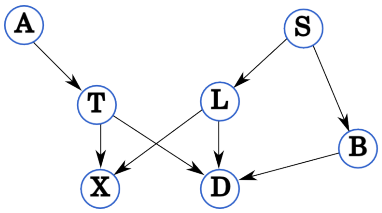
Lecture 9: Exact Inference

Cutset Conditioning

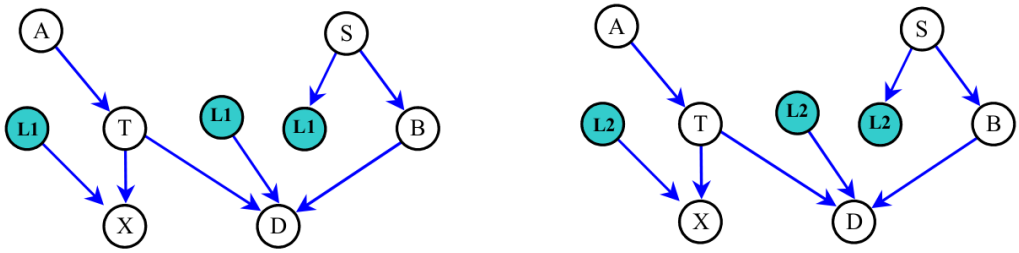
Pearl suggested a technique called Cutset Conditioning to deal with the problem of propagation in multiply connected networks. The idea of this is to find a minimal set of nodes whose instantiation will make the remainder of the network singly connected and therefore safe for propagation. For example, in tutorial 3 we looked at the following causal Bayesian net for dyspnea.



A : Asia
B : Bronchitis
D : Dyspnea
L : Lung Cancer
S : Smoker
T : Tuberculosis
X : XRay

This has two loops, the first being round nodes DBSL and the second involving DLXT. Notice that the direction of the arrows in not significant in considering the effect of loops, since probabilities propagate both in the direction of the arrows (π messages) and against them (λ messages). For certain instantiations propagation can be achieved. For example if D is not instantiated then propagation between all the other nodes is possible, though it may not give the correct result in this case. However, if D is instantiated probabilities could propagate around either loop unchecked.

Looking at the network, we see that instantiating node L effectively blocks both loops. L is a single parent node, and therefore if it is instantiated any change in S is blocked from X and D and vice versa. With L instantiated we can think of the network as belonging to one of two singly connected possibilities:

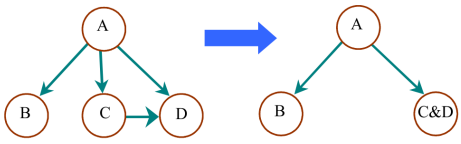


If we now are presented with data which does not include an instantiation of L, but does include an instantiation of some other nodes, for example D, we can calculate the probabilities using both networks, and then combine them by weighting them according to the prior probability distribution over L. This is a safe, and correct propagation technique which will always work, but it does rely on us having an estimate of the prior probabilities of node L.

There are problems with cutset conditioning. First, it explodes computationally with the size of the cutset. For example, if the cutset has three nodes each with eight states, then we will need to propagate the probabilities for each of the 512 possible instantiations. Secondly, and more importantly, we need a reliable estimate of the prior probability of each of those instantiations. This may not be reliably obtainable from the data.

Joining dependent variables

Joining variables is another way of reducing dependencies. If C and D are dependent (given A) we can combine them into one new variable by joining them. If two variables are to be joined, C having $|C|$ states and D having $|D|$ states the new variable $C\&D$ will have $|C| \times |D|$ states. The increased number of states is undesirable and limits the applicability of this approach.

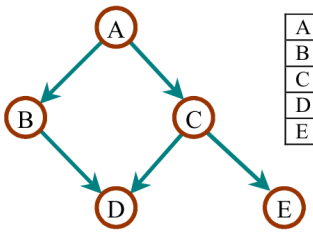


Join Tree Algorithm

A generalisation of the idea of joining nodes is the creation of a join tree. The idea is to replace a multiply connected network network by a singly connected network in which the nodes are not represented by single variables, but by the junction of several variables.

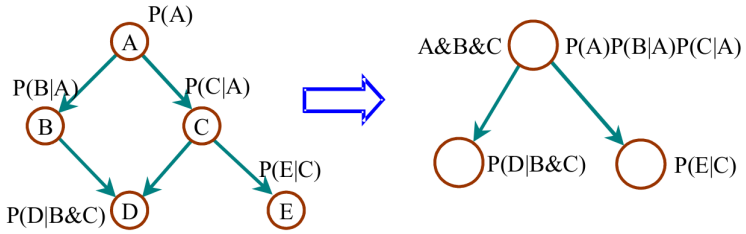
We have mentioned several times that, in the spanning tree algorithm we added the arcs in the order given by the dependency between nodes. However for a set of n variables there are $n!/(2!(n-2)!)$ possible arcs of which only $n-1$ will be used. As the graph gets bigger there will be many unused arcs. Statistically, there is very likely to be some significant dependency for these arcs. To make a network represent the data dependencies more correctly we could add arcs whenever there is significant dependency. However, adding one arc to a tree or a singly connected network will make it multiply connected. Thus, if we could devise a way of propagating probabilities in multiply connected networks we could (arguably) build a better inference system. A famous algorithm for coping with probability propagation in multiply connected, but sparse, graphs was developed by Lauritzen and Spiegelhalter. Unlike the singly connected algorithm of Pearl it has no notion causality. It is purely a mathematical technique.

The basic idea is to transform the network to a different representation where propagation is possible. The new representation is called a join tree or a tree of cliques. We shall use a simple medical example, which is taken from Neapolitan.



A	a1 a2	Metastatic Cancer
B	b1 b2	Total Serum Calcium Increased
C	c1 c2	Brain Tumour
D	d1 d2	Comna
E	e1 e2	Papilledema

The nodes are all binary variables, and the causal graph (derived from expert advice) has one loop, meaning that propagation is not possible when D is instantiated, but A,B and C are not. An example of a join tree is shown below. It has the same joint probability distribution as the original tree, but has only three nodes, each containing a subset of the original variables. We will use the join tree to propagate probabilities, using a revised algorithm, and then calculate the probabilities of the original individual variables.



Potential Representations

In the above example we chose one way to group the variables to form a join tree. This could be done in a number of ways and each is called a potential representation. Suppose we have a causal network with set of variables V, in the example above $V = [A, B, C, D, E]$, then a potential representation of a causal network is a number of subsets W_i of our variables, and a function Ψ with the property:

$$P(V) = \prod_i \Psi(W_i)$$

Each subset W_i in our potential representation is a node of the join tree, and contains one or more variables of the original network. The function Ψ is alled the potential function (or table) and assigns a probability to each joint state of the variables belonging to W_i. It is similar to a joint probability table, and will be used to compute the joint probability table $P(W_i)$ for the node.

The running intersection property

For a potential representation we can define some ordering and intersection sets. We will adopt the convention that the indices i of the subsets of the variables W_i are ordered with those nearest the root having the lower value, so W_1 is used to denote the root of the join tree, and W_2 and W_3 the child nodes in the above example. For each W_i we define a subset S_i which is all the varibales of W_i that have appeared higher up in the tree. Formally we write:

$$S_i = W_i \cap (W_1 \cup W_2 \cup W_3 .... \cup W_{i-1})$$

We use R_i to denote the remaining variables:

$$R_i = W_i - S_i$$

Thus the variables in the set R_i are those that have not appeared higher up in the tree.

To be able to propagate probabilities in a join tree we need to ensure that the running intersection property holds everywhere. This states that the variables belonging to every subset S_i must appear in the parent of node i in the join tree. So if j is the parent of i:

$$S_i \subseteq W_j$$

Given a join tree in which the running intersection property holds we can write:

$$P(V) = P(W_1) \prod_{i=2}^{N} P(R_i|S_i)$$

This property allows us to compute and pass λ and π messages among the groups of variables. We will look in detail at how this is done in the next lecture.

Finding a Join Tree

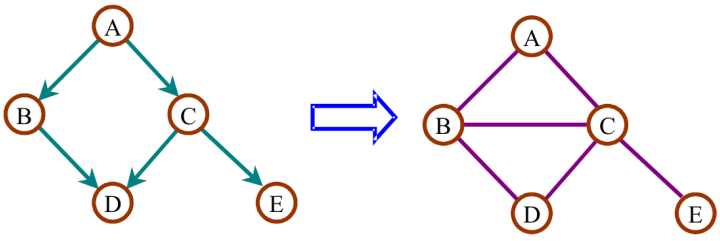
We now consider the question of how to construct a join tree starting from any given causal network. This is always possible using an algorithm which has five steps:

1. Moralise the graph (join any unjoined parents)
2. Triangulate the graph
3. Identify the cliques of the resulting graph. These form the subsets W_i.
4. Order the cliques following the running intersection property.
5. Initialise the clique potential functions ψ as follows: For each clique W_i find the subset of its variables X_i whose parents $Pa(X_i)$ also belong to the clique. Then set:

$$\Psi(W_i) = \prod_{X} P(X_i|Pa(X_i)) \tag{1}$$

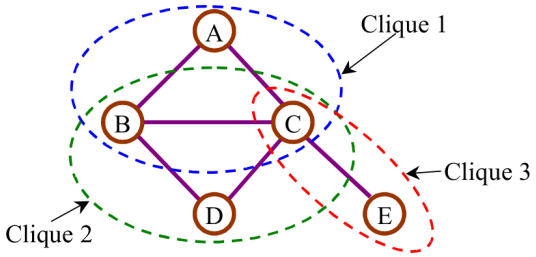
Moralising and Triangulation

We first convert the causal graph by joining together multiple parents of a node (moralisation) and then adding arcs to convert any polygon configurations to triangles. In this example the moral graph is already triangulated and needs no further processing. If the resulting graph were not triangulated then we would need to add internal arcs to make it so. There is no unique qay of doing this, but the algorithm will work for any triangulated graph. Note that the triangulated graph is no longer a causal structure, and consequently we can drop the arrows. There is no implication that there is any dependency expressed by the new arcs.



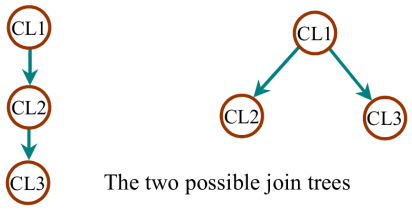
Finding the Cliques

A clique is a maximal set of nodes in which every node is connected to every other. In our example there are three such groupings. Notice that $[B, C]$ is not a clique, since it belongs to $[A, B, C]$ and so is not maximal, and $[A, B, C, D]$ is not a clique since A is not connected to D. Each clique will be a node in the final join tree, and its variables form the sets W_i.



Finding the structure of the join tree

We now need to order and index the cliques such that they form a join tree that conforms to the running intersection property. This can be done by inspection of the cliques found in the previous step, and again there is no unique solution. The ordering given in the figure above clearly obeys th running intersection property, and we can configure the tree in two possible ways.



Having established the ordering we can calculate the S and R sets for each node.

Allocating potential to the nodes of the join tree

Each variable of the original network must be in the R set in exactly one node of the join tree. Moreover its parents in the original network, must be in the same clique as a consequence of the moralisation step. Thus the R set or each clique can be found easily, noting that a root node of the original network has no parents, and so is in the R set of the clique in which it appears. Having found the R sets we define the potential functions as the product of the prior or conditional probabilities with distributions over the variables in the R sets of each clique. The result is as follows:

Index	W	S	R	X	Ψ
1	ABC		ABC	ABC	$P(A)P(B\|A)P(C\|A)$
2	BCD	BC	D	D	$P(D\|B\&C)$
3	CE	C	E	E	$P(E\|C)$

Note that the X set may be empty for some nodes. If that is the case the potential table is initialised to all 1s.