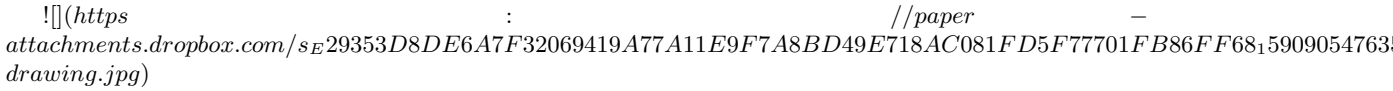
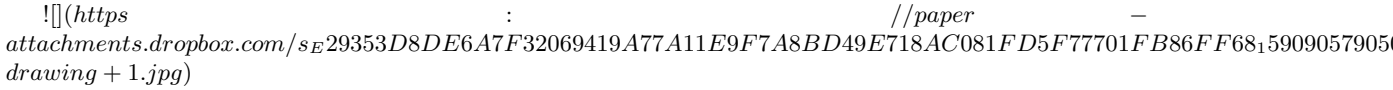
We wish to learn both the parameters and structure of a Bayesian Network.

Parameters

Consider the disease model from the previous lecture:



Cases 1 and 3 have incomplete data, whereas 2 had **complete**.
If every example is complete, the data set is called complete.
If complete, the maximum likelihood parameters are unique.
We find the likelihood of a parameter set like so:


!$[](https://paper-attachments.dropbox.com/s_E29353D8DE6A7F32069419A77A11E9F7A8BD49E718AC081FD5F77701FB86FF68_1590905790$
drawing + 1.jpg)

Param $\{S_1\} \rightarrow BN_1 \rightarrow Pr_1(.)$
Param $\{S_2\} \rightarrow BN_2 \rightarrow Pr_2(.)$
.
.
.
We pick the set of parameters that maximizes the probability,
so score $S_i = \prod Pr_i(e_i)$ is the likelihood of parameters

Parameter Estimation With Complete Data

!$[](https://paper-attachments.dropbox.com/s_E29353D8DE6A7F32069419A77A11E9F7A8BD49E718AC081FD5F77701FB86FF68_1590906565$
$drawing + 2.jpg)$
The large table is our empirical distribution .
This is formed by collapsing our distribution.
$\theta_{\{}\neg S|H\} = \frac{Pr(\neg s,h)}{Pr(h)} = \frac{5}{6}$.
This is our maximum likelihood parameter estimate!

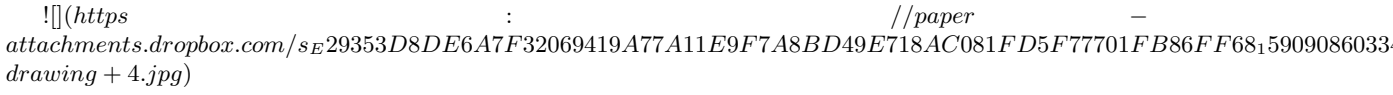
Parameter Estimation With Incomplete Data

This uses an iterative algorithm called **expected maximum**.
Say we wish to fill the row <h, s, ?>.
We start with an arbitrary guess at the CPT.
$\text{CPT}_1 \rightarrow \text{BN}_1 \rightarrow Pr_1(.)$.
We must choose from {TFT, TFF}, so we assign probabilities according to the probability function
x = Pr1(e|h, ¬s)
y = Pr(¬e|h, ¬s)
We then take the assignment with the greater probability, yielding
$\text{CPT}_2 \rightarrow \text{BN}_2 \rightarrow Pr_2(.)$.

We only needed to surmise one term here, but we may need to iterate.
The probabilities Pr1(.) & Pr2(.) are guaranteed to be non-increasing, so this converges.
This is thus a form of local search algorithm, so we know we may have to run repeatedly.
Note: this is the thinking behind the original algorithm, but it isn't how it is performed in practice.

Structure

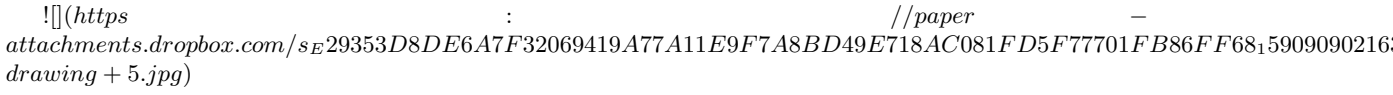
Consider the following three structures:


!(https : //paper −
attachments.dropbox.com/sE29353D8DE6A7F32069419A77A11E9F7A8BD49E718AC081FD5F77701FB86FF681590908603
drawing + 4.jpg)

What do I need to optimize to choose between these three structures?
We have learned many of these algorithms, so we won't discuss in detail, but:

1. local search methods (approximate methods)
   → we transform the structure looking for a better score
   We must thus specify movement within the neighborhood structure: → legal operations: add/remove/reverse edge
2. systemic search methods (exact methods)
   → A* is a good example

Why can't we just use maximum likelihood? We face **overfitting**.
Say we are using likelihood to compare; then C > B > A.
We will thus end up with a complete DAG, no matter what.
Thus we need a model that balances structure complexity with likelihood.
Why? Consider:


!(https://paper-attachments.dropbox.com/s_E29353D8DE6A7F32069419A77A11E9F7A8BD49E718AC081FD5F77701FB86FF68_1590909021631_drawing+5.jpg)

The data is clearly a linear fit, but if we estimate to the fourth degree, we get bad data!
There is no fixed answer to the problem; we clearly need some function of the form f(likelihood) - g(complexity).
A common one is called MDK, but we need not discuss details.

Model-Oriented Vs. Query-Oriented Learning

This can also be referred to as (unsupervised vs supervised) or (unlabeled vs labeled).
We have done model learning now; we move on to query based.

A model-based approach might give us the following:

$$

Say we promise only to infer upward:

Then we don't 'model' the system and instead prepare for a specific 'query'.

Labeling of correct responses is done by humans, and this is done to 'train'.

If we want to talk about supervised learning, we need to introduce

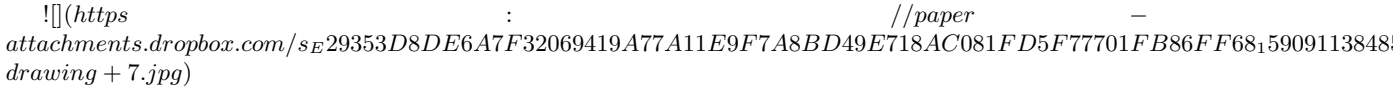
Arithmetic Circuits

!(https : //paper −
attachments.dropbox.com/sE29353D8DE6A7F32069419A77A11E9F7A8BD49E718AC081FD5F77701FB86FF681590910293
Shot + 2020 − 05 − 31 + at + 12.30.28 + AM.png)

The (+) symbols represent OR gates
The (*) symbols represent AND gates
The θ are the parameters
The λ are the evidence

When given a query, we can perform weighted model counting on the equivalent arithmetic circuit.
If A=True, $\lambda_A = 1$; if A=False, $\lambda_{\neg A} = 1$; if unknown, both are 1
Thus we can evaluate this in linear time;u nfortunately converting to an arithmetic circuit is $O(nd^w)$

BUT what do we do if we don't have the parameters? We use the labeled data:


!(https : //paper −
attachments.dropbox.com/sE29353D8DE6A7F32069419A77A11E9F7A8BD49E718AC081FD5F77701FB86FF681590911384
drawing + 7.jpg)

Cross Entropy gives us a measure of the disagreement between the two.

Therefore, we often seek to minimize it across a structure.
We generally use it to perform gradient descent (nth dimensional hill climbing).
Consider the example of recognizing shapes:

$![](https : //paper - attachments.dropbox.com/s_E29353D8DE6A7F32069419A77A11E9F7A8BD49E718AC081FD5F77701FB86FF68_1590911957$
$drawing + 8.jpg)$

Though pixel is functional in general, we allow it to be probabilistic to permit noise.
It will thus be inferred to be very close to 0/1.
A lot of heights will be zeroed as well depending on the column.
We can thus see that we have a lot of **background knowledge** The simplest case of this is $A \iff B$, for which we could simply substitute and reduce cases.