Artificial Intelligence

Graphical Models

Marc Toussaint
University of Stuttgart
Winter 2019/20

Motivation:

Graphical models are a generic language to express "structured" probabilistic models. Structured simply means that we talk about many random variables and many coupling terms, where each coupling term concerns only a (usually small) subset of random variables. so, structurally they are very similar to CSPs. But the coupling terms are not boolean functions but real-valued functions, called factors. And that defines a probability distribution over all RVs. The problem then is either to find the most probable value assignment to all RVs (called MAP inference problem), or to find the probabilities over the values of a single variable that arises from the couplings (called marginal inference).

There are so many applications of graphical models that is it hard to pick some to list: Modelling gene networks (e.g. to understand genetic diseases), structured text models (e.g. to cluster text into topics), modelling dynamic processes like music or human activities (like cooking or so), modelling more structured Markov Decision Processes (hierarchical RL, POMDPs, etc), modelling multi-agent systems, localization and mapping of mobile robots, and also many of the core ML methods can be expressed as graphical models, e.g. Bayesian (kernel) logistic/ridge regression, Gaussian mixture models, clustering methods, many unsupervised learning methods, ICA, PCA, etc. It is though fair to say that these methods do not *have* to be expressed as graphical models; but they can be and I think it is very helpful to see the underlying principles of these methods when expressing them in terms of graphical models. And graphical models then allow you to invent variants/combinations of such methods specifically for your particular data domain.

In this lecture we introduce Bayesian networks and factor graphs and discuss probabilistic inference methods. Exact inference amounts to summing over variables in a certain order. This can be automated in a way that exploits the graph structure, leading to what is called variable

elimination and message passing on trees. The latter is perfectly analogous to constraint propagation to exactly solve tree CSPs. For non-trees, message passing becomes loopy belief propagation, which approximates a solution. Monte-Carlo sampling methods are also important tools for approximate inference, which are beyond this lecture though.

Bayes Nets and Conditional Independence

Outline

- A. Introduction
  – Motivation and definition of Bayes Nets
  – Conditional independence in Bayes Nets
  – Examples
- B. Inference in Graphical Models
  – Variable Elimination & Factor Graphs
  – Message passing, Loopy Belief Propagation
  – Sampling methods (Rejection, Importance, Gibbs)

Graphical Models

- The core difficulty in modelling is specifying

  What are the relevant variables?

  How do they depend on each other?

  (Or how *could* they depend on each other $\rightarrow$ learning)

- **Graphical models** are a graphical notation for

  1) which random variables exist

  2) which random variables are directly coupled, and how

  Thereby they *describe a joint probability distribution* $P(X_1, .., X_n)$ over n random variables.

- 2 basic variants:
  - Bayesian Networks (aka. directed model, belief network)
  - Factor Graphs (aka. undirected model, Markov Random Field)

Example

drinking red wine $\rightarrow$ longevity?

Bayesian Networks

- A **Bayesian Network** is a
  - directed acyclic graph (DAG)
  - where each node represents a random variable X_i
  - for each node we have a conditional probability distribution
$$P(X_i \mid \text{Parents}(X_i))$$

- In the simplest case (discrete RVs), the conditional distribution is represented as a conditional probability table (**CPT**)

Bayesian Networks

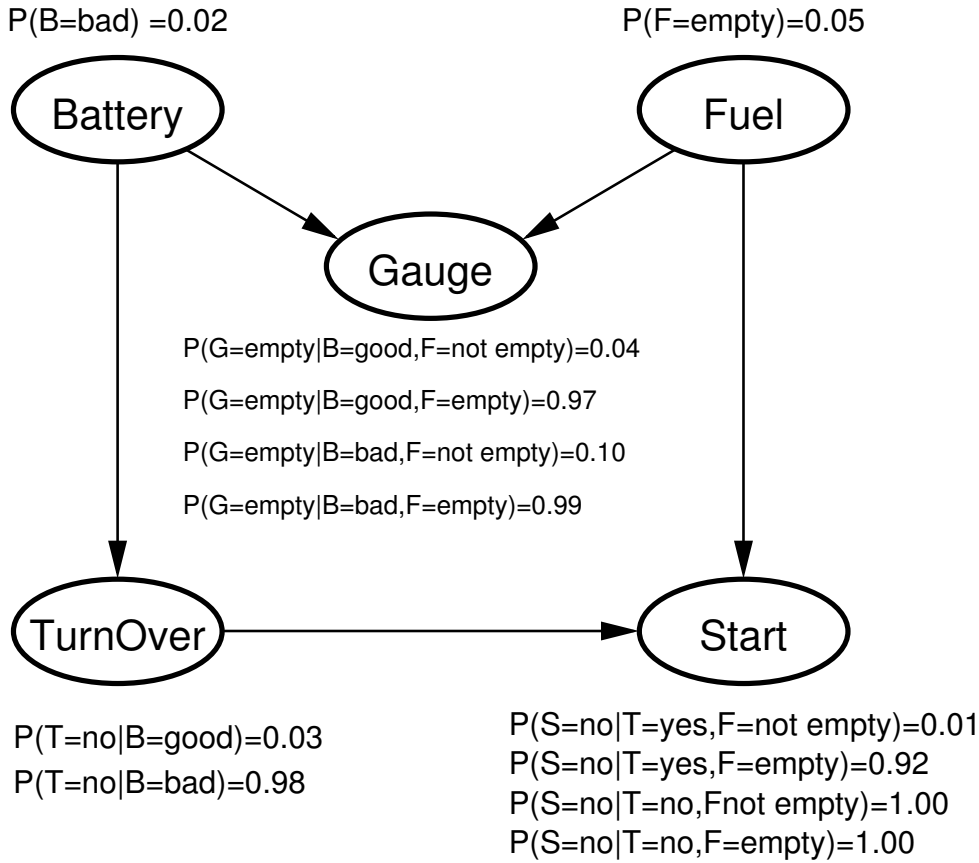
- DAG $\rightarrow$ we can sort the RVs; edges only go from lower to higher index
- **The joint distribution can be factored as**

$$P(X_{1:n}) = \prod_{i=1}^{n} P(X_i \mid \text{Parents}(X_i))$$

- Missing links imply conditional independence
- Forward sampling from joint distribution

Example

(Heckermann 1995)



$$\iff \quad P(S,T,G,F,B) = P(B)\ P(F)\ P(G|F,B)\ P(T|B)\ P(S|T,F)$$

- Table sizes: LHS = $2^5 - 1 = 31$ RHS = $1 + 1 + 4 + 2 + 4 = 12$

Bayes Nets & conditional independence

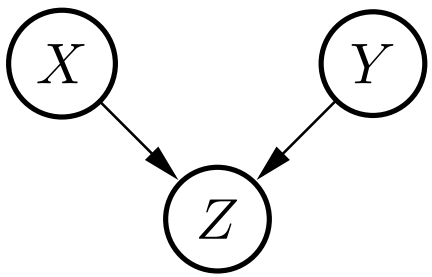
- Independence: $Indep(X, Y) \iff P(X, Y) = P(X)\ P(Y)$
- Conditional independence:

$$Indep(X, Y|Z) \iff P(X, Y|Z) = P(X|Z)\ P(Y|Z)$$

Bayes Nets & conditional independence

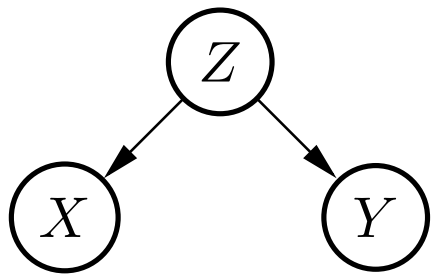
- Independence: $Indep(X, Y) \iff P(X, Y) = P(X)\ P(Y)$
- Conditional independence:

$$Indep(X, Y|Z) \iff P(X, Y|Z) = P(X|Z)\ P(Y|Z)$$



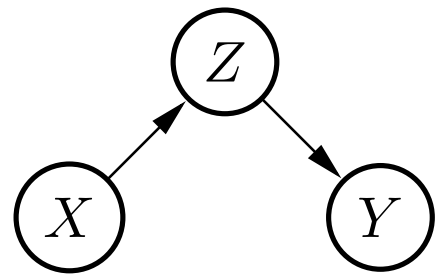
(head-to-head)

$$Indep(X, Y)$$
$$\neg Indep(X, Y|Z)$$



(tail-to-tail)

$$\neg Indep(X, Y)$$
$$Indep(X, Y|Z)$$



(head-to-tail)

$$\neg Indep(X, Y)$$
$$Indep(X, Y|Z)$$

- Head-to-head: $Indep(X, Y)$
  $P(X, Y, Z) = P(X)\ P(Y)\ P(Z|X, Y)$
  $P(X, Y) = P(X)\ P(Y)\ \sum_Z P(Z|X, Y) = P(X)\ P(Y)$

- Tail-to-tail: $Indep(X, Y|Z)$
  $P(X, Y, Z) = P(Z)\ P(X|Z)\ P(Y|Z)$
  $P(X, Y|Z) = P(X, Y, Z)/P(Z) = P(X|Z)\ P(Y|Z)$

- Head-to-tail: $Indep(X, Y|Z)$
  $P(X, Y, Z) = P(X)\ P(Z|X)\ P(Y|Z)$
  $P(X, Y|Z) = \frac{P(X,Y,Z)}{P(Z)} = \frac{P(X,Z)\ P(Y|Z)}{P(Z)} = P(X|Z)\ P(Y|Z)$

General rules for determining conditional independence in a Bayes net:

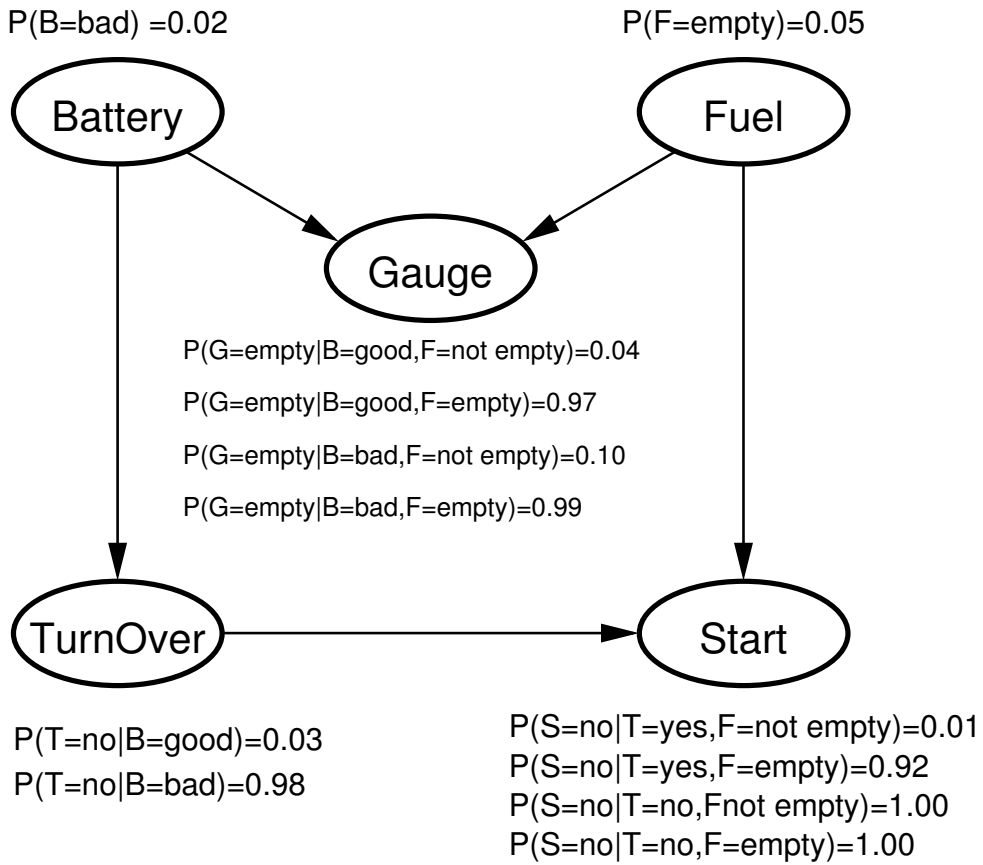
- Given three groups of random variables X, Y, Z

  $Indep(X, Y|Z) \iff$ every path from X to Y is "blocked by Z"

- A path is "blocked by Z" $\iff$ on this path...
  - $\exists$ a node in Z that is head-to-tail w.r.t. the path, or
  - $\exists$ a node in Z that is tail-to-tail w.r.t. the path, or
  - $\exists$ another node A which is head-to-head w.r.t. the path and neither A nor any of its descendants are in Z

Example

(Heckermann 1995)



$Indep(T, F)$? $Indep(B, F|S)$? $Indep(B, S|T)$?

What can we do with Bayes nets?

- **Inference:** Given some pieces of information (prior, observed variabes) what is the implication (the implied information, the posterior) on a non-observed variable

- **Decision Making:** If utilities and decision variables are defined $\rightarrow$ compute optimal decisions in probabilistic domains

- **Learning:**
  - Fully Bayesian Learning: Inference over parameters (e.g., β)
  - Maximum likelihood training: Optimizing parameters

- **Structure Learning** (Learning/Inferring the graph structure itself): Decide which model (which graph structure) fits the data best; thereby uncovering conditional independencies in the data.

Inference

- Inference: Given some pieces of information (prior, observed variabes) what is the implication (the implied information, the posterior) on a non-observed variable

- In a Bayes Nets: Assume there is three groups of RVs:
  - Z are observed random variables
  - X and Y are hidden random variables
  - We want to do inference about X, not Y

  Given some observed variables Z, compute the **posterior marginal** $P(X \mid Z)$ for some hidden variable X.

$$P(X \mid Z) = \frac{P(X, Z)}{P(Z)} = \frac{1}{P(Z)} \sum_{Y} P(X, Y, Z)$$

  where Y are all hidden random variables except for X

- Inference requires summing over *(eliminating)* hidden variables.

Example: Holmes & Watson

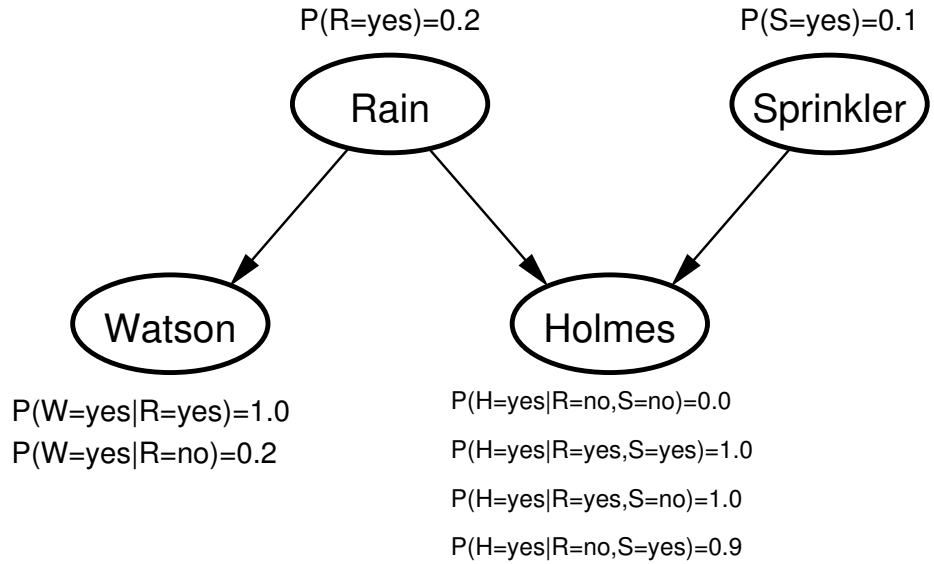
- Mr. Holmes lives in Los Angeles. One morning when Holmes leaves his house, he realizes that his grass is wet. Is it due to rain, or has he forgotten to turn off his sprinkler?
  - Calculate $P(R|H)$, $P(S|H)$ and compare these values to the prior probabilities.
  - Calculate $P(R, S|H)$.
    Note: R and S are marginally independent, but conditionally dependent

- Holmes checks Watson's grass, and finds it is also wet.
  - Calculate $P(R|H, W)$, $P(S|H, W)$
  - This effect is called explaining away

JavaBayes: run it from the html page

`http://www.cs.cmu.edu/~javabayes/Home/applet.html`

Example: Holmes & Watson



$$P(H, W, S, R) = P(H|S, R)\ P(W|R)\ P(S)\ P(R)$$

$$P(R|H) = \sum_{W,S} \frac{P(R, W, S, H)}{P(H)} = \frac{1}{P(H)} \sum_{W,S} P(H|S, R)\ P(W|R)\ P(S)\ P(R)$$

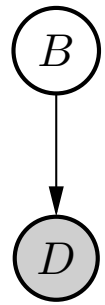
$$= \frac{1}{P(H)} \sum_{S} P(H|S, R)\ P(S)\ P(R)$$

$$P(R{=}1 \mid H{=}1) = \frac{1}{P(H{=}1)}(1.0 \cdot 0.2 \cdot 0.1 + 1.0 \cdot 0.2 \cdot 0.9) = \frac{1}{P(H{=}1)} 0.2$$

$$P(R{=}0 \mid H{=}1) = \frac{1}{P(H{=}1)}(0.9 \cdot 0.8 \cdot 0.1 + 0.0 \cdot 0.8 \cdot 0.9) = \frac{1}{P(H{=}1)} 0.072$$

- These types of calculations can be automated
  $\rightarrow$ Variable Elimination Algorithm

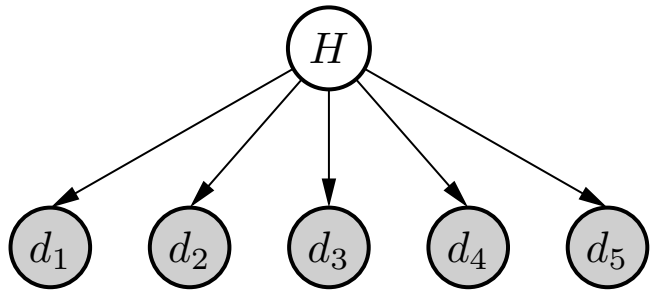
Example: Bavarian dialect



- Two binary random variables(RVs): B (bavarian) and D (dialect)
- Given:
  $P(D, B) = P(D \mid B)\ P(B)$
  $P(D{=}1 \mid B{=}1) = 0.4$, $P(D{=}1 \mid B{=}0) = 0.01$, $P(B{=}1) = 0.15$

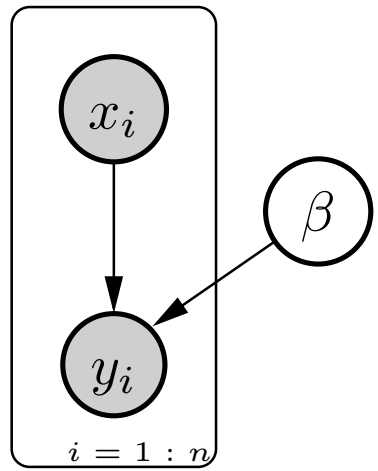
- **Notation:** Grey shading usually indicates “observed”

Example: Coin flipping



- One binary RV H (hypothesis), 5 RVs for the coin tosses $d_1, .., d_5$
- Given:
  $P(D, H) = \prod_i P(d_i \mid H)\ P(H)$
  $P(H=1) = \frac{999}{1000}$, $P(d_i = \mathtt{H} \mid H=1) = \frac{1}{2}$, $P(d_i = \mathtt{H} \mid H=2) = 1$

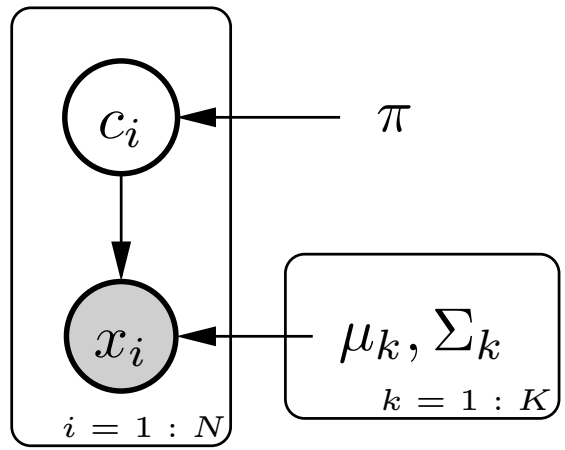
Example: Ridge regression**



- One multi-variate RV β, $2n$ RVs $x_{1:n}$, $y_{1:n}$ (observed data)
- Given:
  $P(D, \beta) = \prod_i \Big[P(y_i \,|\, x_i, \beta)\; P(x_i)\Big]\; P(\beta)$
  $P(\beta) = \mathcal{N}(\beta \,|\, 0, \frac{\sigma^2}{\lambda})$, $P(y_i \,|\, x_i, \beta) = \mathcal{N}(y_i \,|\, x_i^\top \beta, \sigma^2)$

- **Plate notation:** Plates (boxes with index ranges) mean "copy n-times"

Example: Gaussian Mixture Model**



- Discrete latent RVs $c_{1:n}$ indicating mixture component, cont. RVs $x_{1:n}$ (observed data)
- Model: $P(x_i \mid \mu_{1:K}, \Sigma_{1:K}) = \sum_{k=1}^{K} \mathcal{N}(x_i \mid \mu_k, \Sigma_k)\ P(c_i = k)$

Inference Methods in Graphical Models

Inference methods in graphical models

- **Message passing:**
  - Exact inference on trees (includes the Junction Tree Algorithm)
  - Belief propagation

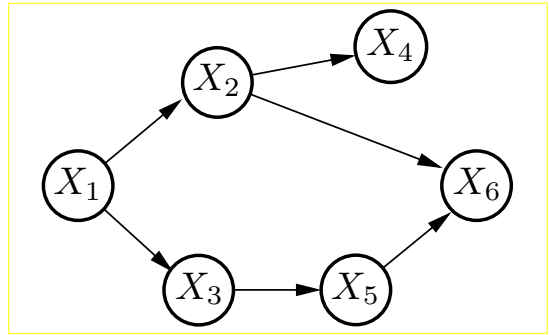
- **Sampling:**
  - Rejection samping, importance sampling, Gibbs sampling
  - More generally, Markov-Chain Monte Carlo (MCMC) methods

- **Other approximations/variational methods**
  - Expectation propagation
  - Specialized variational methods depending on the model

- **Reductions:**
  - Mathematical Programming (e.g. LP relaxations of MAP)
  - Compilation into Arithmetic Circuits (Darwiche at al.)

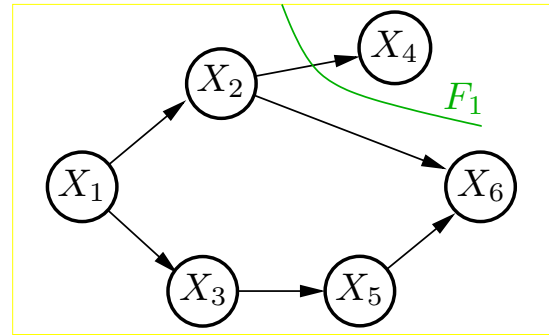
Variable Elimination

Variable Elimination example



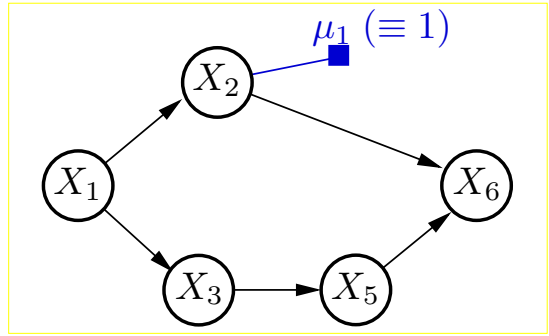
$$P(x_5)$$
$$= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\ P(x_2|x_1)\ P(x_3|x_1)\ P(x_4|x_2)\ P(x_5|x_3)\ P(x_6|x_2,x_5)$$

Variable Elimination example



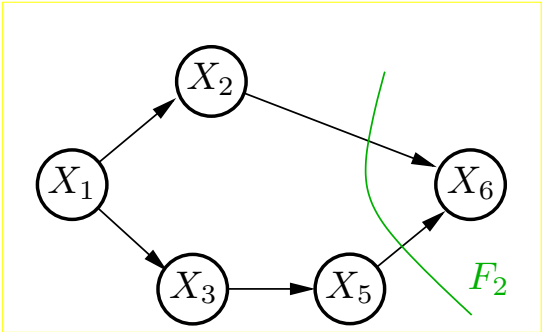
$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)}
\end{aligned}
$$

Variable Elimination example



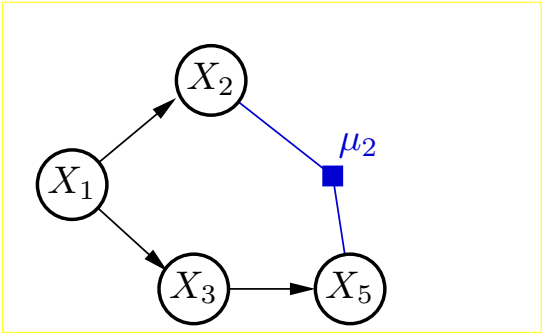
$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\ P(x_2|x_1)\ P(x_3|x_1)\ P(x_4|x_2)\ P(x_5|x_3)\ P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\ P(x_2|x_1)\ P(x_3|x_1)\ P(x_5|x_3)\ P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\ P(x_2|x_1)\ P(x_3|x_1)\ P(x_5|x_3)\ P(x_6|x_2,x_5)\ \mu_1(x_2)
\end{aligned}
$$

Variable Elimination example



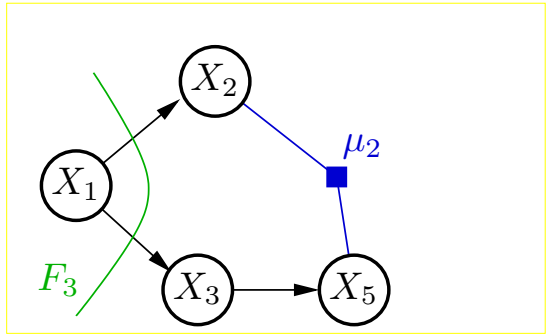
$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2) \sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)}
\end{aligned}
$$

Variable Elimination example



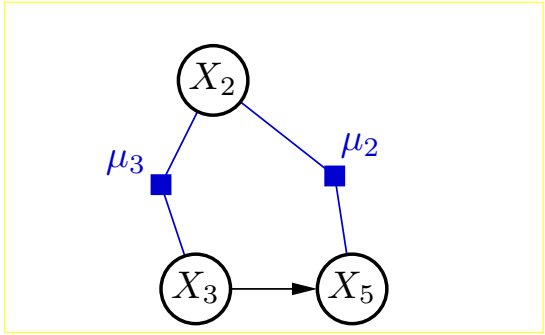
$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2) \sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)} \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5)
\end{aligned}
$$

Variable Elimination example



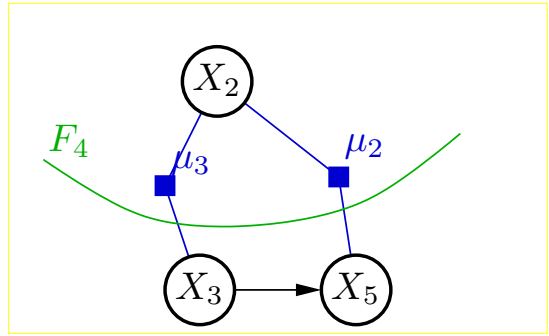
$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2) \sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)} \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \sum_{x_1} \underbrace{P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)}_{F_3(x_1,x_2,x_3)}
\end{aligned}
$$

Variable Elimination example



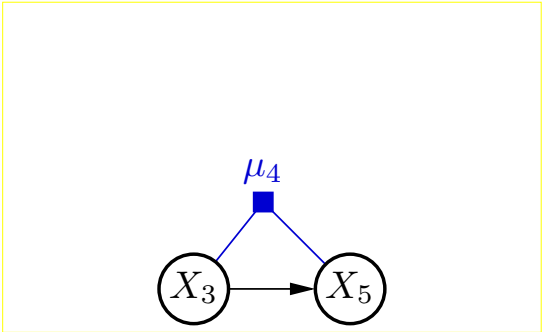
$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2) \sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)} \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \sum_{x_1} \underbrace{P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)}_{F_3(x_1,x_2,x_3)} \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3)
\end{aligned}
$$

Variable Elimination example



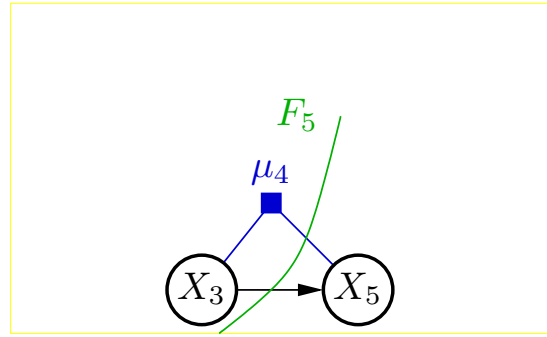
$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2) \sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)} \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \sum_{x_1} \underbrace{P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)}_{F_3(x_1,x_2,x_3)} \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3) \\
&= \sum_{x_3} P(x_5|x_3) \sum_{x_2} \underbrace{\mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3)}_{F_4(x_2,x_3,x_5)}
\end{aligned}
$$

Variable Elimination example



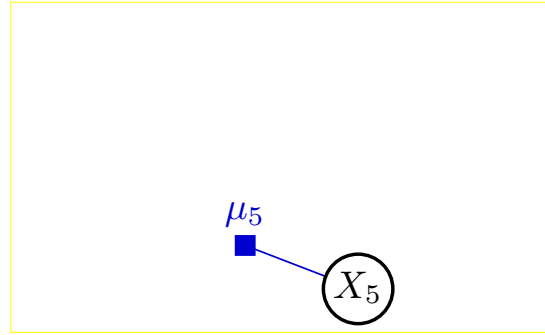
$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2) \sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)} \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \sum_{x_1} \underbrace{P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)}_{F_3(x_1,x_2,x_3)} \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3) \\
&= \sum_{x_3} P(x_5|x_3) \sum_{x_2} \underbrace{\mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3)}_{F_4(x_2,x_3,x_5)} \\
&= \sum_{x_3} P(x_5|x_3)\, \mu_4(x_3,x_5)
\end{aligned}
$$

Variable Elimination example



$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2) \sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)} \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \sum_{x_1} \underbrace{P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)}_{F_3(x_1,x_2,x_3)} \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3) \\
&= \sum_{x_3} P(x_5|x_3) \sum_{x_2} \underbrace{\mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3)}_{F_4(x_2,x_3,x_5)} \\
&= \sum_{x_3} P(x_5|x_3)\, \mu_4(x_3,x_5) \\
&= \sum_{x_3} \underbrace{P(x_5|x_3)\, \mu_4(x_3,x_5)}_{F_5(x_3,x_5)}
\end{aligned}
$$

Variable Elimination example



$$
\begin{aligned}
& P(x_5) \\
&= \sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \sum_{x_4} \underbrace{P(x_4|x_2)}_{F_1(x_2,x_4)} \\
&= \sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2) \sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)} \\
&= \sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5) \sum_{x_1} \underbrace{P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)}_{F_3(x_1,x_2,x_3)} \\
&= \sum_{x_2,x_3} P(x_5|x_3)\, \mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3) \\
&= \sum_{x_3} P(x_5|x_3) \sum_{x_2} \underbrace{\mu_1(x_2)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3)}_{F_4(x_2,x_3,x_5)} \\
&= \sum_{x_3} P(x_5|x_3)\, \mu_4(x_3,x_5) \\
&= \sum_{x_3} \underbrace{P(x_5|x_3)\, \mu_4(x_3,x_5)}_{F_5(x_3,x_5)} \\
&= \mu_5(x_5)
\end{aligned}
$$

Variable Elimination example – lessons learnt

- There is a dynamic programming principle behind Variable Elimination:
  - For eliminating $X_{5,4,6}$ we use the solution of eliminating $X_{4,6}$
  - The "sub-problems" are represented by the F terms, their solutions by the *remaining μ terms*
  - We'll continue to discuss this 4 slides later!

- The factorization of the joint
  - determines in which order Variable Elimination is efficient
  - determines what the terms $F(...)$ and $\mu(...)$ depend on

- We can automate Variable Elimination. For the automation, all that matters is the factorization of the joint.

Factor graphs

- In the previous slides we introduces the box ■ notation to indicate *terms* that depend on some variables. That's exactly what factor graphs represent.

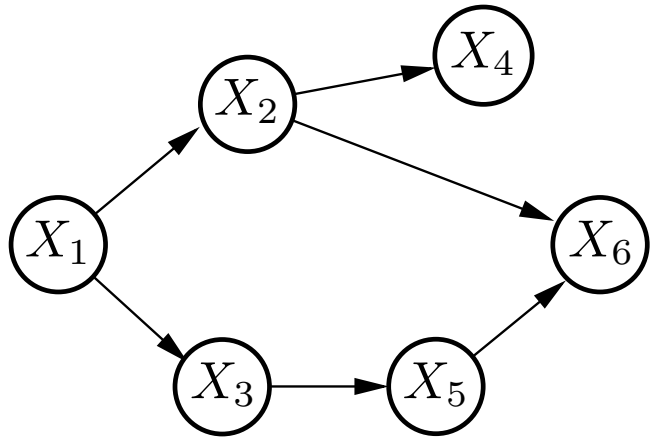
- A **Factor graph** is a
  - bipartite graph
  - where each circle node represents a random variable X_i
  - each box node represents a **factor** f_k, which is a function $f_k(X_{\partial k})$
  - the joint probability distribution is given as

$$P(X_{1:n}) = \prod_{k=1}^{K} f_k(X_{\partial k})$$

Notation: ∂k is shorthand for Neighbors(k)

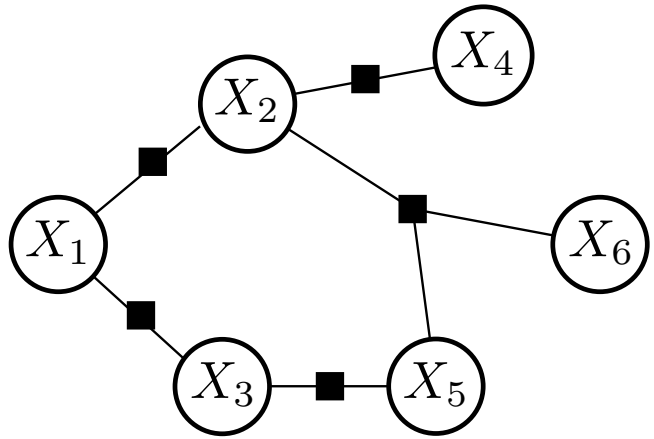
Bayes Net $\rightarrow$ factor graph

- Bayesian Network:



$$P(x_{1:6}) = P(x_1)\ P(x_2|x_1)\ P(x_3|x_1)\ P(x_4|x_2)\ P(x_5|x_3)\ P(x_6|x_2, x_5)$$

- Factor Graph:



$$P(x_{1:6}) = f_1(x_1, x_2)\ f_2(x_3, x_1)\ f_3(x_2, x_4)\ f_4(x_3, x_5)\ f_5(x_2, x_5, x_6)$$

Variable Elimination Algorithm

- `eliminate_single_variable`(F, i)

1: **Input:** list F of factors, variable id i
2: **Output:** list F of factors
3: find relevant subset $\hat{F} \subseteq F$ of factors coupled to i: $\hat{F} = \{k : i \in \partial k\}$
4: create new factor $\hat{k}$ with neighborhood $\partial\hat{k}$ = all variables in $\hat{F}$ except i
5: compute $\mu_{\hat{k}}(X_{\partial\hat{k}}) = \sum_{X_i} \prod_{k \in \hat{F}} f_k(X_{\partial k})$
6: remove old factors $\hat{F}$ and append new factor $\mu_{\hat{k}}$ to F
7: return F

- `elimination_algorithm`(F, M)

1: **Input:** list F of factors, tuple M of desired output variables ids
2: **Output:** single factor μ over variables X_M
3: define all variables present in F: $V = \text{vars}(F)$
4: define variables to be eliminated: $E = V \setminus M$
5: for all $i \in E$: `eliminate_single_variable`(F, i)
6: for all remaining factors, compute the product $\mu = \prod_{f \in F} f$
7: return μ

Conditioning on Observed Variables

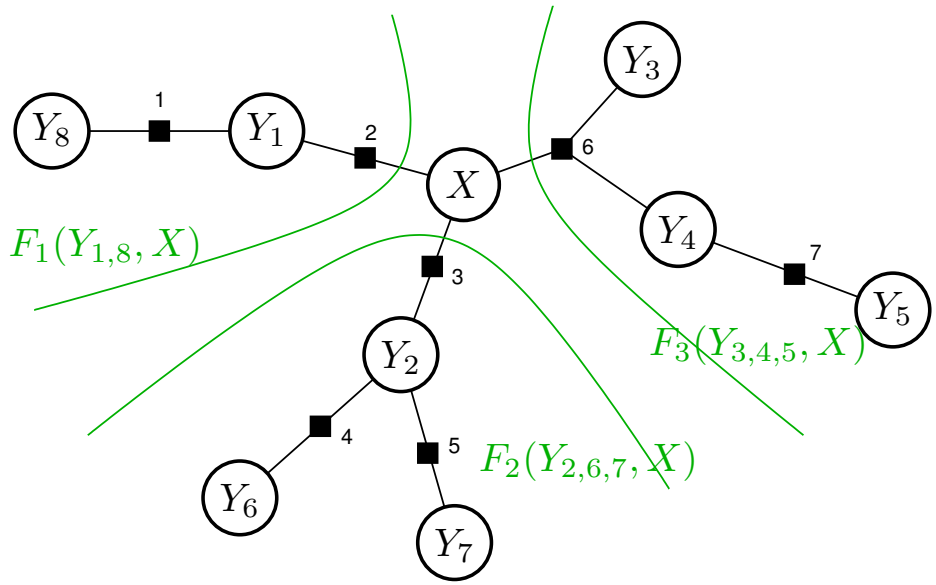
- Before, we defined the general problem of inference as computing

$$P(X \mid Z) = \frac{P(X, Z)}{P(Z)} = \frac{1}{P(Z)} \sum_{Y} P(X, Y, Z)$$

- How can we account for conditioning on Z?

- Answer: Don't sum over Z!
- More practical answer: Add additional "one-hot factors" $\varrho(Z)$ which is zero for all values of Z, except for the observed value, where it is 1.
  - Add such 'conditioning factors' to the list of factors before calling variable elimination – the rest is unchanged
  - Adding such 'conditioning factors' makes the overall factor graph non-normalized, exactly by the evidence $P(Z)$

Variable Elimination on trees



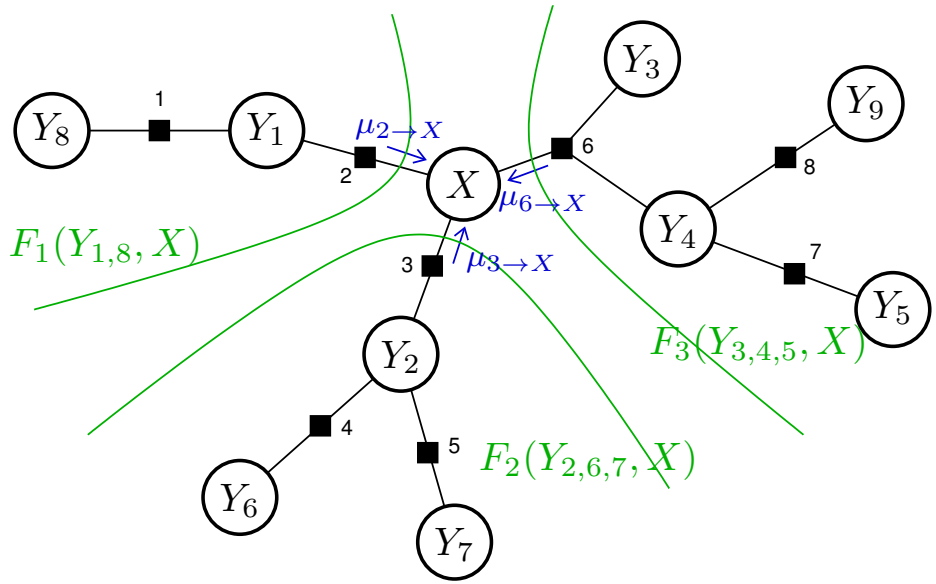
The subtrees w.r.t. X can be described as

$$F_1(Y_{1,8}, X) = f_1(Y_8, Y_1)\ f_2(Y_1, X)$$
$$F_2(Y_{2,6,7}, X) = f_3(X, Y_2)\ f_4(Y_2, Y_6)\ f_5(Y_2, Y_7)$$
$$F_3(Y_{3,4,5}, X) = f_6(X, Y_3, Y_4)\ f_7(Y_4, Y_5)$$

The joint distribution is:

$$P(Y_{1:8}, X) = F_1(Y_{1,8}, X)\ F_2(Y_{2,6,7}, X)\ F_3(Y_{3,4,5}, X)$$

Variable Elimination on trees



We can eliminate each tree independently. The remaining terms (**messages**) are:

$$\mu_{F_1\to X}(X) = \sum_{Y_{1,8}} F_1(Y_{1,8}, X)$$
$$\mu_{F_2\to X}(X) = \sum_{Y_{2,6,7}} F_2(Y_{2,6,7}, X)$$
$$\mu_{F_3\to X}(X) = \sum_{Y_{3,4,5}} F_3(Y_{3,4,5}, X)$$

The marginal $P(X)$ is the **product of subtree messages**

$$P(X) = \mu_{F_1\to X}(X)\ \mu_{F_2\to X}(X)\ \mu_{F_3\to X}(X)$$

Variable Elimination on trees – lessons learnt

- The "remaining terms" μ's are called **messages**
  Intuitively, **messages subsume information from a subtree**

Variable Elimination on trees – lessons learnt

- The "remaining terms" μ's are called **messages**
  Intuitively, **messages subsume information from a subtree**

- Marginal = product of messages, $P(X) = \prod_k \mu_{F_k \to X}$, is very intuitive:
  - *Fusion of independent information* from the different subtrees
  - Fusing independent information $\leftrightarrow$ multiplying probability tables

Variable Elimination on trees – lessons learnt

- The "remaining terms" μ's are called **messages**
  Intuitively, **messages subsume information from a subtree**

- Marginal = product of messages, $P(X) = \prod_k \mu_{F_k \to X}$, is very intuitive:
  - *Fusion of independent information* from the different subtrees
  - Fusing independent information $\leftrightarrow$ multiplying probability tables

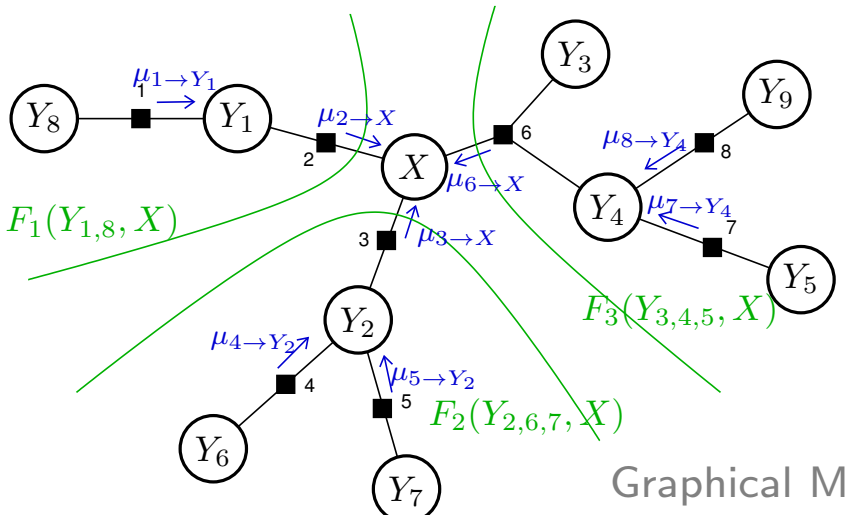
- Along a (sub-) tree, messages can be computed recursively

Message passing

- General equations (**belief propagation (BP)**) for recursive message computation (writing $\mu_{k\to i}(X_i)$ instead of $\mu_{F_k\to X}(X)$):

$$\mu_{k\to i}(X_i) = \sum_{X_{\partial k\setminus i}} \underbrace{f_k(X_{\partial k}) \prod_{j\in\partial k\setminus i} \overbrace{\prod_{k'\in\partial j\setminus k} \mu_{k'\to j}(X_j)}^{\bar{\mu}_{j\to k}(X_j)}}_{F(\text{subtree})}$$

$\prod_{j\in\partial k\setminus i}$: branching at factor k, prod. over adjacent variables j excl. i
$\prod_{k'\in\partial j\setminus k}$: branching at variable j, prod. over adjacent factors k' excl. k
$\bar{\mu}_{j\to k}(X_j)$ are called "variable-to-factor messages": store them for efficiency



Example messages:

$\mu_{2\to X} = \sum_{Y_1} f_2(Y_1, X)\, \mu_{1\to Y_1}(Y_1)$
$\mu_{6\to X} = \sum_{Y_3,Y_4} f_6(Y_3, Y_4, X)\, \mu_{7\to Y_4}(Y_4)\, \mu_{8\to Y_4}($
$\mu_{3\to X} = \sum_{Y_2} f_3(Y_2, X)\, \mu_{4\to Y_2}(Y_2)\, \mu_{5\to Y_2}(Y_2)$

Constraint propagation is 'boolean' message passing

- Assume all factors are binary $\rightarrow$ boolean constraint functions as for CSP
- All messages are binary vectors
- The product of incoming messages indicates the variable's remaining domain D_i (analogous to the marginal $P(X_i)$)
- The message passing equations do constraint propagation

Message passing remarks

- Computing these messages recursively on a tree does nothing else than Variable Elimination
  $\Rightarrow P(X_i) = \prod_{k \in \partial i} \mu_{k \to i}(X_i)$ is the correct posterior marginal

- However, since it stores all "intermediate terms", we can compute ANY marginal $P(X_i)$ for any i

- Message passing exemplifies how to exploit the factorization structure of the joint distribution for the algorithmic implementation

- Note: These are recursive equations. They can be resolved exactly if and only if the dependency structure (factor graph) is a tree. If the factor graph had loops, this would be a "loopy recursive equation system"...

Message passing variants

- Message passing has many important applications:
  - Many models are actually trees: In particular chains esp. *Hidden Markov Models*
  - Message passing can also be applied on non-trees ($\leftrightarrow$ loopy graphs) $\rightarrow$ approximate inference *(Loopy Belief Propagation)*
  - Bayesian Networks can be "squeezed" to become trees $\rightarrow$ exact inference in Bayes Nets! *(Junction Tree Algorithm)*

Loopy Belief Propagation

- If the graphical model is not a tree (=has loops):
  - The recursive message equations cannot be resolved.
  - However, we could try to just iterate them as update equations...

- Loopy BP update equations: (initialize with $\mu_{k\to i} = 1$)

$$\mu_{k\to i}^{\text{new}}(X_i) = \sum_{X_{\partial k\setminus i}} f_k(X_{\partial k}) \prod_{j\in\partial k\setminus i} \prod_{k'\in\partial j\setminus k} \mu_{k'\to j}^{\text{old}}(X_j)$$

Loopy BP remarks

- Problem of loops intuitively:
  loops $\Rightarrow$ branches of a node to not represent independent information!
  – BP is multiplying (=fusing) messages from dependent sources of information

- No convergence guarantee, but if it converges, then to a state of **marginal consistency**

$$\sum_{X_{\partial k \setminus i}} b(X_{\partial k}) = \sum_{X_{\partial k' \setminus i}} b(X_{\partial k'}) = b(X_i)$$

  and to the minimum of the **Bethe approximation** of the free energy (Yedidia, Freeman, & Weiss, 2001)

- We shouldn't be overly disappointed:
  - if BP was exact on loopy graphs we could efficiently solve NP hard problems...
  - loopy BP is a *very* interesting approximation to solving an NP hard problem
- Ways to tackle the problems with BP convergence:
  - Damping (Heskes, 2004: *On the uniqueness of loopy belief propagation fixed points*)
  - CCCP (Yuille, 2002: *CCCP algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation*)
  - Tree-reweighted MP (Kolmogorov, 2006: *Convergent tree-reweighted message passing for energy minimization*)

Maximum a-posteriori (MAP) inference**

- Often we want to compute the most likely global assignment

$$X_{1:n}^{\text{MAP}} = \operatorname*{argmax}_{X_{1:n}} P(X_{1:n})$$

of all random variables. This is called MAP inference and can be solved by replacing all $\sum$ by $\max$ in the message passing equations – the algorithm is called **Max-Product Algorithm** and is a generalization of Dynamic Programming methods like *Viterbi* or *Dijkstra*.

- Application: **Conditional Random Fields**

$$f(y,x) = \phi(y,x)^\top \beta = \sum_{j=1}^{k} \phi_j(y_{\partial j}, x)\beta_j = \log\Big[\prod_{j=1}^{k} e^{\phi_j(y_{\partial j},x)\beta_j}\Big]$$

with prediction $x \mapsto y^*(x) = \operatorname*{argmax}_y f(x,y)$

Finding the argmax is a MAP inference problem! This is frequently needed in the innerloop of CRF learning algorithms.

Conditional Random Fields**

- The following are interchangable:

  "Random Field" $\leftrightarrow$ "Markov Random Field" $\leftrightarrow$ Factor Graph

- Therefore, a CRF is a conditional factor graph:
  - *A CRF defines a mapping from input x to a factor graph over y*
  - Each feature $\phi_j(y_{\partial j}, x)$ depends only on a subset ∂j of variables $y_{\partial j}$
  - If $y_{\partial j}$ are discrete, a feature $\phi_j(y_{\partial j}, x)$ is usually an indicator feature (see lecture 03); the corresponding parameter β_j is then one entry of a factor $f_k(y_{\partial j})$ that couples these variables

Junction Tree Algorithm**

Junction Tree Algorithm**

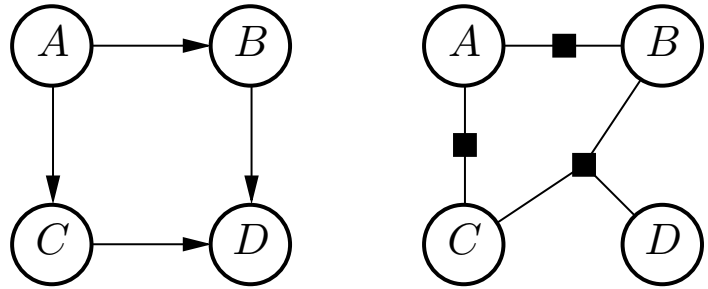
- Many models have loops

  Instead of applying loopy BP in the hope of getting a good approximation, it is possible to convert every model into a tree by redefinition of RVs. The Junction Tree Algorithms converts a loopy model into a tree.

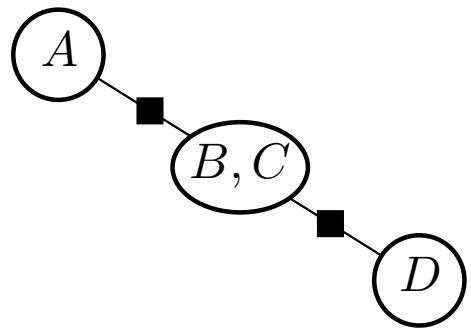
- Loops are resolved by defining larger variable groups (*separators*) on which messages are defined

Junction Tree Example

- Example:



- Join variable B and C to a single **separator**



  This can be viewed as a variable substitution: rename the tuple (B, C) as a single random variable

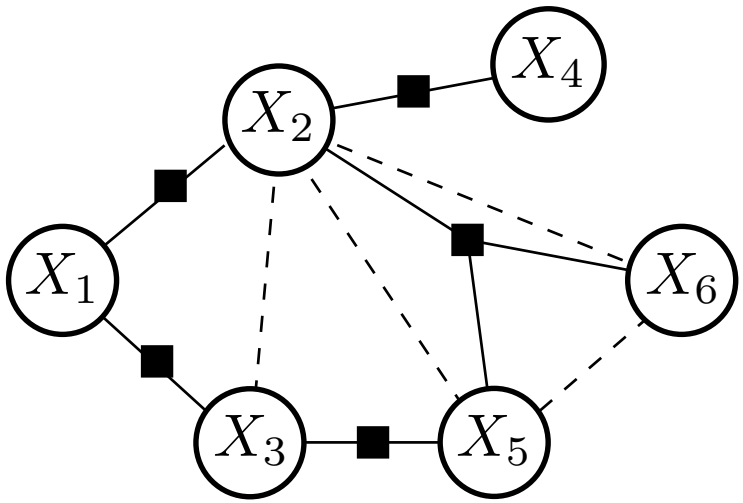
- A single random variable may be part of multiple separators – but only along a *running intersection*

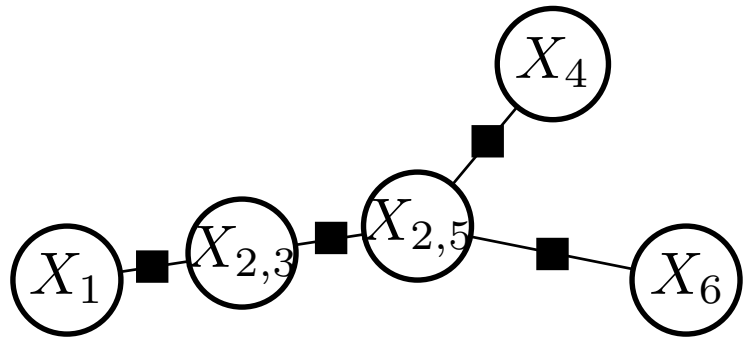
Junction Tree Algorithm

- Standard formulation: *Moralization & Triangulation*
  A **clique** is a fully connected subset of nodes in a graph.
  1) Generate the factor graph (classically called "moralization")
  2) Translate each factor to a clique: Generate the undirected graph where undirected edges connect all RVs of a factor
  3) Triangulate the undirected graph. (This is the critical step!)
  4) Translate each clique back to a factor; identify the separators between factors

- Formulation in terms of variable elimination for a *given* variable order:
  1) Start with a factor graph
  2) Choose an order of variable elimination (This is decided implicitly by trangulation above)
  3) Keep track of the "remaining μ terms" (slide 14): which RVs would they depend on? $\rightarrow$ this identifies the separators

Junction Tree Algorithm Example





- If we eliminate in order 4, 6, 5, 1, 2, 3, we get remaining terms

$$(X_2),\ (X_2, X_5),\ (X_2, X_3),\ (X_2, X_3),\ (X_3)$$

which translates to the Junction Tree on the right

Sampling**

Sampling

- Read:
  Andrieu et al: *An Introduction to MCMC for Machine Learning*
  (Machine Learning, 2003)

- Here I'll discuss only thee basic methods:
  - Rejection sampling
  - Importance sampling
  - Gibbs sampling

Monte Carlo methods

- General, the term *Monte Carlo simulation* refers to methods that generate many i.i.d. random samples $x_i \sim P(x)$ from a distribution $P(x)$. Using the samples one can estimate expectations of anything that depends on x, e.g. $f(x)$:

$$\langle f \rangle \;=\; \int_x P(x)\; f(x)\; dx \;\approx\; \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

  (In this view, Monte Carlo approximates an integral.)

- Example: What is the probability that a solitair would come out successful? (Original story by Stan Ulam.) Instead of trying to analytically compute this, generate many random solitairs and count.

- The method developed in the 40ies, where computers became faster. Fermi, Ulam and von Neumann initiated the idea. von Neumann called it “Monte Carlo” as a code name.

Rejection Sampling

- We have a Bayesian Network with RVs $X_{1:n}$, some of which are observed: $X_{obs} = y_{obs}$, $obs \subset \{1:n\}$

- The goal is to compute marginal posteriors $P(X_i \mid X_{obs} = y_{obs})$ conditioned on the observations.

- We generate a set of K (joint) samples of all variables

$$\mathcal{S} = \{x^k_{1:n}\}^K_{k=1}$$

  Each sample $x^k_{1:n} = (x^k_1, x^k_2, .., x^k_n)$ is a list of instantiation of all RVs.

Rejection Sampling

- To generate a single sample $x^k_{1:n}$:
  1. Sort all RVs in topological order; start with $i = 1$
  2. Sample a value $x^k_i \sim P(X_i \mid x^k_{\text{Parents}(i)})$ for the ith RV conditional to the previous samples $x^k_{1:i\text{-}1}$
  3. If $i \in obs$ compare the sampled value x^k_i with the observation y_i. *Reject* and repeat from a) if the sample is not equal to the observation.
  4. Repeat with $i \leftarrow i + 1$ from 2.
- We compute the marginal probabilities from the sample set $\mathcal{S}$:

$$P(X_i = x \mid X_{obs} = y_{obs}) \approx \frac{\text{count}_{\mathcal{S}}(x^k_i = x)}{K}$$

or pair-wise marginals:

$$P(X_i = x, X_j = x' \mid X_{obs} = y_{obs}) \approx \frac{\text{count}_{\mathcal{S}}(x^k_i = x \wedge x^k_j = x')}{K}$$

Importance Sampling (with likelihood weighting)

- Rejecting whole samples may become very inefficient in large Bayes Nets!

- New strategy: We generate a **weighted** sample set

$$\mathcal{S} = \{(x^k_{1:n}, w^k)\}_{k=1}^K$$

  where each sample $x^k_{1:n}$ is associated with a weight w^k

- In our case, we will choose the weights proportional to the likelihood $P(X_{obs} = y_{obs} \mid X_{1:n} = x^k_{1:n})$ of the observations conditional to the sample $x^k_{1:n}$

Importance Sampling

- To generate a single sample $(w^k, x^k_{1:n})$:
  1. Sort all RVs in topological order; start with $i = 1$ and $w^k = 1$
  2. a) If $i \notin obs$, sample a value $x^k_i \sim P(X_i \mid x^k_{\text{Parents}(i)})$ for the ith RV conditional to the previous samples $x^k_{1:i\text{-}1}$
     b) If $i \in obs$, set the value $x^k_i = y_i$ and update the weight according to likelihood
     $$w^k \leftarrow w^k \; P(X_i = y_i \mid x^k_{1:i\text{-}1})$$
  3. Repeat with $i \leftarrow i + 1$ from 2.
- We compute the marginal probabilities as:

$$P(X_i = x \mid X_{obs} = y_{obs}) \approx \frac{\sum_{k=1}^K w^k [x^k_i = x]}{\sum_{k=1}^K w^k}$$

and likewise pair-wise marginals, etc.
Notation: $[expr] = 1$ if $expr$ is true and zero otherwise

Gibbs Sampling**

- In Gibbs sampling we also generate a sample set $\mathcal{S}$ – but in this case the samples are not independent from each other. The next sample "modifies" the previous one:
- First, all observed RVs are *clamped* to their fixed value $x_i^k = y_i$ for any k.
- To generate the $(k+1)$th sample, iterate through the latent variables $i \notin obs$, updating:

$$\begin{aligned} x_i^{k+1} &\sim P(X_i \,|\, x_{1:n\setminus i}^k) \\ &\sim P(X_i \,|\, x_1^k, x_2^k, .., x_{i\text{-}1}^k, x_{i+1}^k, .., x_n^k) \\ &\sim P(X_i \,|\, x_{\text{Parents}(i)}^k) \prod_{j:i\in\text{Parents}(j)} P(X_j = x_j^k \,|\, X_i, x_{\text{Parents}(j)\setminus i}^k) \end{aligned}$$

That is, each x_i^{k+1} is *resampled* conditional to the other (neighboring) current sample values.

Gibbs Sampling**

- As for rejection sampling, Gibbs sampling generates an unweighted sample set $\mathcal{S}$ which can directly be used to compute marginals.
  In practice, one often discards an initial set of samples (burn-in) to avoid starting biases.

- Gibbs sampling is a special case of **MCMC sampling.**
  Roughly, MCMC means to invent a sampling process, where the next sample may stochastically depend on the previous (Markov property), *such that* the final sample set is guaranteed to correspond to $P(X_{1:n})$.
  $\rightarrow$ *An Introduction to MCMC for Machine Learning*

Sampling – conclusions

- Sampling algorithms are very simple, very general and very popular
  - they equally work for continuous & discrete RVs
  - one only needs to ensure/implement the ability to sample from conditional distributions, no further algebraic manipulations
  - MCMC theory can reduce required number of samples

- In many cases exact and more efficient approximate inference is possible by actually computing/manipulating whole distributions in the algorithms instead of only samples.

What we didn't cover

- A very promising line of research is solving inference problems using mathematical programming. This unifies research in the areas of optimization, mathematical programming and probabilistic inference.

  Linear Programming relaxations of MAP inference and CCCP methods are great examples.