



Reasoning under uncertainty topics- Conditional Independence Representation:

In which we explain how to build network models to reason under uncertainty according to the laws of probability theory.

Introduction

Conditionals of one kind or another are the dominant form of knowledge representation in Artificial Intelligence. However, despite the fact that they play a key role in such different formalisms as production systems and logic programming, there has been little effort made to study the relationships between these different kinds of conditionals. In this paper I present a framework that attempts to unify the two kinds of conditionals and discuss its potential application to the modeling of human reasoning.

Conditionals are also a distinctive feature of logic programming, which has been used widely, also since the 1970s, both to implement practical applications and to formalise knowledge representation in Artificial Intelligence. Conditionals in logic programming have both a logical interpretation as conditionals of the form conclusion if conditions and an interpretation as procedures that reduce the goal of establishing the conclusion to the sub-goals of establishing the conditions. Goal-reduction is a form of backward reasoning. Although forward reasoning has been studied extensively in psychology, backward reasoning seems to have received little attention in studies of human reasoning.

- Production systems, initially developed in AI and later adopted in Cognitive Psychology, have primarily focused on modeling human skills but have had limited impact on human reasoning studies, despite both emphasizing conditionals.
- Production rules in production systems resemble logical implications, although there is debate among scholars about their relationship with formal logic.
- AI tends to use non-monotonic reasoning methods like negation as failure, while Psychology emphasizes classical negation and its role in reasoning.
- An Abductive Logic Programming (ALP) framework to bridge the gap between production rules and logic, integrating goals, beliefs, and abductive reasoning within an agent-based model.
- Details about logic programming semantics and natural language conditionals in logic programming terms are provided in appendices.

Uncertainty:

- Back to planning:
 - Let action $A(t)$ denote leaving for the airport t minutes before the flight
 - For a given value of t , will $A(t)$ get me there on time?



- Problems:

- Partial observability (roads, other drivers' plans, etc.)
- Noisy sensors (traffic reports)
- Uncertainty in action outcomes (flat tire, etc.)
- Immense complexity of modeling and predicting traffic.

How To Deal With Uncertainty

- Implicit methods:

- Ignore uncertainty as much as possible.
- Build procedures that are robust to uncertainty.
- This is the approach in the planning methods studied so far (e.g. monitoring and replanning).

- Explicit methods

- Build a model of the world that describes the uncertainty (about the system's state, dynamics, sensors, model).
- Reason about the effect of actions given the model.

Conditional Probability

- The basic statements in the Bayesian framework talk about conditional probabilities.

- $P(A|B)$ is the belief in event A given that event B is known with certainty

- The product rule gives an alternative formulation:

$$P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$$

- Note: we often write $P(A, B)$ as a shorthand for $P(A \wedge B)$

Conditional Independence

- Two variables X and Y are conditionally independent given Z if:

$$P(x|y, z) = P(x|z), \forall x, y, z$$

- This means that knowing the value of Y does not change the prediction about X if the value of Z is known.

Example

- Consider a patient with three random variables:

B (patient has bronchitis), F (patient has fever), C (patient has a cough)

- The full joint distribution has $2^3 - 1 = 7$ independent entries

- If someone has bronchitis, we can assume that, the probability of a cough does not depend on whether they have a fever:

$$P(C|B, F) = P(C|B) \text{-----(1)}$$

I.e., C is conditionally independent of F given B

- The same independence holds if the patient does not have bronchitis:

$$P(C|\neg B, F) = P(C|\neg B) \text{----- (2)}$$



therefore C and F are conditionally independent given B.

The full joint distribution can now be written as:

$$\begin{aligned} P(C, F, B) &= \\ &= P(C, F|B)P(B) \\ &= P(C|B)P(F|B)P(B) \end{aligned}$$

I.e., $2 + 2 + 1 = 5$ independent numbers (equations 1 and 2 remove two numbers)

Much more important savings happen if the system has lots of variables!

1. Conditional independence

Conditional independence can be seen as a generalization of formula-variable independence. Given three sets of propositional variables X , Y and Z , and a propositional formula Σ , we want to express the fact that, given Σ and some knowledge about Z , the truth value of the variables in X may affect the truth value of variables in Y (and vice versa).

1.1 Simple conditional independence

Darwiche and Pearl's conditional independence (often referred as "simple conditional independence" or "conditional independence" in the following) is defined as follows:

Definition (conditional independence). Let Σ be a propositional formula and X , Y , Z be disjoint subsets of PS. X and Y are independent given Z with respect to Σ (denoted by $X \sim_Z \Sigma Y$) if and only if $\forall \omega_X \in \Omega_X, \forall \omega_Y \in \Omega_Y, \forall \omega_Z \in \Omega_Z$, the consistency of both $\omega_X \wedge \omega_Z \wedge \Sigma$ and $\omega_Y \wedge \omega_Z \wedge \Sigma$ implies the consistency of $\omega_X \wedge \omega_Y \wedge \omega_Z \wedge \Sigma$.

Example 1: Let $\Sigma = \{\neg a \vee \neg b \vee c, \neg a \vee b \vee d, a \vee \neg c, \neg a \vee c \vee d, b \vee \neg c \vee d\}$. We have the following:

- $c \sim \emptyset \Sigma d$. Indeed, $(c \wedge d) \wedge \Sigma$, $(c \wedge \neg d) \wedge \Sigma$, $(\neg c \wedge d) \wedge \Sigma$ and $(\neg c \wedge \neg d) \wedge \Sigma$ are all consistent.

- $c \sim \{a\} \Sigma d$. Indeed, $(a \wedge \neg c \wedge \neg d) \wedge \Sigma$ is inconsistent while $(a \wedge \neg c) \wedge \Sigma$ and $(a \wedge \neg d) \wedge \Sigma$ are both consistent. Intuitively speaking, when a is true, learning $\neg c$ tells that d is true.

- $c \sim \{a, b\} \Sigma d$. Indeed, the set of $\{a, b, c\}$ -worlds that are consistent with Σ is

$$S1 = \{a \wedge b \wedge c, a \wedge \neg b \wedge c, a \wedge \neg b \wedge \neg c, \neg a \wedge b \wedge \neg c, \neg a \wedge \neg b \wedge \neg c\};$$

the set of $\{a, b, d\}$ -worlds that are consistent with Σ is

$$S2 = \{a \wedge b \wedge d, a \wedge b \wedge \neg d, a \wedge \neg b \wedge d, \neg a \wedge b \wedge d, \neg a \wedge b \wedge \neg d, \neg a \wedge \neg b \wedge d\};$$



and the set of $\{a, b, c, d\}$ -worlds that are consistent with Σ (in other terms, the models of Σ) is $S3 = \{a \wedge b \wedge c \wedge d, a \wedge b \wedge c \wedge \neg d, a \wedge \neg b \wedge c \wedge d, a \wedge \neg b \wedge \neg c \wedge d, \neg a \wedge b \wedge \neg c \wedge d, \neg a \wedge b \wedge \neg c \wedge \neg d, \neg a \wedge \neg b \wedge \neg c \wedge d, \neg a \wedge \neg b \wedge \neg c \wedge \neg d\}$;

it can be checked that for each $\omega_1 \in S1$ and each $\omega_2 \in S2$ such that ω_1 and ω_2 give the same truth values to a and b , then $\omega_1 \wedge \omega_2 \in S3$.

As explained by Darwiche and Pearl, $X \sim_Z^\Sigma Y$ holds if and only if for any possible full information about Z , adding some information about Y does not tell us anything new about X . Intuitively, if in the context ω_Z , adding ω_X gives some information about Y , then 86 J. Lang et al. / Artificial Intelligence 141 (2002) 79–121 some partial models of Σ over Y , i.e., those in contrast with the new information obtained on Y , should not remain partial models any longer. As a result, X and Y are independent if, for any “consistent” choice (with Σ) of ω_X , ω_Y , and ω_Z , the formula $\omega_X \wedge \omega_Y \wedge \omega_Z \wedge \Sigma$ is consistent.

Clearly enough, conditional independence given Z with respect to Σ satisfies the following properties -

- (1) $X \sim_Z^\Sigma Y$ if and only if $Y \sim_Z^\Sigma X$.
- (2) If $\Sigma \equiv \Sigma'$, then $(X \sim_Z^\Sigma Y \text{ if and only if } X \sim_{\Sigma'}^\Sigma Y)$.
- (3) If $X' \subseteq X$, $Y' \subseteq Y$ and $X \sim_Z^\Sigma Y$, then $X' \sim_Z^\Sigma Y'$.

Proof. (1) and (2) are straightforward. As to (3), assume $X' \subseteq X$, $Y' \subseteq Y$ and $X \sim_Z^\Sigma Y$ and let $\omega_{X'}$, $\omega_{Y'}$ and ω_Z s.t. $\omega_{X'} \wedge \omega_Z$ and $\omega_{Y'} \wedge \omega_Z$ are both consistent. Since $\omega_{X'} \equiv \bigvee_{\omega_X \supseteq \omega_{X'}} \omega_X$, there is an $\omega_X \supseteq \omega_{X'}$ s.t. $\omega_X \wedge \omega_Z$ is consistent, and similarly, there is an $\omega_Y \supseteq \omega_{Y'}$ s.t. $\omega_Y \wedge \omega_Z$ is consistent. Now, because $X \sim_Z^\Sigma Y$, we get that $\omega_X \wedge \omega_Y \wedge \omega_Z \wedge \Sigma$ is consistent, which in turn implies the consistency of $\omega_{X'} \wedge \omega_{Y'} \wedge \omega_Z \wedge \Sigma$. \square

However, conditional independence is stable neither by contraction nor by expansion of Z . For instance, taking the same Σ as in Example 1, we have $c \sim_\Sigma^\emptyset d$ and however $c \not\sim_\Sigma^{\{a\}} d$; we have $c \not\sim_\Sigma^{\{a\}} d$ and however $c \sim_\Sigma^{\{a,b\}} d$. Conditional independence is also not stable by weakening or strengthening Σ in the general case. Thus, while we have $c \sim_\Sigma^\emptyset d$, we also have $c \not\sim_{\Sigma \cup \{c \leftrightarrow d\}}^\emptyset d$; while we have $c \sim_\Sigma^{\{a,b\}} d$, we also have $c \not\sim_{\Sigma \setminus \{\neg a \vee \neg b \vee c\}}^{\{a,b\}} d$.

The two limit cases when Z is respectively empty or equal to $Var(\Sigma) \setminus (X \cup Y)$, are of particular interest, especially when computational complexity is investigated.

Conditional independence can prove a valuable notion to improve many forms of reasoning (including consistency, entailment, diagnosis and abduction). It can also be helpful in the context of reasoning about actions, when dealing about concurrency: indeed, the following definition of compatibility between actions consisting in saying that *two actions a and b are compatible if and*



only if for each initial situation where both actions a and b are separately applicable (i.e., without producing an inconsistency), then a and b are jointly applicable, and can be mapped easily into a conditional independence problem.

Altogether, this explains why conditional independence is an important notion and motivates the investigation of its computational complexity.

1.2 Strong conditional independence

Strong conditional independence Simple conditional independence does not apply to contexts where the new information that can be learned about Z is incomplete, i.e., the truth value of some variables of Z is not available, or, more generally, many partial (and possibly mutually exclusive) Z -worlds are possible. For instance, if Z represents a set of possibly measurable variables, associated to a set of sensors (one for each $z \in Z$), it can be the case that some measurements fail, i.e. The value of z is not always available.

The following notion, *strong conditional independence*, strengthens Darwiche and Pearl's conditional independence by taking into account the case in which the information about Z is *any conjunctive information*, i.e., any term of $\text{PROP}Z$. Namely, X and Y are strongly independent given Z with respect to Σ if and only if, *whichever conjunctive information (i.e., a set of facts)* we may learn about Z , then the addition of information about Y does not enable one to tell anything new about X .

Definition (strong conditional independence): Let Σ be a propositional formula and X, Y, Z be disjoint subsets of PS . X and Y are strongly independent given Z with respect to Σ

(denoted $X \approx_{\Sigma}^Z Y$) if and only if, for every term γZ of $\text{PROP}Z$, $\forall \omega X \in \Omega X, \forall \omega Y \in \Omega Y$, the consistency of both $\omega X \wedge \gamma Z \wedge \Sigma$ and $\omega Y \wedge \gamma Z \wedge \Sigma$ implies the consistency of $\omega X \wedge \omega Y \wedge \gamma Z \wedge \Sigma$.

Strong conditional independence has the same metatheoretic properties as conditional independence, plus the preservation by contraction of Z (which is a trivial consequence of the definition).

Complexity of conditional independence:

The results are synthesized in Table 1



Table 1
Complexity of conditional independence

$X \sim_{\Sigma}^Z Y$	any Z	$Z = \emptyset$ (marginal independence)	$Z = \text{Var}(\Sigma) \setminus (X \cup Y)$ (ceteris paribus)
any X, Y	Π_2^P -complete	Π_2^P -complete	coNP-complete
$X = \{x\}$ or $Y = \{y\}$	Π_2^P -complete	Π_2^P -complete	coNP-complete
$X = \{x\}$ and $Y = \{y\}$	Π_2^P -complete	coBH ₂ -complete	coNP-complete
$X \cup Y = \text{Var}(\Sigma)$ (twofold partition)	coNP-complete	coNP-complete	coNP-complete

Complexity results of **strong independence** are reported in Table 2

Table 2
Complexity of strong conditional independence

$X \approx_{\Sigma}^Z Y$	any Z	$Z = \text{Var}(\Sigma) \setminus (X \cup Y)$
any X, Y	Π_2^P -complete	Π_2^P -complete
$X = \{x\}$ or $Y = \{y\}$	Π_2^P -complete	Π_2^P -complete
$X = \{x\}$ and $Y = \{y\}$	Π_2^P -complete	Π_2^P -complete

References:

- <https://www.doc.ic.ac.uk/~rak/papers/conditionals.pdf>
- [https://doi.org/10.1016/S0004-3702\(02\)00244-8](https://doi.org/10.1016/S0004-3702(02)00244-8)
- https://www.sciencedirect.com/science/article/pii/S0004370202002448?ref=pdf_download&fr=RR-2&rr=8103be4a0fa01b7d
- https://www.researchgate.net/publication/250297424_Reasoning_with_Conditionals_in_Artificial_Intelligence
- <https://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture10.pdf>



1. APPROXIMATE INFERENCE IN BAYESIAN NETWORKS

Bayesian networks, a well-developed representation for uncertain knowledge. Bayesian networks play a role roughly analogous to that of propositional logic for definite knowledge.

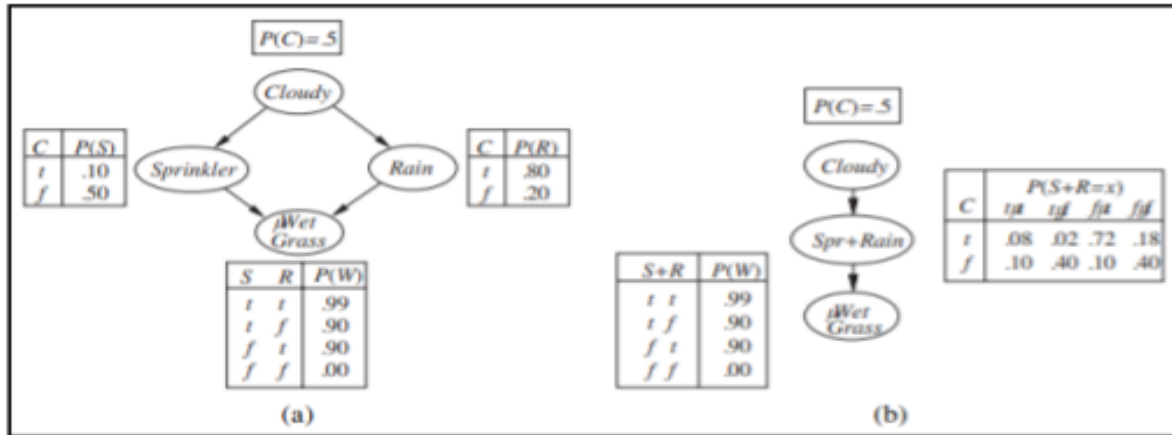
- A Bayesian network is a directed acyclic graph whose nodes correspond to random variables; each node has a conditional distribution for the node, given its parents.
- Bayesian networks provide a concise way to represent **conditional independence** relationships in the domain.
- A Bayesian network specifies a full joint distribution; each joint entry is defined as the product of the corresponding entries in the local conditional distributions. A Bayesian network is often exponentially smaller than an explicitly enumerated joint distribution.
- Stochastic approximation techniques such as likelihood weighting and Markov chain Monte Carlo can give reasonable estimates of the true posterior probabilities in a network and can cope with much larger networks than can exact algorithms.

Given the intractability of exact inference in large, multiply connected networks, it is essential to consider approximate inference methods. This section describes randomized sampling algorithms, also called **Monte Carlo algorithms**. We are interested in sampling applied to the computation of posterior probabilities. We describe two families of algorithms: direct sampling and Markov chain sampling.

1.1 Direct sampling methods.

The primitive element in any sampling algorithm is the generation of samples from a known probability distribution. For example, an unbiased coin can be thought of as a random variable *Coin* with values (*heads*, *tails*) and a prior distribution $P(\text{Coin}) = 0.5, 0.5$. Sampling from this distribution is exactly like flipping the coin: with probability 0.5 it will return *heads*, and with probability 0.5 it will return *tails*. Given a source of random numbers uniformly distributed in the range $[0, 1]$, it is a simple matter to sample any distribution on a single variable, whether discrete or continuous.

The simplest kind of random sampling process for Bayesian networks generates events from a network that has no evidence associated with it. The idea is to sample each variable in turn, in topological order. The probability distribution from which the value is sampled is conditioned on the values already assigned to the variable's parents. This algorithm is shown in Figure 14.13. We can illustrate its operation on the network in Figure(a),



(a) A multiply connected network with conditional probability tables. (b) A clustered equivalent of the multiply connected network.

assuming an ordering $[Cloudy, Sprinkler, Rain, WetGrass]$:

1. Sample from $\mathbf{P}(Cloudy) = \langle 0.5, 0.5 \rangle$, value is *true*.
2. Sample from $\mathbf{P}(Sprinkler | Cloudy = true) = \langle 0.1, 0.9 \rangle$, value is *false*.
3. Sample from $\mathbf{P}(Rain | Cloudy = true) = \langle 0.8, 0.2 \rangle$, value is *true*.
4. Sample from $\mathbf{P}(WetGrass | Sprinkler = false, Rain = true) = \langle 0.9, 0.1 \rangle$, value is *true*.

In this case, PRIOR-SAMPLE returns the event $[true, false, true, true]$.

function PRIOR-SAMPLE(*bn*) **returns** an event sampled from the prior specified by *bn*

inputs: *bn*, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \dots, X_n)$

x \leftarrow an event with *n* elements

foreach variable X_i **in** X_1, \dots, X_n **do**

$x[i] \leftarrow$ a random sample from $\mathbf{P}(X_i | \text{parents}(X_i))$

return **x**

A sampling algorithm that generates events from a Bayesian network. Each variable is sampled according to the conditional distribution given the values already sampled for the variable's parents.

It is easy to see that PRIOR-SAMPLE generates samples from the prior joint distribution specified by the network. First, let $Sps(x_1, \dots, x_n)$ be the probability that a specific event is generated by the PRIOR-SAMPLE algorithm. Just looking at the sampling process, we have



$$S_{PS}(x_1 \dots x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$$

because each sampling step depends only on the parent values. This expression should look familiar, because it is also the probability of the event according to the Bayesian net's representation of the joint distribution.

$$S_{PS}(x_1 \dots x_n) = P(x_1 \dots x_n) .$$

This simple fact makes it easy to answer questions by using samples.

In any sampling algorithm, the answers are computed by counting the actual samples generated. Suppose there are N total samples, and let $N_{ps}(x_1, \dots, x_n)$ be the number of times the specific event x_1, \dots, x_n occurs in the set of samples. We expect this number, as a fraction of the total, to converge in the limit to its expected value according to the sampling probability:

$$\lim_{N \rightarrow \infty} \frac{N_{PS}(x_1, \dots, x_n)}{N} = S_{PS}(x_1, \dots, x_n) = P(x_1, \dots, x_n) .$$

For example, consider the event produced earlier: [true, false, true, true]. The sampling probability for this event is

$$S_{PS}(\text{true}, \text{false}, \text{true}, \text{true}) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324 .$$

Hence, in the limit of large N , we expect 32.4% of the samples to be of this event.

Rejection sampling in Bayesian networks:

Rejection sampling is a general method for producing samples from a hard-to-sample distribution given an easy-to-sample distribution. In its simplest form, it can be used to compute conditional probabilities—that is, to determine $P(X | e)$. The REJECTION-SAMPLING algorithm is shown in Figure 14.14. First, it generates samples from the prior distribution specified by the network. Then, it rejects all those that do not match the evidence. Finally, the estimate $\hat{P}(X = x | e)$ is obtained by counting how often $X = x$ occurs in the remaining samples.

Let $\hat{P}(X | e)$ be the estimated distribution that the algorithm returns. From the definition of the algorithm, we have

$$\hat{P}(X | e) = \alpha N_{PS}(X, e) = \frac{N_{PS}(X, e)}{N_{PS}(e)} .$$

This becomes



Piyush Wairale

GATE in Data Science and AI study material

For GATE DA full test series & study materials. Visit piyushwairale.com

$$\hat{P}(X | e) \approx \frac{P(X, e)}{P(e)} = P(X | e) .$$

That is, rejection sampling produces a consistent estimate of the true probability.

function REJECTION-SAMPLING(X, e, bn, N) **returns** an estimate of $P(X|e)$

inputs: X , the query variable

e , observed values for variables E

bn , a Bayesian network

N , the total number of samples to be generated

local variables: N , a vector of counts for each value of X , initially zero

for $j = 1$ to N **do**

$x \leftarrow \text{PRIOR-SAMPLE}(bn)$

if x is consistent with e **then**

$N[x] \leftarrow N[x] + 1$ where x is the value of X in x

return NORMALIZE(N)

The rejection-sampling algorithm for answering queries given evidence in a Bayesian network.

Likelihood weighting:


Likelihood weighting avoids the inefficiency of rejection sampling by generating only events that are consistent with the evidence e . It is a particular instance of the general statistical technique of **importance sampling**, tailored for inference in Bayesian networks.



```
function LIKELIHOOD-WEIGHTING( $X, \mathbf{e}, bn, N$ ) returns an estimate of  $\mathbf{P}(X|\mathbf{e})$ 
  inputs:  $X$ , the query variable
            $\mathbf{e}$ , observed values for variables  $\mathbf{E}$ 
            $bn$ , a Bayesian network specifying joint distribution  $\mathbf{P}(X_1, \dots, X_n)$ 
            $N$ , the total number of samples to be generated
  local variables:  $\mathbf{W}$ , a vector of weighted counts for each value of  $X$ , initially zero

  for  $j = 1$  to  $N$  do
     $\mathbf{x}, w \leftarrow \text{WEIGHTED-SAMPLE}(bn, \mathbf{e})$ 
     $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$  where  $x$  is the value of  $X$  in  $\mathbf{x}$ 
  return NORMALIZE( $\mathbf{W}$ )
```

```
function WEIGHTED-SAMPLE( $bn, \mathbf{e}$ ) returns an event and a weight
   $w \leftarrow 1$ ;  $\mathbf{x} \leftarrow$  an event with  $n$  elements initialized from  $\mathbf{e}$ 
  foreach variable  $X_i$  in  $X_1, \dots, X_n$  do
    if  $X_i$  is an evidence variable with value  $x_i$  in  $\mathbf{e}$ 
      then  $w \leftarrow w \times P(X_i = x_i \mid \text{parents}(X_i))$ 
    else  $\mathbf{x}[i] \leftarrow$  a random sample from  $\mathbf{P}(X_i \mid \text{parents}(X_i))$ 
  return  $\mathbf{x}, w$ 
```

 The likelihood-weighting algorithm for inference in Bayesian networks. In WEIGHTED-SAMPLE, each nonevidence variable is sampled according to the conditional distribution given the values already sampled for the variable's parents, while a weight is accumulated based on the likelihood for each evidence variable.

1.3 Inference by Markov chain simulation.

Markov chain Monte Carlo (MCMC) algorithms work quite differently from rejection sampling and likelihood weighting. Instead of generating each sample from scratch, MCMC algorithms generate each sample by making a random change to the preceding sample. It is therefore helpful to think of an MCMC algorithm as being in a particular current state specifying a value for every variable and generating a next state by making random changes to the current state.

Here we describe a particular form of MCMC called Gibbs sampling, which is especially well suited for Bayesian networks.

Gibbs sampling in Bayesian networks:



The Gibbs sampling algorithm for Bayesian networks starts with an arbitrary state (with the evidence variables fixed at their observed values) and generates a next state by randomly sampling a value for one of the nonevidence variables X_i . The sampling for X_i is done *conditioned on the current values of the variables in the Markov blanket of X_i* .

Consider the query $P(\text{Rain} \mid \text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true})$ applied to the network in Figure(a). The evidence variables *Sprinkler* and *WetGrass* are fixed to their observed values and the nonevidence variables *Cloudy* and *Rain* are initialized randomly— let us say true and false respectively. Thus, the initial state is $[\text{true}, \text{true}, \text{false}, \text{true}]$. Now the nonevidence variables are sampled repeatedly in an arbitrary order. For example:

- *Cloudy* is sampled, given the current values of its Markov blanket variables: in this case, we sample from $P(\text{Cloudy} \mid \text{Sprinkler} = \text{true}, \text{Rain} = \text{false})$. (Shortly, we will show how to calculate this distribution.) Suppose the result is *Cloudy* = false. Then the new current state is $[\text{false}, \text{true}, \text{false}, \text{true}]$.
- *Rain* is sampled, given the current values of its Markov blanket variables: in this case, we sample from $P(\text{Rain} \mid \text{Cloudy} = \text{false}, \text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true})$. Suppose this yields *Rain* = true. The new current state is $[\text{false}, \text{true}, \text{true}, \text{true}]$.

```
function GIBBS-ASK( $X, e, bn, N$ ) returns an estimate of  $P(X|e)$ 
  local variables:  $N$ , a vector of counts for each value of  $X$ , initially zero
                   $Z$ , the nonevidence variables in  $bn$ 
                   $x$ , the current state of the network, initially copied from  $e$ 

  initialize  $x$  with random values for the variables in  $Z$ 
  for  $j = 1$  to  $N$  do
    for each  $Z_i$  in  $Z$  do
      set the value of  $Z_i$  in  $x$  by sampling from  $P(Z_i|mb(Z_i))$ 
       $N[x] \leftarrow N[x] + 1$  where  $x$  is the value of  $X$  in  $x$ 
  return NORMALIZE( $N$ )
```

The Gibbs sampling algorithm for approximate inference in Bayesian networks; this version cycles through the variables, but choosing variables at random also works.

Let $q(x \rightarrow x')$ be the probability that the process makes a transition from state x to state x' . This transition probability defines what is called a **Markov chain** on the state space. Now suppose that we run the Markov chain for t steps, and let $\pi_t(x)$ be the probability that the system is in state x at time t . Similarly, let $\pi_{t+1}(x')$ be the probability of being in state x' at time $t + 1$. Given $\pi_t(x)$, we can calculate $\pi_{t+1}(x')$ by summing, for all states the system could be in at time t , the probability of being in that state times the probability of making the transition to x' :



$$\pi_{t+1}(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') .$$

We say that the chain has reached its **stationary distribution** if $\pi_t = \pi_{t+1}$. Let us call this stationary distribution π ; its defining equation is therefore

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \quad \text{for all } \mathbf{x}' .$$

Provided the transition probability distribution q is **ergodic**—that is, every state is reachable from every other and there are no strictly periodic cycles—there is exactly one distribution π satisfying this equation for any given q .

$$\pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) \quad \text{for all } \mathbf{x}, \mathbf{x}' .$$

When these equations hold, we say that $q(\mathbf{x} \rightarrow \mathbf{x}')$ is in **detailed balance** with $\pi(\mathbf{x})$.

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}')$$

References:

- Artificial Intelligence A Modern Approach Third Edition Stuart J. Russell and Peter Norvig.



Piyush Wairale

GATE in Data Science and AI study material

For GATE DA full test series & study materials. Visit piyushwairale.com

For GATE DA Crash Course, visit: www.piyushwairale.com



Reasoning under uncertainty topics- Exact inference through variable elimination.

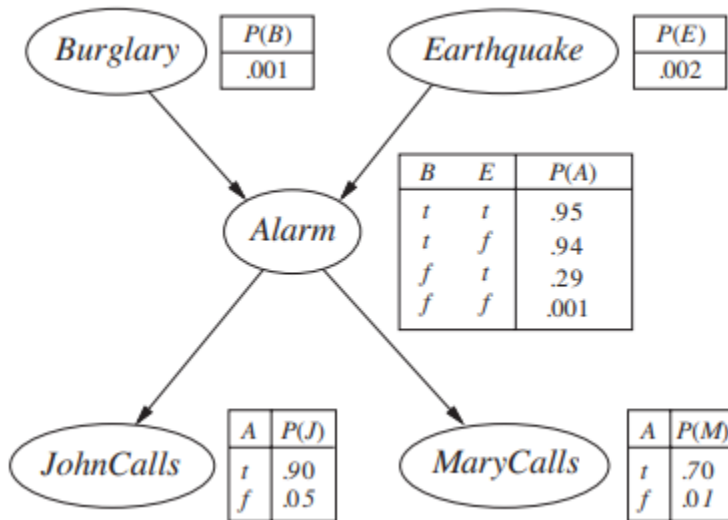
A Bayesian network is a directed graph in which each node is annotated with quantitative probability information. The full specification is as follows:

1. Each node corresponds to a random variable, which may be discrete or continuous.
2. A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y , X is said to be a parent of Y . The graph has no directed cycles (and hence is a directed acyclic graph, or DAG).
3. Each node X_i has a conditional probability distribution $P(X_i | \text{Parents}(X_i))$ that quantifies the effect of the parents on the node.

1.EXACT INFERENCE IN BAYESIAN NETWORKS

The basic task for any probabilistic inference system is to compute the posterior probability distribution for a set of **query variables**, given some observed **event**—that is, some assignment of values to a set of **evidence variables**. To simplify the presentation, we will consider only one query variable at a time; the algorithms can easily be extended to queries with multiple variables. We will use the notation from Chapter 13: X denotes the query variable; E denotes the set of evidence variables E_1, \dots, E_m , and e is a particular observed event; Y will denote the nonevidence, nonquery variables Y_1, \dots, Y_l (called the **hidden variables**). Thus, the complete set of variables is $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$. A typical query asks for the posterior probability distribution $P(X | e)$.

Example: You have a new burglar alarm installed at home. It is fairly reliable at detecting a burglary, but also responds on occasion to minor earthquakes. (This example is due to Judea Pearl, a resident of Los Angeles—hence the acute interest in earthquakes.) You also have two neighbors, John and Mary, who have promised to call you at work when they hear the alarm. John nearly always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too. Mary, on the other hand, likes rather loud music and often misses the alarm altogether. Given the evidence of who has or has not called, we would like to estimate the probability of a burglary.



The network structure shows that burglary and earthquakes directly affect the probability of the alarm's going off, but whether John and Mary call depends only on the alarm. The network thus represents our assumptions that they do not perceive burglaries directly, they do not notice minor earthquakes, and they do not confer before calling.

In the burglary network, we might observe the event in which JohnCalls = true and MaryCalls = true. We could then ask for, say, the probability that a burglary has occurred:

$P(\text{Burglary} \mid \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true}) = 0.284, 0.716$.

In this section we discuss exact algorithms for computing posterior probabilities and will consider the complexity of this task.

1.1 Inference by enumeration:

Any conditional probability can be computed by summing terms from the full joint distribution. More specifically, a query $P(X \mid e)$ can be answered using this Equation

$$P(X \mid e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y), \quad \text{---1}$$

, which we repeat here for convenience:

$$P(X \mid e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y).$$



Now, a Bayesian network gives a complete representation of the full joint distribution. More specifically, the terms $P(x, e, y)$ in the joint distribution can be written as products of conditional probabilities from the network. Therefore, *a query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.*

Consider the query $P(\text{Burglary} \mid \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true})$. The hidden variables for this query are Earthquake and Alarm. From Equation (1), using initial letters for the variables to shorten the expressions, we have

$$\mathbf{P}(B \mid j, m) = \alpha \mathbf{P}(B, j, m) = \alpha \sum_e \sum_a \mathbf{P}(B, j, m, e, a,) .$$

The semantics of Bayesian networks (Equation (14.2)) then gives us an expression in terms of CPT entries. For simplicity, we do this just for Burglary = true:

$$P(b \mid j, m) = \alpha \sum_e \sum_a P(b)P(e)P(a \mid b, e)P(j \mid a)P(m \mid a) .$$

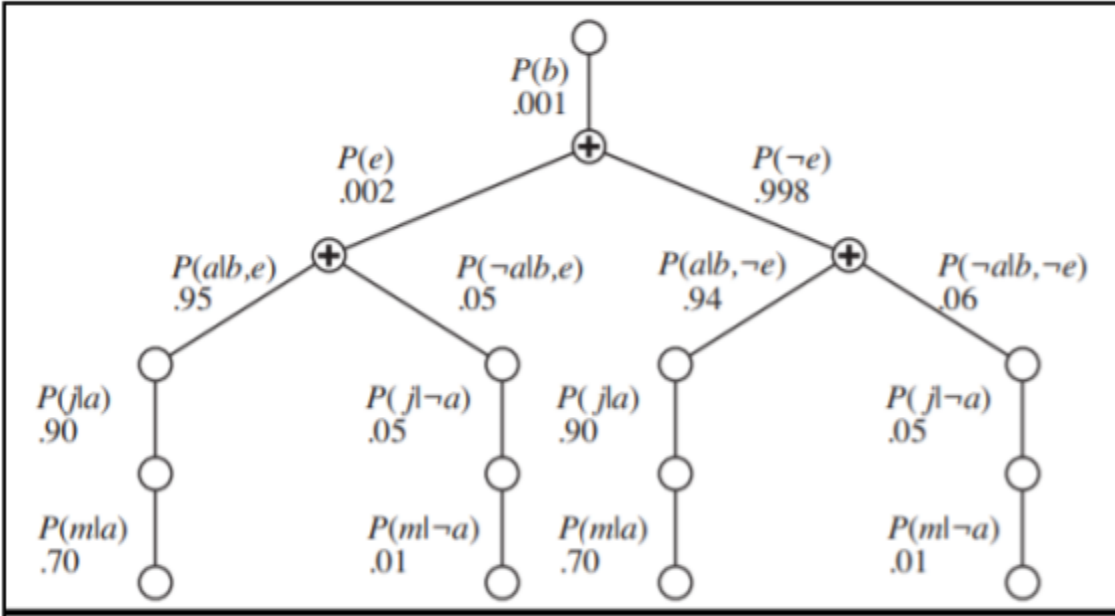
To compute this expression, we have to add four terms, each computed by multiplying five numbers. In the worst case, where we have to sum out almost all the variables, the complexity of the algorithm for a network with n Boolean variables is $O(n^2)$.

An improvement can be obtained from the following simple observations: the $P(b)$ term is a constant and can be moved outside the summations over a and e , and the $P(e)$ term can be moved outside the summation over a . Hence, we have

$$P(b \mid j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e)P(j \mid a)P(m \mid a) . \quad - (2)$$

1.2 The variable elimination algorithm:

The enumeration algorithm can be improved substantially by eliminating repeated calculations of the kind illustrated in below Figure. The idea is simple: do the calculation once and save the results for later use. This is a form of dynamic programming. There are several versions of this approach; we present the **variable elimination** algorithm, which is the simplest. Variable elimination works by evaluating expressions such as equation(2) in right-to-left order (that is, bottom up in Figure). Intermediate results are stored, and summations over each variable are done only for those portions of the expression that depend on the variable.



The structure of the expression shown in Equation (2). The evaluation proceeds top down, multiplying values along each path and summing at the “+” nodes. Notice the repetition of the paths for j and m.

Let us illustrate this process for the burglary network. We evaluate the expression

$$\mathbf{P}(B | j, m) = \alpha \underbrace{\mathbf{P}(B)}_{\mathbf{f}_1(B)} \sum_e \underbrace{P(e)}_{\mathbf{f}_2(E)} \sum_a \underbrace{P(a | B, e)}_{\mathbf{f}_3(A, B, E)} \underbrace{P(j | a)}_{\mathbf{f}_4(A)} \underbrace{P(m | a)}_{\mathbf{f}_5(A)} .$$

Notice that we have annotated each part of the expression with the name of the corresponding factor; each factor is a matrix indexed by the values of its argument variables. For example, the factors $\mathbf{f}_4(A)$ and $\mathbf{f}_5(A)$ corresponding to $P(j | a)$ and $P(m | a)$ depend just on A because J and M are fixed by the query. They are therefore two-element vectors:

$$\mathbf{f}_4(A) = \begin{pmatrix} P(j | a) \\ P(j | \neg a) \end{pmatrix} = \begin{pmatrix} 0.90 \\ 0.05 \end{pmatrix} \quad \mathbf{f}_5(A) = \begin{pmatrix} P(m | a) \\ P(m | \neg a) \end{pmatrix} = \begin{pmatrix} 0.70 \\ 0.01 \end{pmatrix} .$$

$\mathbf{f}_3(A, B, E)$ will be a $2 \times 2 \times 2$ matrix, which is hard to show on the printed page. (The “first” element is given by $P(a | b, e)=0.95$ and the “last” by $P(\neg a | \neg b, \neg e)=0.999$.) In terms of factors, the query expression is written as

$$\mathbf{P}(B | j, m) = \alpha \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \sum_a \mathbf{f}_3(A, B, E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A)$$



```

function ENUMERATION-ASK( $X, \mathbf{e}, bn$ ) returns a distribution over  $X$ 
  inputs:  $X$ , the query variable
            $\mathbf{e}$ , observed values for variables  $\mathbf{E}$ 
            $bn$ , a Bayes net with variables  $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$  /*  $\mathbf{Y} = \text{hidden variables}$  */

   $Q(X) \leftarrow$  a distribution over  $X$ , initially empty
  for each value  $x_i$  of  $X$  do
     $Q(x_i) \leftarrow$  ENUMERATE-ALL( $bn.VARS, \mathbf{e}_{x_i}$ )
    where  $\mathbf{e}_{x_i}$  is  $\mathbf{e}$  extended with  $X = x_i$ 
  return NORMALIZE( $Q(X)$ )

```

```

function ENUMERATE-ALL( $vars, \mathbf{e}$ ) returns a real number
  if EMPTY?( $vars$ ) then return 1.0
   $Y \leftarrow$  FIRST( $vars$ )
  if  $Y$  has value  $y$  in  $\mathbf{e}$ 
    then return  $P(y | \text{parents}(Y)) \times$  ENUMERATE-ALL(REST( $vars$ ),  $\mathbf{e}$ )
  else return  $\sum_y P(y | \text{parents}(Y)) \times$  ENUMERATE-ALL(REST( $vars$ ),  $\mathbf{e}_y$ )
  where  $\mathbf{e}_y$  is  $\mathbf{e}$  extended with  $Y = y$ 

```

The enumeration algorithm for answering queries on Bayesian network.

where the “ \times ” operator is not ordinary matrix multiplication but instead the pointwise product operation, to be described shortly.

The process of evaluation is a process of summing out variables (right to left) from pointwise products of factors to produce new factors, eventually yielding a factor that is the solution, i.e., the posterior distribution over the query variable. The steps are as follows:

- First, we sum out A from the product of f_3 , f_4 , and f_5 . This gives us a new 2×2 factor $f_6(B, E)$ whose indices range over just B and E :

$$\begin{aligned}
 f_6(B, E) &= \sum_a f_3(A, B, E) \times f_4(A) \times f_5(A) \\
 &= (f_3(a, B, E) \times f_4(a) \times f_5(a)) + (f_3(\neg a, B, E) \times f_4(\neg a) \times f_5(\neg a)).
 \end{aligned}$$

Now we are left with the expression

$$P(B | j, m) = \alpha f_1(B) \times \sum_e f_2(E) \times f_6(B, E).$$

- Next, we sum out E from the product of f_2 and f_6 :



$$\begin{aligned} \mathbf{f}_7(B) &= \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B, E) \\ &= \mathbf{f}_2(e) \times \mathbf{f}_6(B, e) + \mathbf{f}_2(\neg e) \times \mathbf{f}_6(B, \neg e) . \end{aligned}$$

This leaves the expression

$$\mathbf{P}(B \mid j, m) = \alpha \mathbf{f}_1(B) \times \mathbf{f}_7(B)$$

which can be evaluated by taking the pointwise product and normalizing the result.

Examining this sequence, we see that two basic computational operations are required: pointwise product of a pair of factors, and summing out a variable from a product of factors. The next section describes each of these operations.

Variable ordering and variable relevance

An unspecified ORDER function to choose an ordering for the variables. Every choice of ordering yields a valid algorithm, but different orderings cause different intermediate factors to be generated during the calculation. For example, in the calculation shown previously, we eliminated A before E; if we do it the other way, the calculation becomes

$$\mathbf{P}(B \mid j, m) = \alpha \mathbf{f}_1(B) \times \sum_a \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A, B, E) ,$$

during which a new factor $\mathbf{f}_6(A, B)$ will be generated.

In general, the time and space requirements of variable elimination are dominated by the size of the largest factor constructed during the operation of the algorithm.

function ELIMINATION-ASK(X, \mathbf{e}, bn) **returns** a distribution over X

inputs: X , the query variable

\mathbf{e} , observed values for variables \mathbf{E}

bn , a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \dots, X_n)$

$factors \leftarrow []$

for each var **in** ORDER($bn.VARS$) **do**

$factors \leftarrow [MAKE-FACTOR(var, \mathbf{e}) | factors]$

if var is a hidden variable **then** $factors \leftarrow SUM-OUT(var, factors)$

return NORMALIZE(POINTWISE-PRODUCT($factors$))

The variable elimination algorithm for inference in Bayesian networks.

This is turn is determined by the order of elimination of variables and by the structure of the network. It turns out to be intractable to determine the optimal ordering, but several good



heuristics are available. One fairly effective method is a greedy one: eliminate whichever variable minimizes the size of the next factor to be constructed.

Let us consider one more query: $P(\text{JohnCalls} \mid \text{Burglary} = \text{true})$. As usual, the first step is to write out the nested summation:

$$P(J \mid b) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e) P(J \mid a) \sum_m P(m \mid a) .$$

Evaluating this expression from right to left, we notice something interesting: $\sum_m P(m \mid a)$ is equal to 1 by definition! Hence, there was no need to include it in the first place; the variable M is irrelevant to this query. Another way of saying this is that the result of the query $P(\text{JohnCalls} \mid \text{Burglary} = \text{true})$ is unchanged if we remove *MaryCalls* from the network altogether. In general, we can remove any leaf node that is not a query variable or an evidence variable. After its removal, there may be some more leaf nodes, and these too may be irrelevant. Continuing this process, we eventually find that *every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query*. A variable elimination algorithm can therefore remove all these variables before evaluating the query.

1.3 The complexity of exact inference:

The complexity of exact inference in Bayesian networks depends strongly on the structure of the network. The burglary network of Figure 14.2 belongs to the family of networks in which there is at most one undirected path between any two nodes in the network. These are called **singly connected** networks or **polytrees**, and they have a particularly nice property: *The time and space complexity of exact inference in polytrees is linear in the size of the network*. Here, the size is defined as the number of CPT entries; if the number of parents of each node is bounded by a constant, then the complexity will also be linear in the number of nodes.

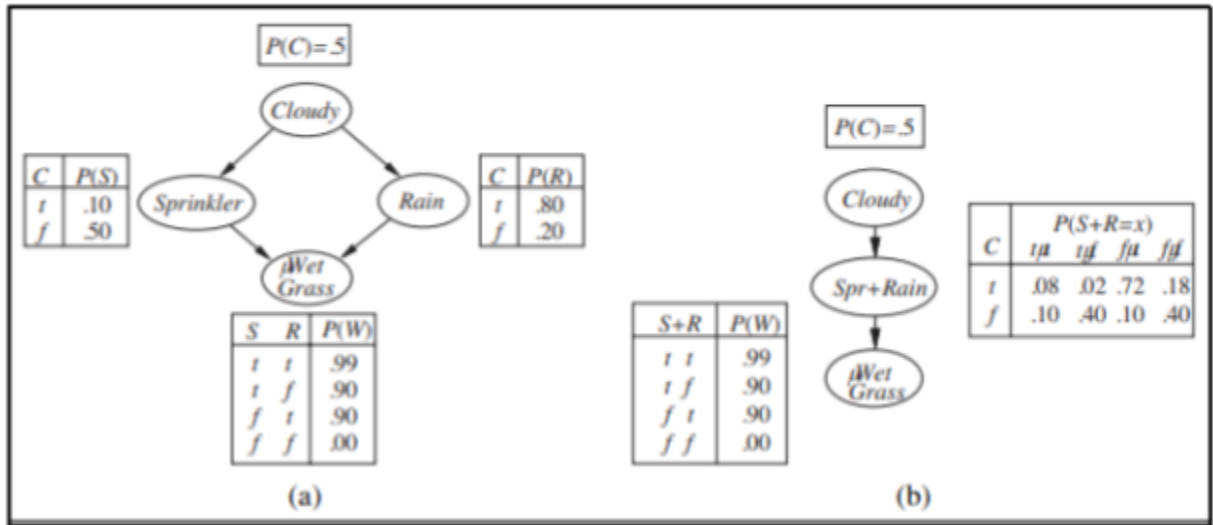
For **multiply connected** networks, such as that of Figure , variable elimination can have exponential time and space complexity in the worst case, even when the number of parents per node is bounded. This is not surprising when one considers that *because it includes inference in propositional logic as a special case, inference in Bayesian networks is NP-hard*.

There is a close connection between the complexity of Bayesian network inference and the complexity of constraint satisfaction problems (CSPs). Measures such as tree width, which bound the complexity of solving a CSP, can also be applied directly to Bayesian networks.



Moreover, the variable elimination algorithm can be generalized to solve CSPs as well as Bayesian networks.

There is a close connection between the complexity of Bayesian network inference and the complexity of constraint satisfaction problems (CSPs). Measures such as tree width, which bound the complexity of solving a CSP, can also be applied directly to Bayesian networks. Moreover, the variable elimination algorithm can be generalized to solve CSPs as well as Bayesian networks.



(a) A multiply connected network with conditional probability tables. (b) A clustered equivalent of the multiply connected network.

There is a close connection between the complexity of Bayesian network inference and the complexity of constraint satisfaction problems (CSPs). Measures such as tree width, which bound the complexity of solving a CSP, can also be applied directly to Bayesian networks. Moreover, the variable elimination algorithm can be generalized to solve CSPs as well as Bayesian networks.

References:

- <https://www.cs.ubc.ca/~jordon/teaching/cpsc322/2019w2/lectures/lecture29.pdf>
- <https://www.cs.ubc.ca/~hutter/teaching/cpsc322/6-Uncertainty6.pdf>
- <https://archive.nptel.ac.in/courses/106/105/106105078/>
- Artificial Intelligence A Modern Approach Third Edition Stuart J. Russell and Peter Norvig.