Probabilistic Graphical Models

Moritz Gerhardt

moricetamol

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Contents

I - Introduction

Probabilistic Graphical Models (PGMs) are a powerful framework for representing and reasoning about uncertainty in complex systems. They combine probability theory and graph theory to model the relationships between random variables.

Main questions in this field include:
- How to represent knowledge?
- How to answer queries with a model using inference?
- How to create the right model from data using learning?

PGMs are used in a wide array of real-world applications, including:
- Speech recognition
- Image and text analysis
- Medical Diagnosis
- Robotics
- Information extraction from documents
- Social network analysis
- Planning under uncertainty
- Combinatorial problems like scheduling or logistics

II - Basic Probabilty Theory

1 - Events and Spaces

Sample Space Ω

The sample space **Ω** or **W** is the set of all possible outcomes of a random experiment.

For example, when rolling a six-sided die, the sample space is $\Omega = \{1, 2, 3, 4, 5, 6\}$.

Events

An event is a subset of the sample space Ω. It represents a specific outcome or a group of outcomes.

For example, when rolling a die, the event of rolling an even number can be represented as the set $\{2, 4, 6\}$.

Event Space S

The event space **S** is the set of all *possible* events. It includes all possible groupings of outcomes from the sample space Ω, which means it also contains the empty set $\{\}$ and the entire sample space Ω itself.

For example, when tossing a coin, the sample space is $\Omega = \{H, T\}$, and the event space S includes the events $\{\}$, $\{H\}$, $\{T\}$, and $\{H, T\}$.

(a) - Probability Measures

Probability measures defined over (Ω, S) must satisfy the following axioms:
- $\forall \alpha \in S : P(\alpha) \geq 0$: All probabilities are non-negative.

- $P(\Omega) = 1$: The probability of the entire sample space is 1.
- If α and β are disjoint events, then $P(\alpha \cup \beta) = P(\alpha) + P(\beta)$: The probability of the union of disjoint events is the sum of their individual probabilities.

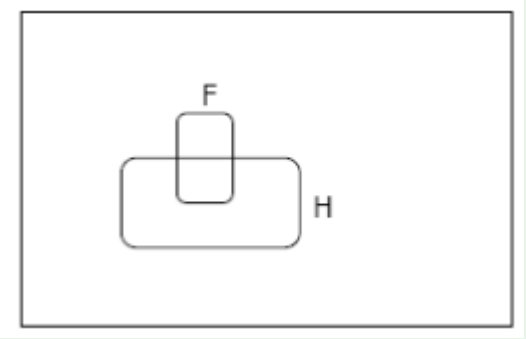
Using these axioms we can define conditional probabilities:

Conditional Probability

The conditional probability of an event **F** given another event **H** is defined as:

$$P(F \mid H) = \frac{P(F \cap H)}{P(H)}$$

provided that P(H) > 0.

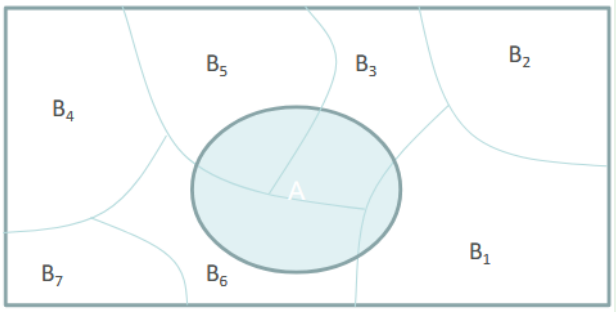


This concept can be expanded to the rule of total probability:

Total Probability

If $\{B_1, B_2, ..., B_n\}$ is a partition of the sample space Ω, then for any event A, the total probability of A can be calculated as:

$$P(A) = \sum P(A \mid B_i) * P(B_i)$$



2 - Random Variables

As defining every single event in the event space S is impractical for large sample spaces, we introduce random variables to simplify the representation of events and their probabilities.

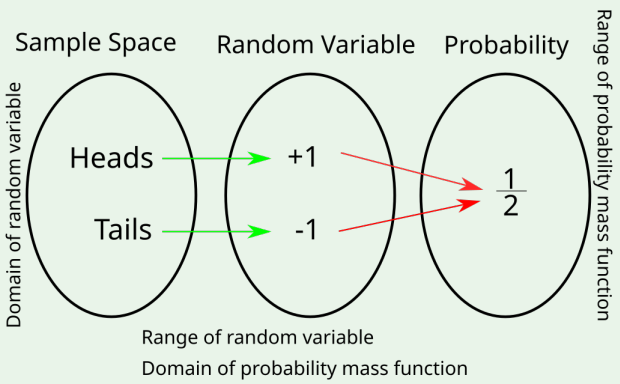
Random Variable

A random variable is a function that maps outcomes from the sample space Ω to real numbers. It assigns a numerical value to each outcome of a random experiment.

For example, take a coin toss where Ω = {H, T}. We can define a random variable X such that:

- X(H) = 1
- X(T) = −1

Since there are only two outcomes, that have an equal number of mappings to real numbers, we can say that $P(X = 1) = \frac{1}{2}$ and $P(X = -1) = \frac{1}{2}$.



3 - Joint Probability Distribution

As we've established, random variables encode attributes, but not all possible combinations of these attributes are equally likely. To capture the relationships between multiple random variables, we use joint probability distributions $P(X = x, Y = y) = P(x, y)$.

Chain Rule

The chain rule allows us to decompose a joint probability distribution into a product of conditional probabilities. For two random variables X and Y, the chain rule states that:

$$P(x, y) = P(x)P(y|x) = P(y)P(x|y)$$

This can be expanded to more variables:

$$P(x, y, z) = P(x)P(y|x)P(z|x, y) = ...$$
$$P(x, y, z, w) = P(x)P(y|x)P(z|x, y)P(w|x, y, z) = ...$$
$$P(x_1, x_2, ..., x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)...P(x_n \mid x_1, x_2, ..., x_{n-1})$$

Likewise, the conditional probabilities utilizing random variables can be expressed as:

$$P(X = x \mid Y = y) = \frac{P(X = x \cap Y = y)}{P(Y = y)}$$

or simpler:

$$P(x \mid y) = \frac{P(x, y)}{P(y)}$$

(a) - Marginalization

Assume we have a joint probability distribution over multiple random variables, but we are only interested in the distribution of a subset of these variables. Marginalization allows us to obtain the marginal probability distribution of a subset by summing over the unwanted variables:

$$P(x) = \sum_y P(x, y)$$

This also can be expanded to more variables:

$$P(x) = \sum_y \sum_z P(x, y, z)$$

etc.

(b) - Bayes Theorem

Bayes Theorem is essentially a recombination of the definition of conditional probability and the rule of total probability. It is a fundamental concept, that describes how to update your belief about somethine after you get new evidence.

Bayes Theorem is comprised of:

- Prior $P(H)$: The initial belief about the probability of a hypothesis before observing any evidence.
- Likelihood $P(E \mid H)$: The probability of observing the evidence given that the hypothesis is true.

- Marginalization $P(E)$: The total probability of observing the evidence under all possible hypotheses.
- Posterior $P(H \mid E)$: The updated belief about the probability of the hypothesis after observing the evidence.

$$\underbrace{P(H \mid E)}_{\text{Posterior}} = \frac{\overbrace{P(H)}^{\text{Prior}} \cdot \overbrace{P(E \mid H)}^{\text{Likelihood}}}{\underbrace{P(E)}_{\text{Marginalization}}}$$

It can also be expanded to multiple variables:

$$P(H_1, ..., H_n \mid E_1, ..., E_m) = \frac{P(H_1, ..., H_n) \cdot P(E_1, ..., E_m \mid H_1, ..., H_n)}{P(E_1, ..., E_m)}$$

4 - Structural Properties

(a) - Independence

Two random variables X and Y are considered independent if the occurrence of one does not affect the probability of the occurrence of the other. Formally, X and Y are independent if:

$$P(X = x, Y = y) = P(X = x) \cdot P(Y = y) \Rightarrow X \perp Y$$

This also means that $P(X \mid Y = y) = P(X)$.

In most models random variables are rarely independent, but we can still leverage the concept of conditional independence to simplify our models:

Conditional Independence

Two random variables X and Y are conditionally independent given a third variable Z if the occurrence of X does not affect the probability of Y when Z is known. Formally, X and Y are conditionally independent given Z if:

$$P(X = x, Y = y \mid Z = z) = P(X = x \mid Z = z) \cdot P(Y = y \mid Z = z) \Rightarrow X \perp Y \mid Z$$

This also means that:

- $P(X \mid Y = y, Z = z) = P(X \mid Z = z)$
- $P(Y \mid X = x, Z = z) = P(Y \mid Z = z)$

This also brings some properties:

- Symmetry: $(X \perp Y \mid Z) \Rightarrow (Y \perp X \mid Z)$
- Decomposition: $(X \perp (Y, W) \mid Z) \Rightarrow (X \perp Y \mid Z)$
- Weak Union: $(X \perp (Y, W) \mid Z) \Rightarrow (X \perp Y \mid (Z, W))$
- Contraction: $(X \perp Y \mid Z) \wedge (X \perp W \mid (Y, Z)) \Rightarrow (X \perp (Y, W) \mid Z)$
- Intersection: $(X \perp Y \mid (W, Z)) \wedge (X \perp W \mid (Y, Z)) \Rightarrow (X \perp (Y, W) \mid Z)$
  - Only holds for strictly positive distributions

III - Bayesian Networks

As probabilities with multiple conditionals get exponentially more difficult to compute the more parameter / variables we add, we can employ the concept of Bayesian Networks using conditional independences to simplify the distribution.

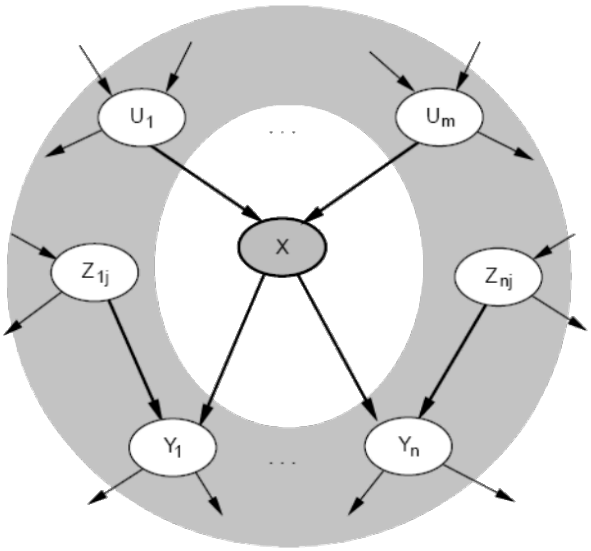
A Bayesian Network (BN) is a probabilistic graphical model that represents a set of random variables and their conditional dependencies as a directed acyclic graph (DAG). In a BN, each node represents a random variable, and the directed edges between nodes represent the conditional dependencies between these variables.

1 - Independence in Bayesian Networks

If we must consider a set of all conditions for a variable, we can reduce the number of parameters we need to store by only considering the parents (direct causes), children (direct effects) and spouses (co-causes) of a variable. This is also known as the Markov Blanket of a variable, as according to the Markov Property.

Local Markov Property

A variable is conditionally independent of all other variables in the network given its Markov Blanket.

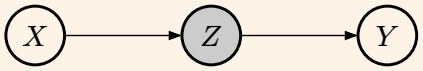


Using graphs as a representation not only yields implementational benefits, but also allows us to visually reason about the relationships between variables and their dependencies. Especially, we can apply the independence properties to identify independencies on a given path. To determine the independencies, we usually use **trails**, which are undirected paths between two nodes in the graph. Trails essentially have two states:

- **Active**: A trail is active if there is no node on the trail that blocks the flow of information between the two nodes.
  - This makes the two nodes dependent.
- **Blocked**: A trail is blocked if there is at least one node on the trail that blocks the flow of information between the two nodes.
  - This makes the two nodes **independent**.

Independence Properties in Bayesian Networks

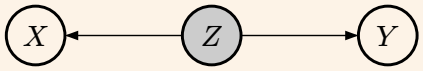
Chain structure:



$\Rightarrow X \not\perp Y$ but: $X \perp Y \mid Z$

- **Blocked Trail if:** Z is **observed**.
- **Active Trail if:** Z is **not** observed.

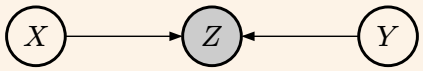
Fork structure:



$\Rightarrow X \not\perp Y$ but: $X \perp Y \mid Z$

- **Blocked Trail if:** Z is **observed**.
- **Active Trail if:** Z is **not** observed.

Collider structure:



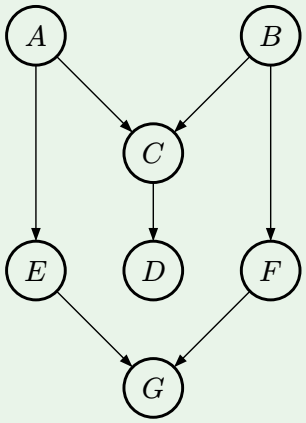
$\Rightarrow X \perp Y$ but: $X \not\perp Y \mid Z$

- **Blocked Trail if:** Z is **not observed**.
- **Active Trail if:** Z (or its descendants) is **observed**.

Important: Just because **a structure** is blocked, does not immediately mean that the two nodes are independent. There might be other active trails between the two nodes.

d-separation

Two nodes X and Y are d-separated by a set of nodes Z if all trails between X and Y are blocked by Z. If X and Y are d-separated by Z, then they are conditionally independent given Z. (Note that Z here can also be a set, including the empty set {}.)



In this example, A and B are d-separated given {}, as **all** trails between them are blocked and therefore independent (G not observed and C or D not observed).

If we observe C,D or G, the trail between A and B becomes active, making them dependent.

d-separation is useful as it only captures true independencies in the graph structure, making it applicable for any probability distribution that the graph represents.

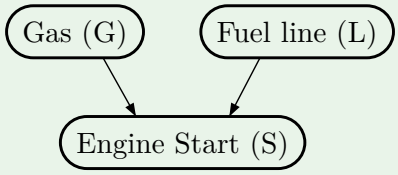
d-separation Algorithm: Bayes-Ball

TODO FIX

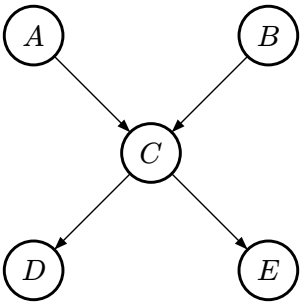
1. Start at source node
2. Get successors:
   - If current node is **unobserved**:
     - Any child node
     - Any parent node if coming from a child
   - If current node is **observed**:
     - Any parent node if coming from a child
3. Utilize a search strategy (e.g., BFS or DFS) to traverse the graph according to the successors rules
4. If you reach the target node, then the source and target are **dependent** given the observed nodes

Context-Specific Independence (CSI)

While d-separation captures independencies that hold for all values of the variables (e.g. $X \perp Y$ always holds, regardless of their values), sometimes independencies only hold for specific values of the conditioning variables. This is known as **Context-Specific Independence (CSI)**. Two variables X and Y might be independent given a specific value of a third variable $Z = z_1$, but dependent given another value $Z = z_2$. For example:



Usually $S \not\perp L$ as the state of the engine start depends on if the fuel line is working. Same with gas: $S \not\perp G$. However, if we know that there is no gas ($G =$ no Gas), then the state of the fuel line does not matter anymore, as the engine won't start regardless. Therefore, in this context we have $S \perp L \mid (G = \text{no Gas})$.



For a Bayesian Network such as this the Joint Distribution normally would be:

$$P(A, B, C, D, E) = P(A)P(B \mid A)P(C \mid A, B)P(D \mid A, B, C)P(E \mid A, B, C, D)$$

However, by leveraging conditional independences we can reduce this to:

$$P(A, B, C, D, E) = P(A)P(B)P(C \mid A, B)P(D \mid C)P(E \mid C)$$

which is significantly easier to compute and store.

So the general form for a Bayesian Network's Joint Distribution is:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid \text{Parents}(X_i))$$

2 - Naïve Bayes

Naïve Bayes is a simple yet powerful algorithm for classification. Its main goal is to look at a piece of data or "instance", examine its features or "attributes" and assign it to a specific category or "class". One simple example is spam filtering, whereby:

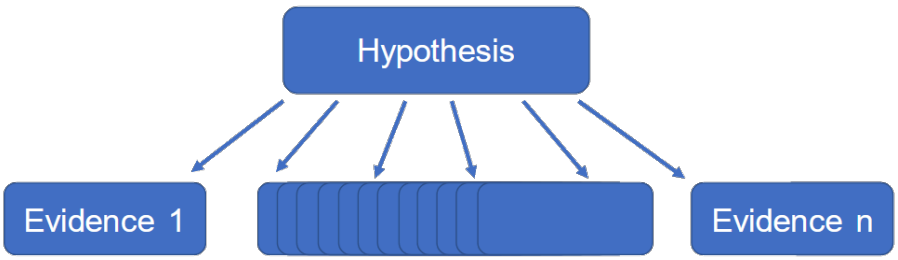
- **Instance**: An email
- **Attributes**: Words in the email
- **Class**: Spam or Not Spam

The model is called "naïve" because it makes a strong assumption that **all attributes are conditionally independent given the class label**. This simplifies the computation so much that makes it feasible to use even with large datasets.

In the example this means that specific words like "estate", "tax", and "income" are considered independent features when determining if an email is spam, though this might not be true in reality.

Formally, the Naïve Bayes classifier is a simple Bayesian Network where one hypothesis H is the parent to all evidence nodes $E_1, E_2, ..., E_n$, with the goal to find the most likely class H_{MAP} (Maximum A Posteriori) given the evidence:

$$H_{MAP} = \arg\max_{h_j \in H} \underbrace{P(h_j \mid E_1, E_2, ..., E_n)}_{\text{Posterior}} = \arg\max_{h_j \in H} \underbrace{P(h_j)}_{\text{Prior}} \cdot \prod_{i=1}^{n} \underbrace{P(E_i \mid h_j)}_{\text{Likelihood}}$$



In naïve bayes, the Hypothesis H is often also referred to as the **class variable C**, while the evidence variables $E_1, E_2, ..., E_n$ are called **attribute variables** or **features**.

There are many more use cases, though Naïve Bayes is mostly used for text classification tasks like spam detection, sentiment analysis, and document categorization.

Despite its "naïve" assumption, Naïve Bayes often performs surprisingly well in practice, especially when the independence assumption holds reasonably well. It is computationally efficient, easy to implement, and requires relatively small amounts of training data to estimate the necessary parameters.

Learning the model

To use a model we need to find the parameters (probabilities) from data. These include:

1. **Class prior probabilities**: $P(C = c_j)$ The overall probability of each class in the training data.
2. **Attribute likelihoods**: $P(E_i \mid C = c_j)$ The probability of each attribute given each class.

The simplest approach is to use Maximum Likelihood Estimation (MLE) to estimate these probabilities from the training data by counting occurrences:

$$\hat{P}(c_j) = \frac{N(c_j)}{N}$$

$$\hat{P}(E_i \mid c_j) = \frac{N(E_i, c_j)}{N(c_j)}$$

The problem here comes from zero probabilities, when an attribute never occurs with a specific class in the training data. This would break the classifier as the entire posterior probability would become zero due to multiplication with zero probabilities. To solve this, we can use Laplace Smoothing ("Add-1" smoothing):

$$\hat{P}(H_i \mid c_j) = \frac{N(H_i, c_j) + 1}{N(c_j) + k}$$

where k is the number of possible values that H_i can take.

For example for text classification, k would be the size of the vocabulary.

3 - Independence Maps

As a probability distribution (P) can be represented as a bayesian network (G), there can be multiple graphs that represent the same distribution. To formalize this, we can use Independence Maps (I-Maps).

Independence Map (I-Map)

A bayesian network G is an I-Map of a probability distribution P if all the conditional independencies represented in G also hold in P. In other words, the graph G is an I-Map of P if all the independency assumptions in G are valid in P.

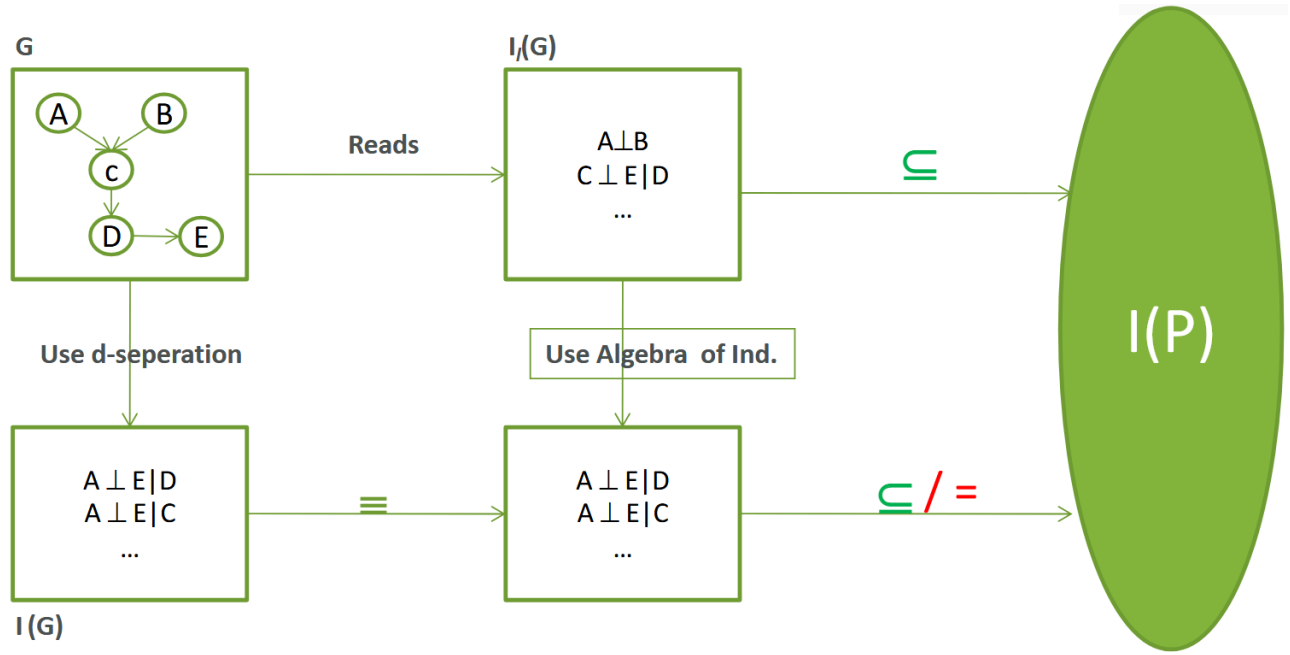
Formally:

$$I_I(G) \subseteq I(P)$$

where $I_I(G)$ are the independencies in G and $I(P)$ are the independencies in P.

Therefore, a distribution P can have multiple I-Maps, as different graphs can represent the same set of conditional independencies.

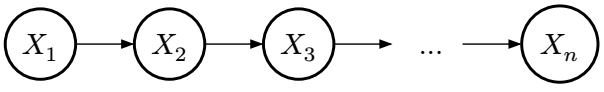
What this means in practice is, that when constructing a bayesian network, adding extra edges (dependencies) will always result in a valid I-Map, as it does not remove any independencies. However, removing edges (dependencies) can lead to invalid I-Maps if the removed edge represented a true dependency in the distribution P.



4 - Inference in Bayesian Networks

Inference in BNs is very complex, even approximate inference is NP-hard. In practice, we therefore usually exploit the BN structure.

(a) - Inference in Simple Chains



In a simple chain like this, how do we compute $P(X_n)$?

The naïve approach would be to sum over all other variables:

$$P(X_n) = \sum_{X_{n-1}} ... \sum_{X_2} \sum_{X_1} P(X_1, X_2, ..., X_n) = \sum_{X_{n-1}} ... \sum_{X_2} \sum_{X_1} P(X_1)P(X_2 \mid X_1)...P(X_n \mid X_{n-1})$$

This, of course, is incredible inefficient, as the number of summations grows exponentially with n: $O(k^n)$ with k being the number of possible values for each variable.

One easier way is to iteratively compute $P(X_1)$, $P(X_2)$, ..., $P(X_n)$:

$$P(X_{i+1}) = \sum_{x_i} P(X_i) \cdot P(X_{i+1} \mid X_i)$$

This is a lot more efficient as we only have to do n summations, each with k terms, leading to a total complexity of $O(n \cdot k^2)$.

(b) - Variable Elimination

Variable elimination is an algorithm for performing inference in Bayesian Networks by systematically eliminating variables from the network to compute the desired probabilities more efficiently.

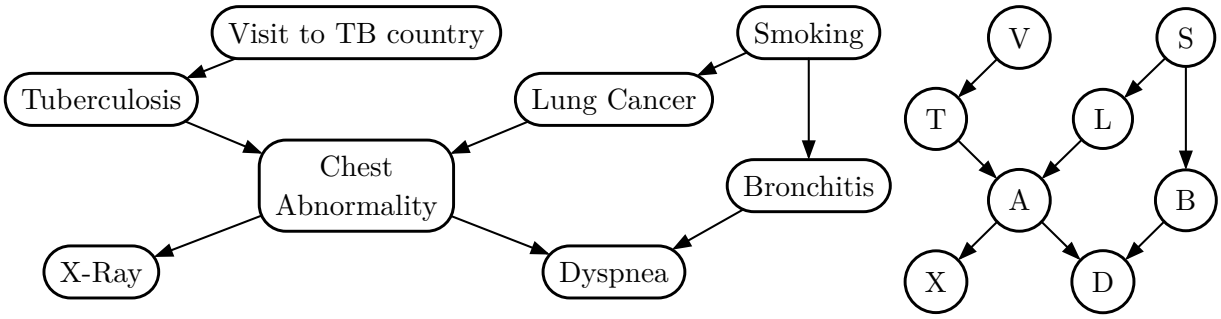
The main idea behind variable elimination is to break down the joint probability distribution into smaller factors, which can be manipulated and combined to compute the desired probabilities without having to consider the entire joint distribution at once.

Generally, we want to write a query in the form:

$$P(X_n, e) = \sum_{x_k} ... \sum_{x_3} \sum_{x_2} \prod_i P(X_i \mid \text{Parents}(X_i))$$

And then iteratively:
1. Move all irrelevant factors (not involving query or evidence variables) out of the innermost sum
2. Perform innermost sum, creating a new factor (or potential)
3. Insert the new term back into the distribution

For example:



The probability distribution for this network is:

$$P(v, s, t, l, b, a, x, d) = P(v)P(s)P(t \mid v)P(l \mid s)P(b \mid s)P(a \mid t, l)P(x \mid a)P(d \mid b, d)$$

Assume we want to compute $P(d)$. In that case we need to eliminate: v, s, x, t, l, a, b.

Starting with eliminating v:

1. Gather all factors involving v: $P(v), P(t|v)$
2. Sum over v: $f_v(t) = \sum_v P(v)P(t|v)$
   - This is just $P(t)$

3. New factors: $f_v(t)(= P(t)), P(s), P(l|s), P(b|s), P(a|t,l), P(x|a), P(d|a,b)$

Repeat this for all other variables:

$$
\begin{aligned}
& & P(v,s,x,t,l,a,b,d) &= \underline{P(v)}P(s)\underline{P(t|v)}P(l|s)P(b|s)P(a|l,t)P(x|a)P(d|a,b) \\
v \Rightarrow & & P(s,x,t,l,a,b,d) &= \boldsymbol{f_v(t)}\underline{P(s)P(l|s)P(b|s)}P(a|l,t)P(x|a)P(d|a,b) \\
s \Rightarrow & & P(x,t,l,a,b,d) &= f_v(t)\boldsymbol{f_s(b,l)}P(a|t,l)\underline{P(x|a)}P(d|a,b) \\
x \Rightarrow & & P(t,l,a,b,d) &= \underline{f_v(t)}f_s(b,l)\boldsymbol{f_x(a)}\underline{P(a|t,l)}P(d|a,b) \\
t \Rightarrow & & P(l,a,b,d) &= \underline{f_s(b,l)}f_x(a)\underline{\boldsymbol{f_t(a,l)}}P(d|a,b) \\
l \Rightarrow & & P(a,b,d) &= \underline{f_x(a)\boldsymbol{f_l(a,b)}P(d|a,b)} \\
a \Rightarrow & & P(b,d) &= \underline{\boldsymbol{f_a(b,d)}} \\
b \Rightarrow & & P(d) &= \boldsymbol{f_b(d)}
\end{aligned}
$$

(Note: The underlined terms are the ones that get moved out of the summation in the next step, while the **colored-in terms** are the new factors created after summation.)

If a factor like $P(x \mid a)$ stands alone (no other factors involving the same variable), we can directly sum it out, as it will always sum to 1.

Variable Elimination with Evidence

When evidence is present (e.g. $V = \texttt{true}$), we do not need to eliminate the evidence variables. Instead, we can simply replace the factors involving the evidence variables with their observed values:

$$
\begin{aligned}
f_{P(V)} &= P(V = t) \\
f_{P(T \mid V)}(T) &= P(T \mid V = t)
\end{aligned}
$$

The resulting terms that are constant can then be moved out of the summations during variable elimination, simplifying the computation further. The rest of the process remains the same as before.

In general, the efficiency of variable elimination depends a lot on the order in which variables are eliminated. Choosing an optimal elimination order can significantly reduce the computational complexity of the inference process. However, finding the optimal order is itself a challenging problem and often requires heuristics or approximations.

In graphs that are "tree-like", one common practice is to start eliminating from the leaves of the tree towards the root, as this often leads to smaller intermediate factors and more efficient computation.

(c) - Potentials

Potential

A **potential** f_A or more often ϕ_A over a set of variables A is a function that maps each configuration into a *non-negative real number*. Therefore it is not necessarily a probability, a conditional probability, or anything else. In essence it's just a table of numbers associated with each configuration of the variables in A.

Conditional Probability Table $P(F \mid D, E)$:

D	E	P(F)
T	T	0.8
T	F	0.5
F	T	0.2
F	F	0.7

Potential $\phi_{D,E,F}$:

d	e	f	0.8
d	e	$\neg f$	0.2
d	$\neg e$	f	0.5
d	$\neg e$	$\neg f$	0.5
$\neg d$	e	f	0.2
$\neg d$	e	$\neg f$	0.8
$\neg d$	$\neg e$	f	0.7
$\neg d$	$\neg e$	$\neg f$	0.3

Potentials can be combined using **factor multiplication** and **factor marginalization** (summing out variables), similar to how we handled probabilities in variable elimination.

(d) - Abductive Inference in BNs

So far we've only discussed the goal to obtain posterior probabilities given evidence. Abductive inference aims to find the configuration of a set of variables (hypotheses) which best explains the observed evidence.

We usually differentiate between two types of abductive inference:

Maximum A Posteriori (MAP)

The most probable configuration of a subset of variables given the evidence:

$$MAP = \max_m P(M = m \mid E = e) = \max_m \sum_r P(M = m, R = r \mid E = e)$$

with $M \subseteq H$ = the set of hypothesis variables and

$R = H \setminus M$ = the set of remaining hypothesis variables

Most Probable Explanation (MPE)

The most probable configuration of all variables given the evidence:

$$MPE = \max_h P(H = h \mid E = e)$$

with H = the set of all hypothesis variables

For example:

Assume we have a bayesian network representing a medical diagnosis system, where:

- Hypotheses: Flu (F), Allergy (A), Sinus infection (S) and Nose running (N)

given:

- Evidence: Headache (H) = `true`

The **MPE** would then be defined as

$$\max_{f,a,s,n} P(F = f, A = a, S = s, N = n \mid H = t)$$

(the best overall explanation for the headache considering all variables),

whereas the **MAP** could be defined as

$$\max_{a} P(A = a \mid H = t)$$

(The best single value for allergy considering all other possibilities.)

MPE and MAP are not consistent. Just because a value is part of the MPE, does not imply that it is part of the MAP. "The most probable complete story (MPE) might not contain the most probable individual part (MAP)"

Finding MPE

To find the MPE, we can use a modified version of variable elimination:

- Instead of summing over the hypothesis variables, we take the maximum.
- The rest of the process remains the same as in standard variable elimination.

So the individual steps for each variable we want to eliminate are:

1. Gather all factors involving the variable.
2. Take the maximum of the product of these factors over the variable.
3. Insert the new factor back into the distribution.

This might seem a bit weird, as this would just yield the probability of the MPE, but not the actual configuration. What we do to get the configuration is to keep track of the values that yielded the maximum ($\underline{\arg}\max$) for each step. Once we have attained the MPE probability, we can backtrack through these stored values to reconstruct the most probable configuration of the hypothesis variables.

(e) - Complexity of Conditional Probability Queries

As mentioned before, inference in Bayesian Networks is generally NP-hard. This means that there is no known polynomial-time algorithm that can solve all instances of the problem efficiently. The complexity arises from the combinatorial nature of the problem, as the number of possible configurations of the variables grows exponentially with the number of variables in the network.

This, does however, not mean, that we cannot solve inference for any given network. There simply is no general algorithm that can solve *all*.

IV - Markov Random Field (MRF)

A Markov Random Field (MRF), also known as a Markov Network, is an undirected graphical model that represents the joint distribution of a set of random variables. In an MRF, nodes represent random variables, and edges represent dependencies between these variables. Unlike Bayesian Networks, which use directed edges to represent causal relationships, MRFs use **undirected** edges to capture mutual dependencies.

So in essence, the difference between BNs and MRFs is, that BNs represent conditional probability distributions using directed edges, while MRFs are parameterized by **potentials** over cliques (fully connected subgraphs) in the graph using undirected edges.

Cliques

A **clique** is a subset of nodes where every node has a direct edge to every other node in the subset. A **maximal clique** is a clique that cannot be extended by including an adjacent node, meaning it is not a subset of a larger clique.

Joint Distribution in MRFs

The probability of a configuration of random variables $x = \{x_1, x_2, ..., x_n\}$ in an MRF is given by:

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k\left(x_{\{k\}}\right)$$

where k = the cliques in the graph

Z = the partition function: sum of the product of potentials over all cliques

$x_{\{k\}}$ = the variables in clique k

The **partition function Z** is a normalizing constant that ensures that the probabilities sum to 1 over all possible configurations of the variables. It is computed by summing the product of the potentials over all possible configurations of the variables in the MRF. This can be computationally expensive, especially for large networks with many variables and complex dependencies ($O(k^n)$, with k being the number of states per variable).

Markov Assumptions in MRFs

MRFs rely on the Markov assumptions to simplify the representation of dependencies between variables. The key Markov assumptions in MRFs are:

1. **Pairwise Markov Property**: Two non-adjacent nodes are conditionally independent given all other nodes in the graph.
2. **Local Markov Property**: A node is conditionally independent of all other nodes in the graph given its neighbors.
3. **Global Markov Property**: If a set of nodes S seperates a set A from a set B (All paths from A to B pass through S), then A and B are conditionally independent given S ($A \perp B \mid S$).

(Note: Global ⇒ Local ⇒ Pairwise)

Usually the Global Markov Property is equivalent to the other two, but not always, for example when there are deterministic relationships between variables.

If $\forall x : P(x) > 0$ (for all configurations), then the three properties are equivalent (Hammersley-Clifford Theorem).

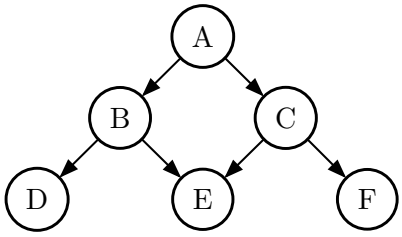
These properties allow us to factor the joint distribution into smaller, more manageable components, making it easier to work with and perform inference on the MRF.

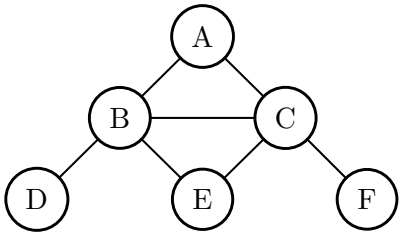
1 - Domain Graph / Moral Graph

Formal Definition

Let $F = \{f_1, ..., f_n\}$ be a set of potentials over a set of variables $U = \{A_1, ..., A_m\}$. Each potential f_i has a domain D_i, over the variables it depends on. Hereby:

- The **domain graph** is an **undirected** graph $G = (V, E)$ where:
  - $V = U$: The nodes are the set of variables
  - $E = \{(A_i, A_j) \mid \exists f_k \in F : A_i \in D_k \wedge A_j \in D_k\}$ (an edge between two variables if they appear together in the domain of the same potential)





CPDs:

$$P(A)$$
$$P(B \mid A)$$
$$P(C \mid A)$$
$$P(D \mid B)$$
$$P(E \mid B, C)$$
$$P(F \mid C)$$

Domains:

$$D_1(\phi_A) = \{A\}$$
$$D_2(\phi_B) = \{B, A\}$$
$$D_3(\phi_C) = \{C, A\}$$
$$D_4(\phi_D) = \{D, B\}$$
$$D_5(\phi_E) = \{E, B, C\}$$
$$D_6(\phi_F) = \{F, C\}$$

The domain graph (also moral graph) is a graph where edges between nodes represent the occurrence of variables together in the same potential. It is undirected, as the relationships

represented by potentials do not have a direction like in Bayesian Networks. As such **it is an MRF** representation of the same distribution as the original Bayesian Network.

As we lose independence information when converting from a BN to a domain graph, we cannot go back from a domain graph to a BN uniquely.

We can do variable elimination on the domain graph similarly to how we did it on the Bayesian Network, by eliminating nodes and connecting their neighbors. This process helps us understand how the elimination of variables affects the dependencies between the remaining variables in the network.

2 - Variable Elimination in MRFs

The steps for variable elimination in MRFs are:
1. Select a variable to eliminate.
2. Identify all neighbors of the variable in the graph.
3. Connect all neighbors to each other (if not already connected) to form a clique. (Fill-ins)
4. Remove the variable from the graph.

This process is repeated until all variables except the query and evidence variables are eliminated. The resulting graph can then be used to compute the desired probabilities more efficiently.

This example shows the elimination of variable C. Its neighbors B, A, E, and F are all connected to each other to form a clique before removing C from the graph.

This is very dependant on the elimination order, as eliminating a variable with many neighbors can create large cliques, increasing the complexity of subsequent eliminations.

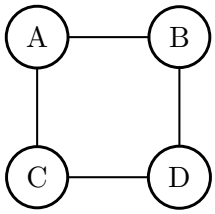
We usually want to avoid creating large cliques (so eliminating variables with many neighbors) to keep the graph as sparse as possible, which helps in reducing the computational complexity of inference. So eliminating "leaf nodes" first (nodes with few connections) is often a good strategy.

If we have a tree-structured graph, we can always eliminate leaf nodes first without creating any new edges, keeping the graph a tree throughout the process, making inference linear.

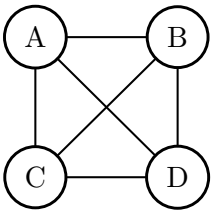
3 - Triangulated Graphs

A graph is **triangulated** if it has a perfect elimination order (no fill-ins).

More formally, a graph is triangulated if every cycle of **four or more nodes** has a chord (an edge that is not part of the cycle but connects two nodes of the cycle). This property ensures that when we perform variable elimination, we do not need to add any fill-in edges, as the existing edges already connect all necessary neighbors.

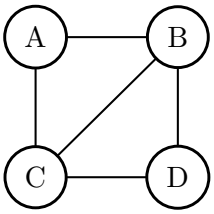


Not Triangulated



Triangulated

(Order does not matter)



Triangulated

(Order: Start with A or D)

Simplicial Nodes

A **simplicial node** is a node whose neighbors form a clique. In other words, all the neighbors of a simplicial node are directly connected to each other.

Eliminating simplicial nodes does not require any fill-in edges, as their neighbors are already fully connected. Therefore, if we can find a perfect elimination order that consists entirely of simplicial nodes, the graph is triangulated:

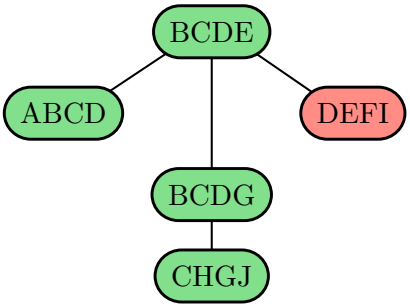
Let G be a triangulated graph and X a simplicial node in G.

Then the graph G' obtained by removing X from G is also triangulated.

4 - Join Trees

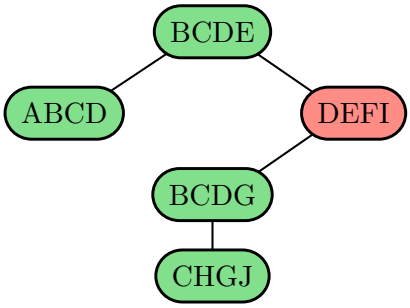
A graph G consisting of the set of cliques from an undirected graph is a **join tree** if it is a tree and satisfies the **running intersection property**:

G is a **join tree** if for any pair of cliques V, W in G, the intersection V ∩ W is contained in every clique on the unique path between V and W in G.



Join Tree

(All nodes on path contain C)



Not a Join Tree

(A Node on path does not contain C)

It holds:

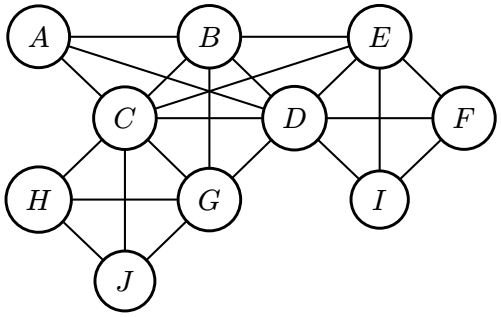
Join Tree ⇔ Triangulated Graph

If G is a triangulated graph, then the graph of its maximal cliques is a join tree.

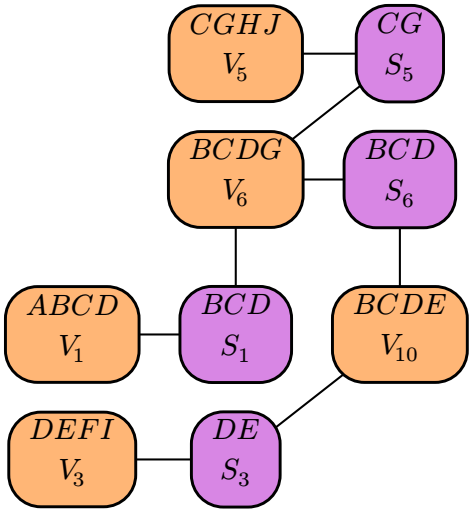
Conversely, if the graph of maximal cliques of G is a join tree, then G is triangulated.

Going from Triangulated Graph to Join Tree

1. Identify cliques
   (a) Find a simplicial node x in the triangulated graph
   (b) Family of x is a maximal clique C
   (c) Elimate all nodes from C that only have neighbors in C
   (d) Give clique of x number i = number of eliminated nodes so far and denote clique as V_i
   (e) Denote the set of remaining nodes as S_i (**separators**)
   (f) Repeat until there are no nodes left to eliminate
2. Connect as tree
   (a) Each clique V_i is connected to its separator S_i
   (b) Connect each seperator S_i to a clique V_j, with $j > i$, such that $S_i \subseteq V_j$



Triangulated Graph



Triangulated Graph

The potential representation of a join tree is the product of the clique potentials, divided by the product of the seperator potentials

$$P(X) = \frac{\prod_C \phi_C(X)}{\prod_S \phi_S(X)}$$

5 - Junction Trees

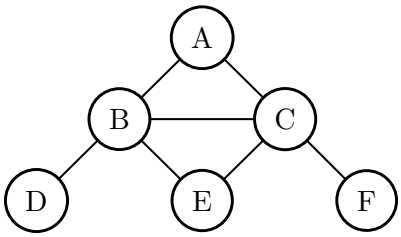
A **Junction Tree** is a specialized type of join tree used in probabilistic graphical models, particularly for performing efficient inference in Bayesian Networks and Markov Random Fields.

It is constructed from the cliques of a triangulated graph and satisfies the running intersection property, similar to join trees. In addition to the basic structure of a join tree, junction trees also contain information about the **potentials associated with each clique and separator**.

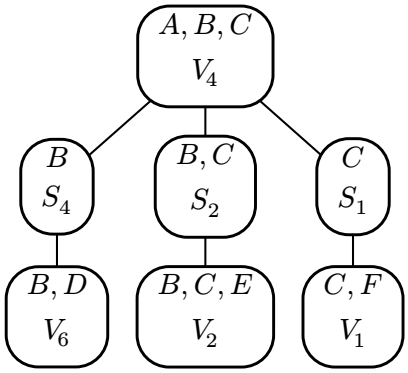
Formal Definition

Let F be a set of potentials with a triangulated domain graph G. A **junction tree** for F is a join tree for G with:

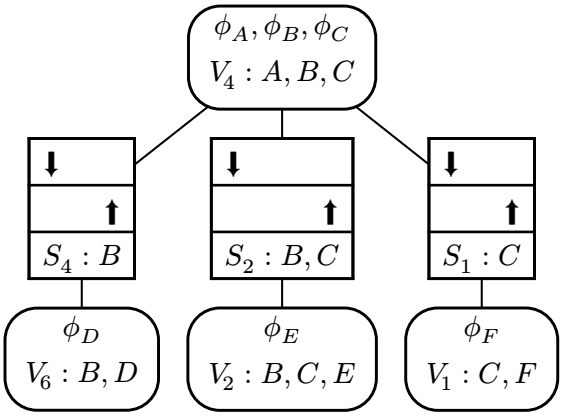
- Each potential ϕ in F is associated to a clique containing the domain $D(\phi)$
- Each link has separator attached containing two "mailboxes", one for each direction of message passing.



MRF



Join Tree



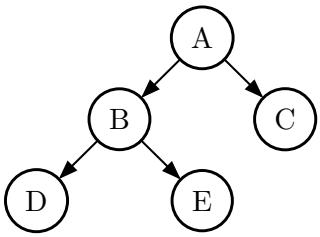
Junction Tree

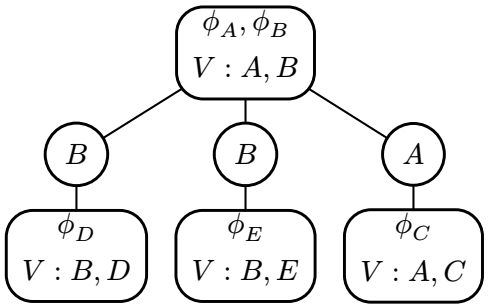
(a) - Optimal Junction Trees

We can always find a **trivial junction tree** with one node containing all variables of the original graph. This is undesirable, as it does not help in reducing the complexity of inference.

An **optimal junction tree** is a junction tree that that minimize the size of the cliques. Again, finding the optimal junction tree is NP-hard, but heuristics exist to find good approximations.

A special case of optimal junction trees are those derived from **tree-structured BNs**:

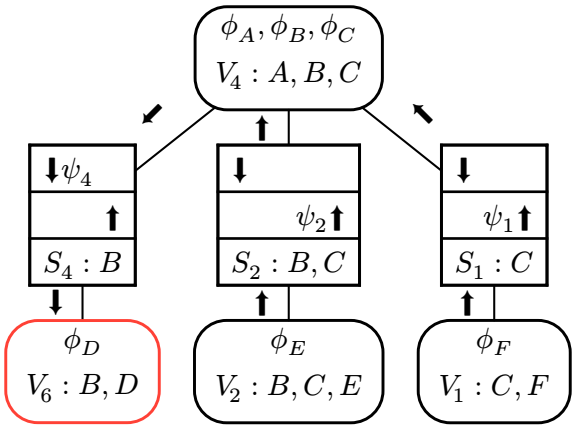




(b) - Propagation on Junction Trees

- Each node V can send exactly one message to a neighbor W, **only** after it has received messages from all of its other neighbors.
- Choose one clique as the root of the tree.

- Collect message to this node and then distribute messages from this node to all other nodes.
- After collection and distribution, each clique contains the marginal distribution over its variables.



Here we want to find $P(D)$, so we need to find the clique containing D, which is V_6. We choose this as the root of the tree.

We then send messages from leaves to the root. V_4 assembles messages from its other neighbors before sending to V_6.

Message Calculation

The message ψ sent from clique V to clique W over separator S is calculated as:

- Marginalize V's potential to get new potential $\phi_S^* = \psi_S$ for S:

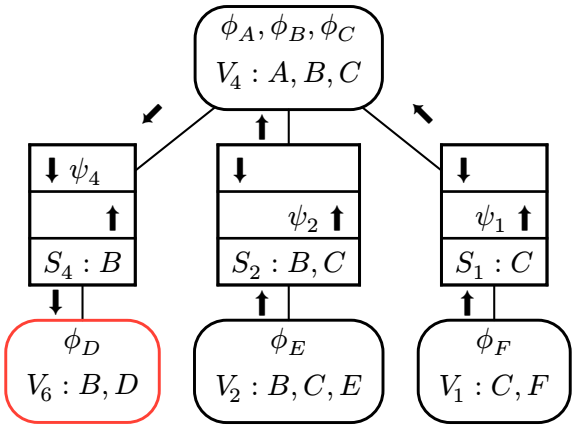
$$\phi_S^* = \sum_{V \setminus S} \phi_V$$

- Update the potential for W:

$$\phi_W^* = \phi_W \cdot \frac{\phi_S^*}{\phi_S}$$

- Update the separator potential (Initially all separator potentials are 1):

$$\phi_S = \phi_S^*$$

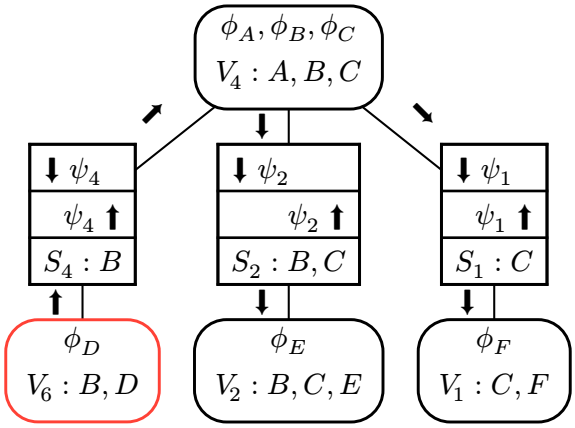


In this example:

$$
\begin{aligned}
\phi_1^* &= \sum_{V_1 \setminus S_1} \phi_F = \sum_F P(F \mid C) = 1 \\
\phi_2^* &= \sum_{V_2 \setminus S_2} \phi_E = \sum_E P(E \mid B, C) = 1 \\
\phi_4^* &= \sum_{V_4 \setminus S_4} (\phi_A \cdot \phi_B \cdot \phi_C \cdot \phi_1^* \cdot \phi_2^*) = \sum_A \phi_A \cdot \phi_B \cdot \sum_C \phi_C \cdot \phi_1^* \cdot \phi_2^* \\
&= \sum_A P(A) P(B \mid A) \cdot \sum_C P(C \mid A) \cdot 1 \cdot 1 \\
&= P(B)
\end{aligned}
$$

$$
P(D) = \sum_B \phi_D \cdot \phi_4^* = \sum_B P(D \mid B) P(B) = \sum_B P(B, D) = P(D)
$$

To get the other marginals, we can just send messages from the root to the leaves, using the same message calculation as before. After this, each clique will contain the marginal distribution over its variables.



6 - Handling Non-Triangulated Graphs

Real-world bayesian networks often do not lead to triangulated domain graphs. In such cases, we need to triangulate the graph before constructing a junction tree.

We can do that by:

- Choose an elimination order for the variables.
- During elimination, add fill-in edges to connect all neighbors of the eliminated variable, ensuring that the resulting graph is triangulated.
- If we then use the original graph with the added fill-in edges to construct the junction tree, we can perform inference as usual.

V - Learning Bayesian Networks from Data

Learning is the process of improving a model over time based on observed data.

For Bayesian Networks, we essentially want to solve this formula:

$$\text{Data} + \text{Prior Knowledge} \xrightarrow{\text{Learning Algorithm}} \text{Model Parameters}$$

We want to learn from data because:

- **Knowledge Acquisition Bottleneck:** It is expensive, difficult and time-consuming to get experts to manually specify all the probabilities in a Bayesian Network.
- **Data Availability:** Data is cheap and abundant in many domains, making it feasible to learn models directly from data.

1 - Data Representation

In general, we assume a dataset D to be consisting of **attributes / variables** $A_1, A_2, ..., A_n$ and **data classes** $X_1, X_2, ..., X_m$, whereby each each pair of attribute and class has a specific value.

Unfortunately, real-world data is often incomplete, meaning some of the states for some random variables might be missing. Other variables might be unobserved (**latent or hidden**) altogether.

	A_1	A_2	...	A_n
X_1	true	true	...	false
X_2	false	true	...	true
⋮	⋮	⋮	...	⋮
X_m	false	true	...	true

Ideal, Complete Dataset

	A_1	A_2	...	A_n
X_1	?	?	...	false
X_2	false	?	...	true
⋮	⋮	⋮	...	⋮
X_m	false	?	...	?

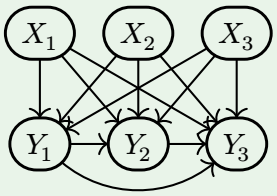
Real-World, Incomplete Dataset

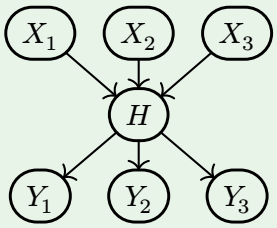
(a) - Hidden / Latent Variables

Hidden or latent variables are not always unwanted. When modelling complex systems, they oftentimes are introduced conciously to simplify a model or to capture underlying patterns in the data.

Simplification / Parameter Reduction

By introducing latent variables, we can often reduce the number of parameters needed to represent the model. This is particularly useful when dealing with high-dimensional data, where the number of possible configurations of observed variables can be extremely large.





Clustering

Latent variables can also help in capturing underlying patterns or structures in the data. For example, in clustering tasks, latent variables can represent cluster memberships, allowing the model to group similar data points together based on shared characteristics.

We've already seen that in **Naïve Bayes**, but it's also used in other models such as Autoclass or kMeans.

Hereby, we have a bunch of observed variables $X_1, X_2, ..., X_n$ but not labels for clusters. We introduce a latent variable C that acts as a parent node for all observed variables, representing the cluster assignment. We assume that the observed variables are conditionally independent given the cluster assignment.