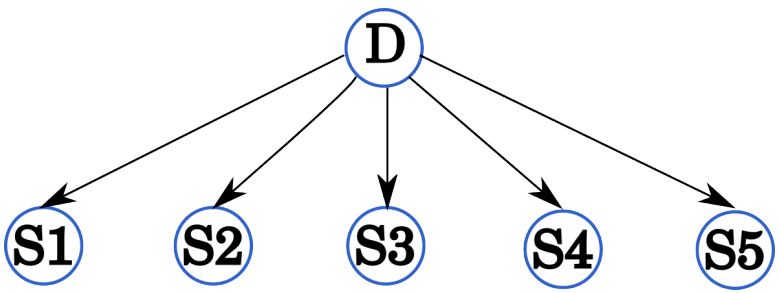
If this is high we expect that benefits will occur from introducing a hidden node.

However below a certain threshold we are unlikely to benefit from a hidden node and may choose to ignore the dependency.

Other Hidden Node Methodologies

Other more heuristic methods have been suggested for employing hidden nodes.

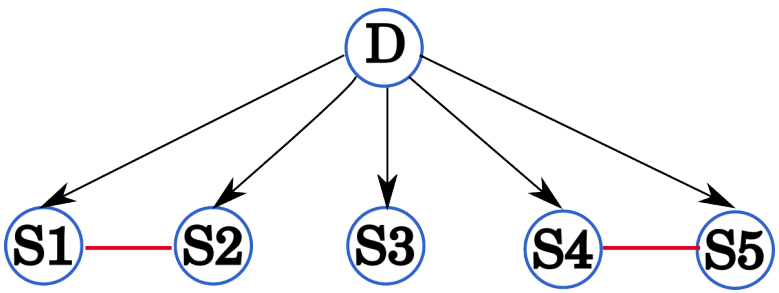
Starting with a naive network:



Other Hidden Node Methodologies

Other more heuristic methods have been suggested for employing hidden nodes.

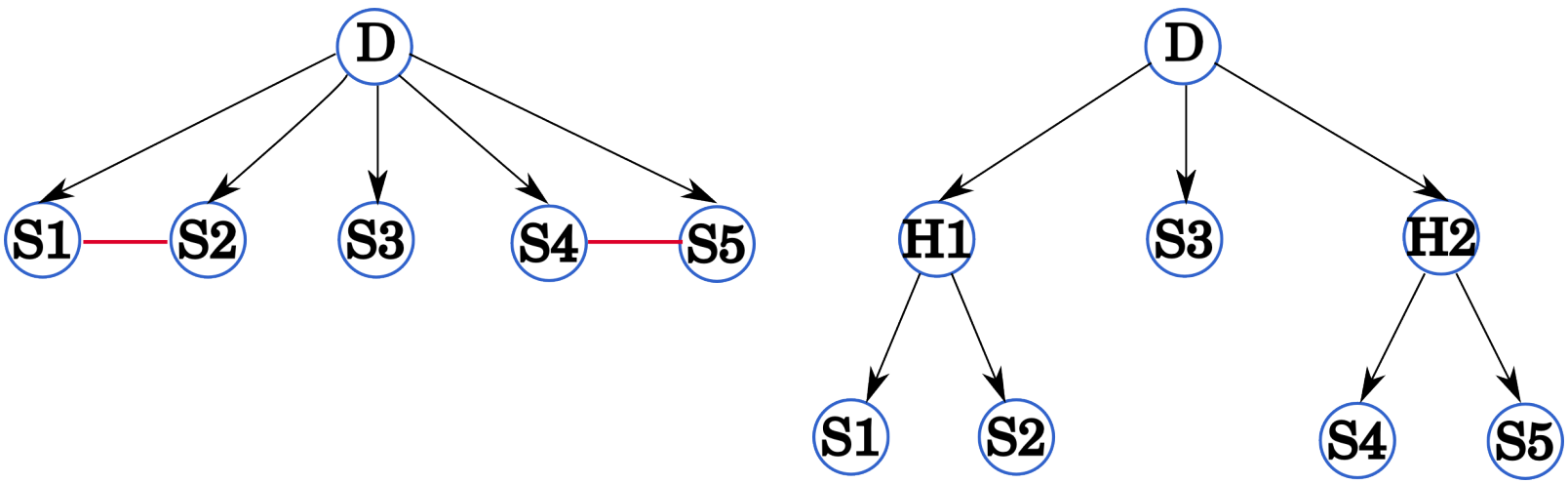
Find all significant conditional dependencies:



Other Hidden Node Methodologies

Other more heuristic methods have been suggested for employing hidden nodes.

Model them with hidden nodes:



Other Hidden Node Methodologies

A similar idea can be applied starting with a spanning tree;