(https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-variables-models#cheatsheet)Variables-based models with CSP and Bayesian networks

By Afshine Amidi (https://twitter.com/afshinea) and Shervine Amidi (https://twitter.com/shervinea)

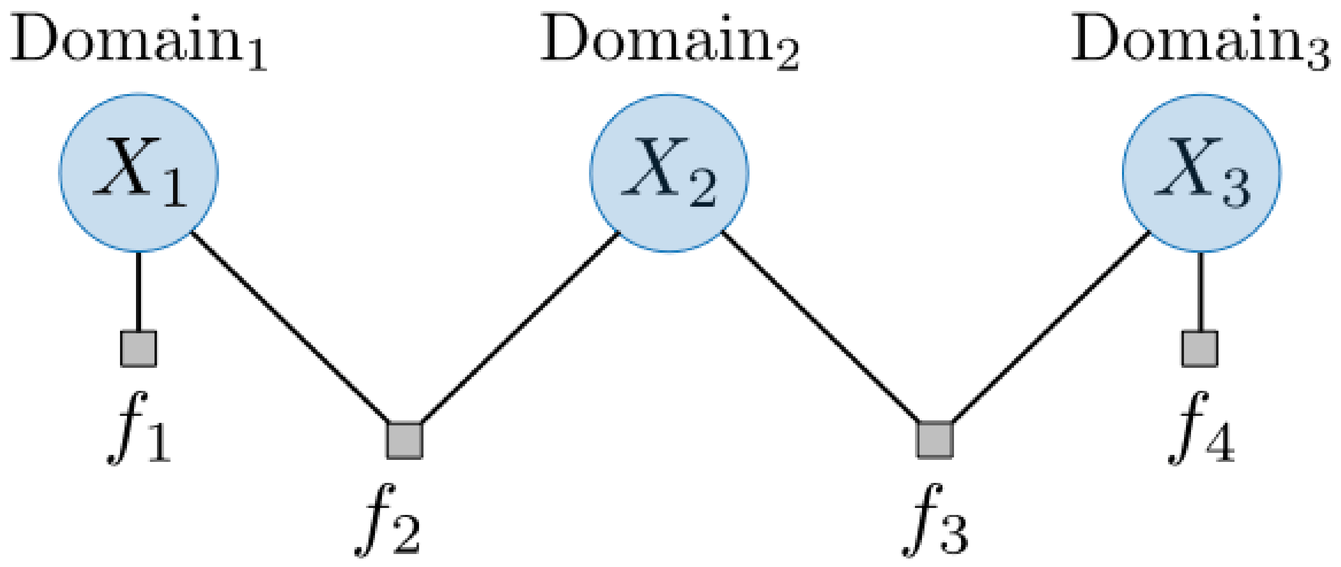
Star 2,698

Constraint satisfaction problems

In this section, our objective is to find maximum weight assignments of variable-based models. One advantage compared to states-based models is that these algorithms are more convenient to encode problem-specific constraints.

https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-variables-nodels#factor-graphs) Factor graphs

❐ **Definition** — A factor graph, also referred to as a Markov random field, is a set of variables $X=(X_1,...,X_n)$ where $X_i\in\textrm{Domain}_i$ and m factors $f_1,...,f_m$ with each $f_j(X)\geqslant0$.



❒ **Scope and arity** — The scope of a factor f_j is the set of variables it depends on. The size of this set is called the arity.

Remark: factors of arity 1 and 2 are called unary and binary respectively.

❒ **Assignment weight** — Each assignment $x=(x_1,...,x_n)$ yields a weight $\text{Weight}(x)$ defined as being the product of all factors f_j applied to that assignment. Its expression is given by:

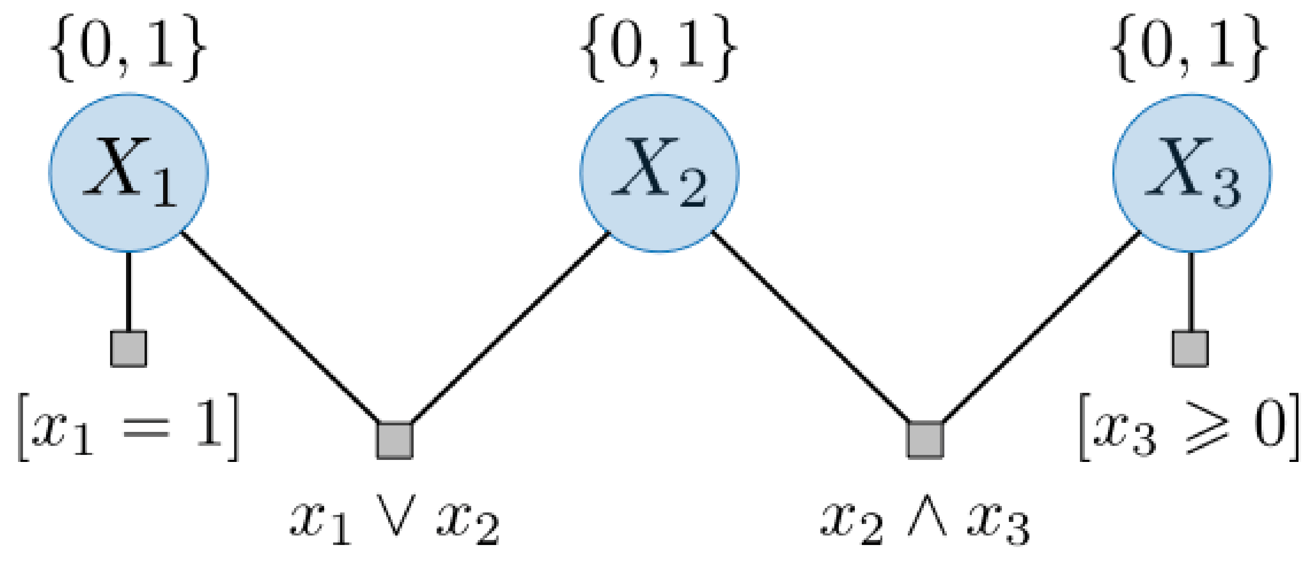
$$\boxed{\text{Weight}(x)=\prod_{j=1}^{m}f_j(x)}$$

❒ **Constraint satisfaction problem** — A constraint satisfaction problem (CSP) is a factor graph where all factors are binary; we call them to be constraints:

$$\boxed{\forall j\in[\![1,m]\!],\quad f_j(x)\in\{0,1\}}$$

Here, the constraint j with assignment x is said to be satisfied if and only if $f_j(x)=1$.

❒ **Consistent assignment** — An assignment x of a CSP is said to be consistent if and only if $\text{Weight}(x)=1$, i.e. all constraints are satisfied.



https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-variables-nodels#dynamic-ordering) Dynamic ordering

❒ **Dependent factors** — The set of dependent factors of variable X_i with partial assignment x is called $D(x, X_i)$, and denotes the set of factors that link X_i to already assigned variables.

❒ **Backtracking search** — Backtracking search is an algorithm used to find maximum weight assignments of a factor graph. At each step, it chooses an unassigned variable and explores its values by recursion. Dynamic ordering (*i.e.* choice of variables and values) and lookahead (*i.e.* early elimination of inconsistent options) can be used to explore the graph more efficiently, although the worst-case runtime stays exponential: $O(|\textrm{Domain}|^n)$.

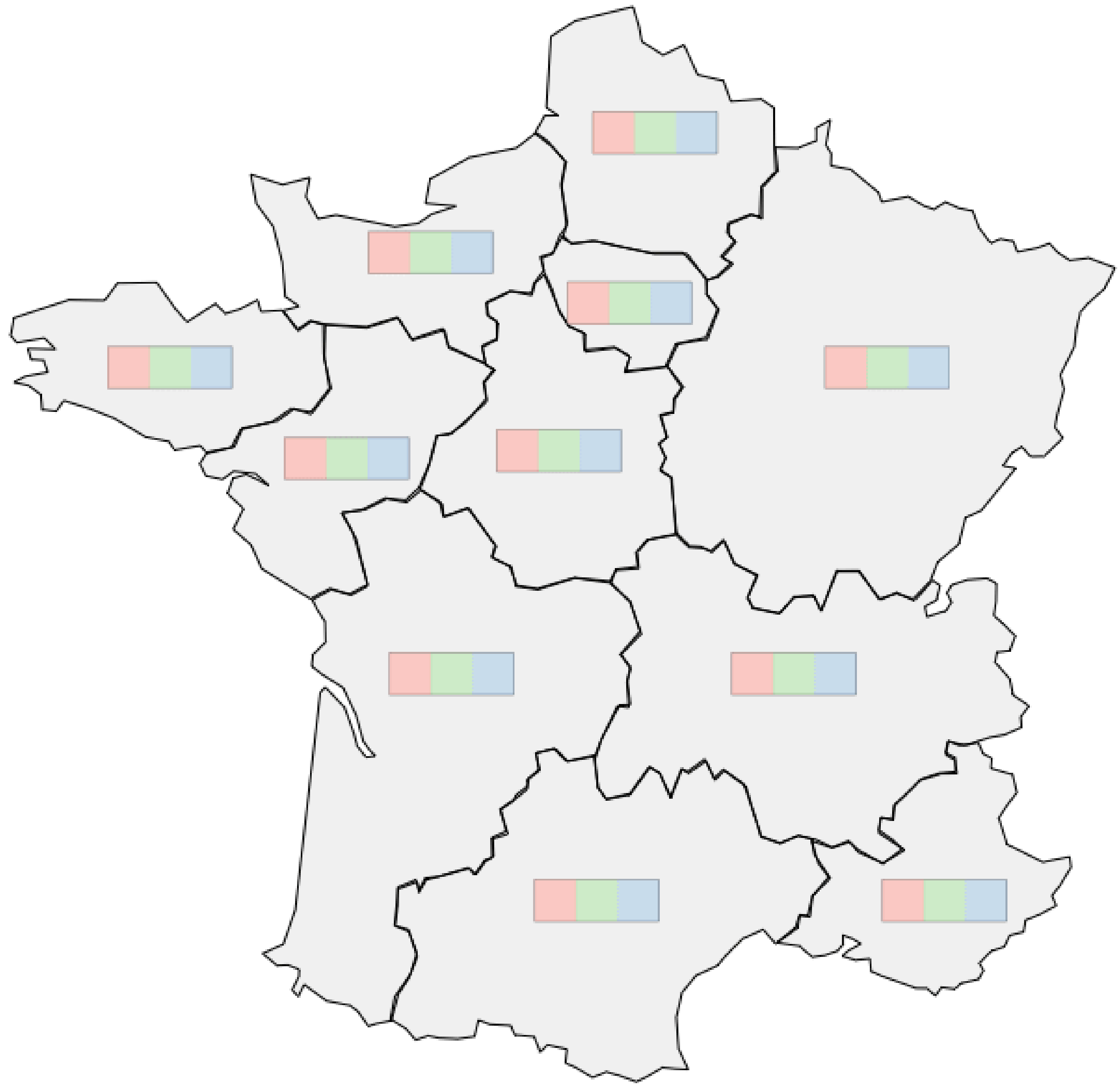
❒ **Forward checking** — It is a one-step lookahead heuristic that preemptively removes inconsistent values from the domains of neighboring variables. It has the following characteristics:

- After assigning a variable X_i, it eliminates inconsistent values from the domains of all its neighbors.
- If any of these domains becomes empty, we stop the local backtracking search.
- If we un-assign a variable X_i, we have to restore the domain of its neighbors.

❒ **Most constrained variable** — It is a variable-level ordering heuristic that selects the next unassigned variable that has the fewest consistent values. This has the effect of making inconsistent assignments to fail earlier in the search, which enables more efficient pruning.

❐ **Least constrained value** — It is a value-level ordering heuristic that assigns the next value that yields the highest number of consistent values of neighboring variables. Intuitively, this procedure chooses first the values that are most likely to work.

Remark: in practice, this heuristic is useful when all factors are constraints.



The example above is an illustration of the 3-color problem with backtracking search coupled with most constrained variable exploration and least constrained value heuristic, as well as forward checking at each step.

❐ **Arc consistency** — We say that arc consistency of variable X_l with respect to X_k is enforced when for each $x_l \in \textrm{Domain}_l$:

- unary factors of X_l are non-zero,

- there exists at least one $x_k \in \mathrm{Domain}_k$ such that any factor between X_l and X_k is non-zero.

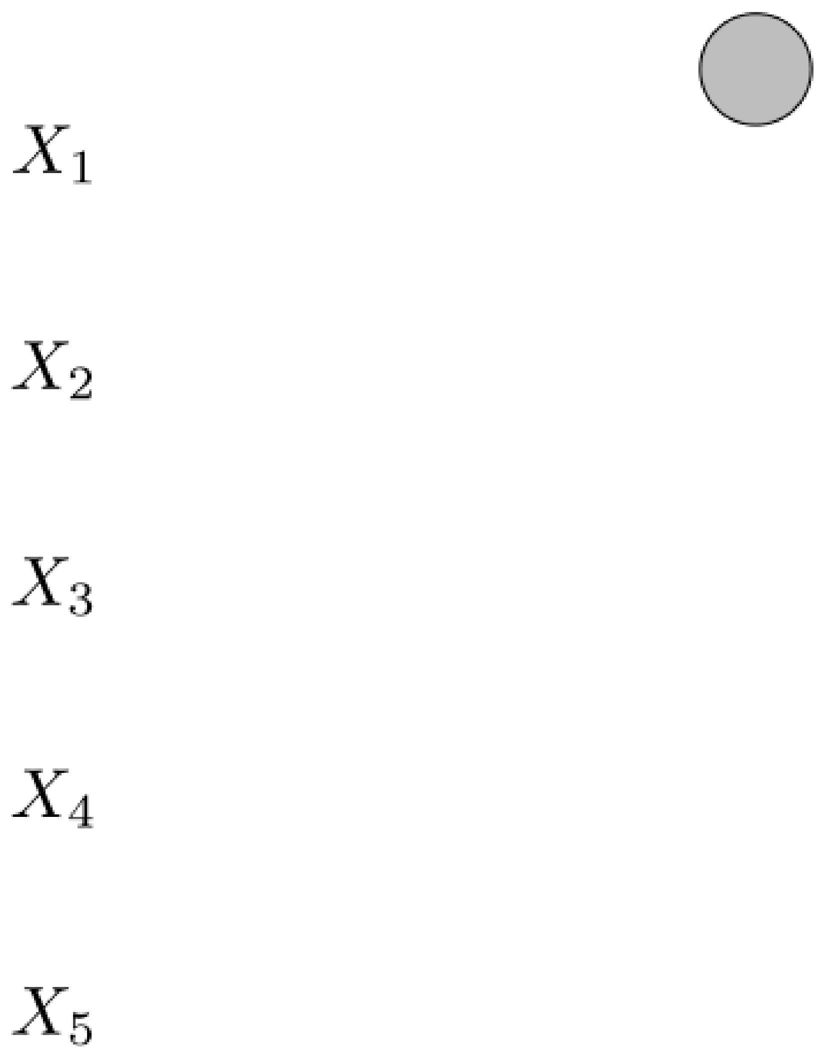
❒ **AC-3** — The AC-3 algorithm is a multi-step lookahead heuristic that applies forward checking to all relevant variables. After a given assignment, it performs forward checking and then successively enforces arc consistency with respect to the neighbors of variables for which the domain change during the process.

Remark: AC-3 can be implemented both iteratively and recursively.

(https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-variables-nodels#approximate-methods)
Approximate methods

❒ **Beam search** — Beam search is an approximate algorithm that extends partial assignments of n variables of branching factor $b = |\mathrm{Domain}|$ by exploring the K top paths at each step. The beam size $K \in \{1, ..., b^n\}$ controls the tradeoff between efficiency and accuracy. This algorithm has a time complexity of $O(n \cdot Kb\log(Kb))$.

The example below illustrates a possible beam search of parameters $K = 2$, $b = 3$ and $n = 5$
.



Remark: $K=1$ corresponds to greedy search whereas $K\rightarrow+\infty$ is equivalent to BFS tree search.

❐ **Iterated conditional modes** — Iterated conditional modes (ICM) is an iterative approximate algorithm that modifies the assignment of a factor graph one variable at a time until convergence. At step i, we assign to X_i the value v that maximizes the product of all factors connected to that variable.

Remark: ICM may get stuck in local minima.

❐ **Gibbs sampling** — Gibbs sampling is an iterative approximate method that modifies the assignment of a factor graph one variable at a time until convergence. At step i:

- we assign to each element $u\in\textrm{Domain}_i$ a weight $w(u)$ that is the product of all factors connected to that variable,
- we sample v from the probability distribution induced by w and assign it to X_i.

Remark: Gibbs sampling can be seen as the probabilistic counterpart of ICM. It has the advantage to be able to escape local minima in most cases.

(https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-variables-models#factor-graph-transformations) Factor graph transformations

❐ **Independence** — Let A, B be a partitioning of the variables X. We say that A and B are independent if there are no edges between A and B and we write:

$$\boxed{A \perp\!\!\!\perp B}$$

Remark: independence is the key property that allows us to solve subproblems in parallel.

❐ **Conditional independence** — We say that A and B are conditionally independent given C if conditioning on C produces a graph in which A and B are independent. In this case, it is written:

$$\boxed{A \perp\!\!\!\perp B | C}$$

❐ **Conditioning** — Conditioning is a transformation aiming at making variables independent that breaks up a factor graph into smaller pieces that can be solved in parallel and can use backtracking. In order to condition on a variable $X_i = v$, we do as follows:

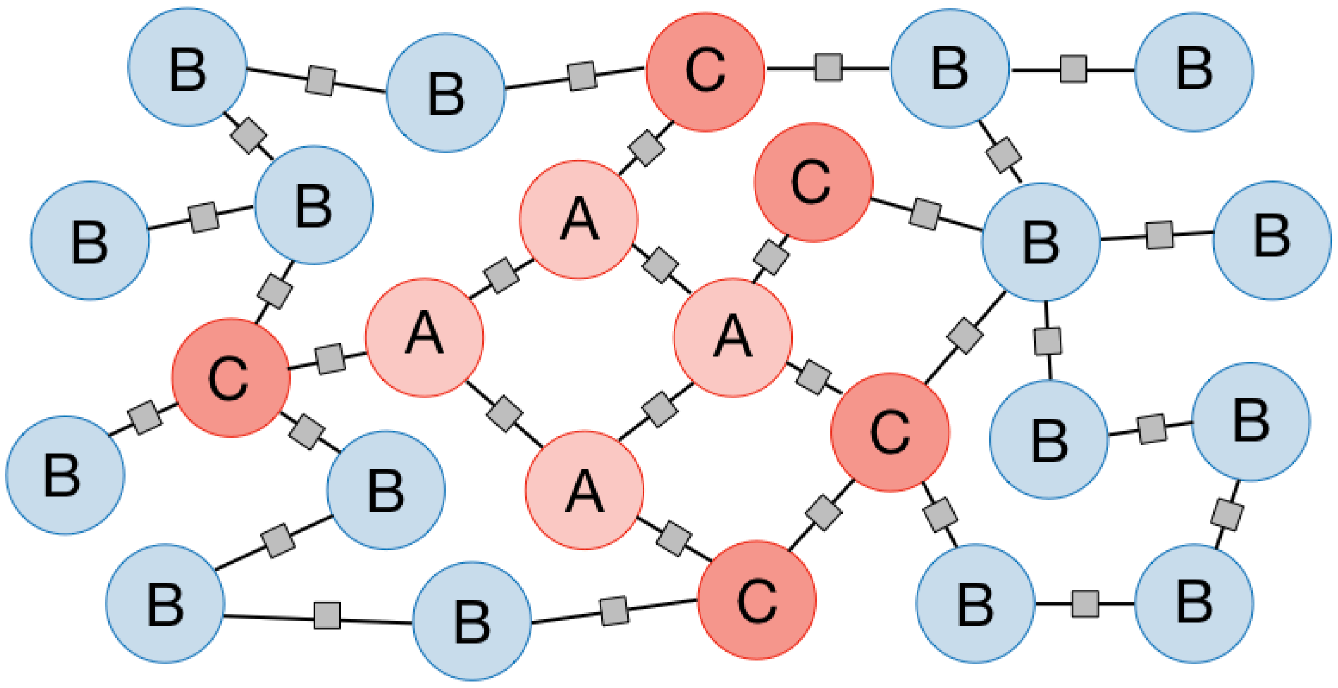
- Consider all factors $f_1, ..., f_k$ that depend on X_i
- Remove X_i and $f_1, ..., f_k$
- Add $g_j(x)$ for $j \in \{1, ..., k\}$ defined as:

$$\boxed{g_j(x) = f_j(x \cup \{X_i : v\})}$$

❐ **Markov blanket** — Let $A \subseteq X$ be a subset of variables. We define $\textrm{MarkovBlanket}(A)$ to be the neighbors of A that are not in A.

❐ **Proposition** — Let $C = \textrm{MarkovBlanket}(A)$ and $B = X \backslash (A \cup C)$. Then we have:

$$\boxed{A \perp\!\!\!\perp B | C}$$



❐ **Elimination** — Elimination is a factor graph transformation that removes X_i from the graph and solves a small subproblem conditioned on its Markov blanket as follows:

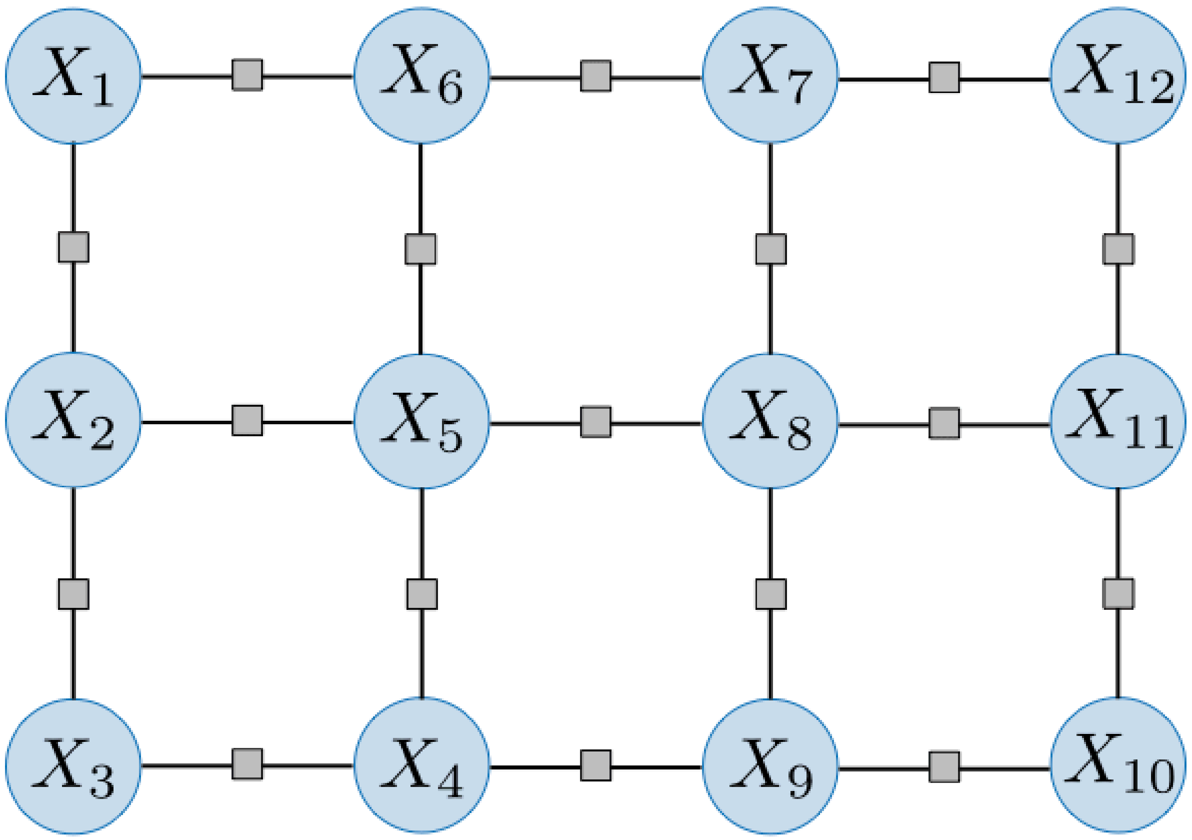
- Consider all factors $f_{i,1}, ..., f_{i,k}$ that depend on X_i
- Remove X_i and $f_{i,1}, ..., f_{i,k}$
- Add $f_{\text{new},i}(x)$ defined as:

$$\boxed{f_{\text{new},i}(x) = \max_{x_i} \prod_{l=1}^{k} f_{i,l}(x)}$$

❐ **Treewidth** — The treewidth of a factor graph is the maximum arity of any factor created by variable elimination with the best variable ordering. In other words,

$$\boxed{\text{Treewidth} = \min_{\text{orderings}} \max_{i\in\{1,...,n\}} \text{arity}(f_{\text{new},i})}$$

The example below illustrates the case of a factor graph of treewidth 3.



Remark: finding the best variable ordering is a NP-hard problem.

(https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-variables-models#bayesian-networks) Bayesian networks

In this section, our goal will be to compute conditional probabilities. What is the probability of a query given evidence?

Introduction

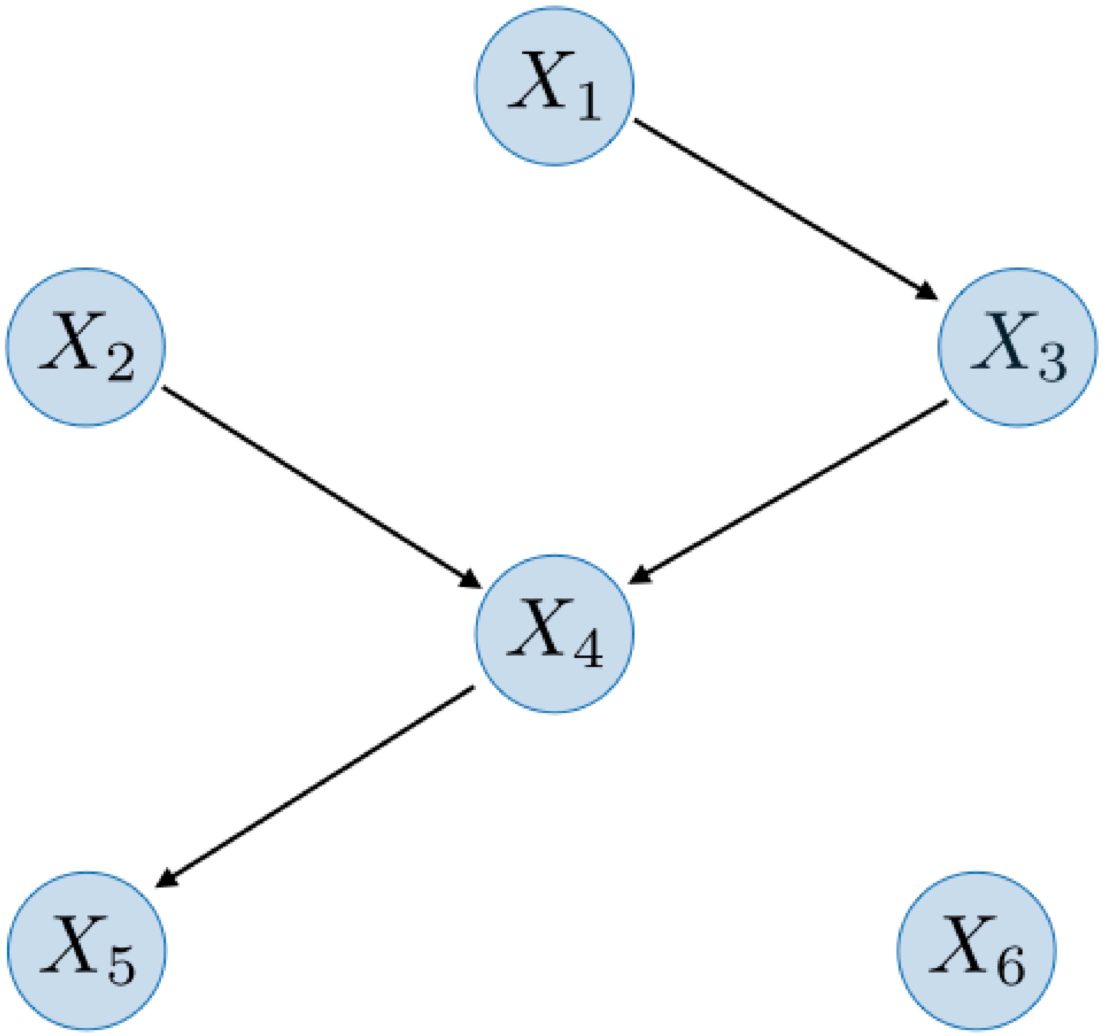
❒ **Explaining away** — Suppose causes C_1 and C_2 influence an effect E. Conditioning on the effect E and on one of the causes (say C_1) changes the probability of the other cause (say C_2). In this case, we say that C_1 has explained away C_2.

❒ **Directed acyclic graph** — A directed acyclic graph (DAG) is a finite directed graph with no directed cycles.

❒ **Bayesian network** — A Bayesian network is a directed acyclic graph (DAG) that specifies a joint distribution over random variables $X = (X_1, ..., X_n)$ as a product of local conditional distributions, one for each node:

$$\boxed{P(X_1=x_1,...,X_n=x_n)\triangleq\prod_{i=1}^{n}p(x_i|x_{\textrm{Parents}(i)})}$$

Remark: Bayesian networks are factor graphs imbued with the language of probability.



❐ **Locally normalized** — For each $x_{\textrm{Parents}(i)}$, all factors are local conditional distributions. Hence they have to satisfy:

$$\boxed{\sum_{x_i}p(x_i|x_{\textrm{Parents}(i)})=1}$$

As a result, sub-Bayesian networks and conditional distributions are consistent.

Remark: local conditional distributions are the true conditional distributions.

❐ **Marginalization** — The marginalization of a leaf node yields a Bayesian network without that node.

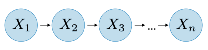
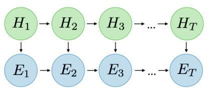
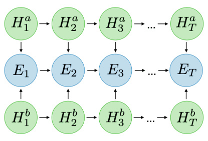
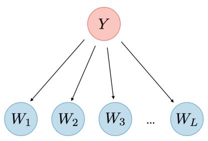
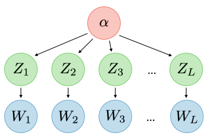
https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-variables-odels#probabilistic-program)

Probabilistic programs

❐ **Concept** — A probabilistic program randomizes variables assignment. That way, we can write down complex Bayesian networks that generate assignments without us having to explicitly specify associated probabilities.

Remark: examples of probabilistic programs include Hidden Markov model (HMM), factorial HMM, naive Bayes, latent Dirichlet allocation, diseases and symptoms and stochastic block models.

❐ **Summary** — The table below summarizes the common probabilistic programs as well as their applications:

Program	**Algorithm**	**Illustration**	**Example**
Markov Model	Generate $X_i \sim p(X_i\|X_{i-1})$		Language modeling
Hidden Markov Model (HMM)	Generate $H_t \sim p(H_t\|H_{t-1})$ Generate $E_t \sim p(E_t\|H_t)$		Object tracking
Factorial HMM	Generate H_t^o $\sim$ $o \in \{a,b\}$ $p(H_t^o\|H_{t-1}^o)$ Generate $E_t \sim p(E_t\|H_t^a, H_t^b)$		Multiple object tracking
Naive Bayes	Generate $Y \sim p(Y)$ Generate $W_i \sim p(W_i\|Y)$		Document classification
Latent Dirichlet Allocation (LDA)	Generate $\alpha \in \mathbb{R}^K$ distribution Generate $Z_i \sim p(Z_i\|\alpha)$ Generate $W_i \sim p(W_i\|Z_i)$		Topic modeling

(https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-variables-models#inference)

Inference

❐ **General probabilistic inference strategy** — The strategy to compute the probability $P(Q|E=e)$ of query Q given evidence $E=e$ is as follows:

- Step 1: Remove variables that are not ancestors of the query Q or the evidence E by marginalization
- Step 2: Convert Bayesian network to factor graph
- Step 3: Condition on the evidence $E=e$
- Step 4: Remove nodes disconnected from the query Q by marginalization
- Step 5: Run a probabilistic inference algorithm (manual, variable elimination, Gibbs sampling, particle filtering)

❐ **Forward-backward algorithm** — This algorithm computes the exact value of $P(H=h_k|E=e)$ (smoothing query) for any $k\in\{1,...,L\}$ in the case of an HMM of size L. To do so, we proceed in 3 steps:

- Step 1: for $i\in\{1,...,L\}$, compute $F_i(h_i)=\sum_{h_{i-1}}F_{i-1}(h_{i-1})p(h_i|h_{i-1})p(e_i|h_i)$
- Step 2: for $i\in\{L,...,1\}$, compute $B_i(h_i)=\sum_{h_{i+1}}B_{i+1}(h_{i+1})p(h_{i+1}|h_i)p(e_{i+1}|h_{i+1})$
- Step 3: for $i\in\{1,...,L\}$, compute $S_i(h_i)=\frac{F_i(h_i)B_i(h_i)}{\sum_{h_i}F_i(h_i)B_i(h_i)}$

with the convention $F_0=B_{L+1}=1$. From this procedure and these notations, we get that

$$\boxed{P(H=h_k|E=e)=S_k(h_k)}$$

Remark: this algorithm interprets each assignment to be a path where each edge $h_{i-1}\rightarrow h_i$ is of weight $p(h_i|h_{i-1})p(e_i|h_i)$.

❐ **Gibbs sampling** — This algorithm is an iterative approximate method that uses a small set of assignments (particles) to represent a large probability distribution. From a random assignment x, Gibbs sampling performs the following steps for $i\in\{1,...,n\}$ until convergence:

- For all $u\in\textrm{Domain}_i$, compute the weight $w(u)$ of assignment x where $X_i=u$
- Sample v from the probability distribution induced by w: $v\sim P(X_i=v|X_{-i}=x_{-i})$
- Set $X_i=v$

Remark: X_{-i} denotes $X\backslash\{X_i\}$ and x_{-i} represents the corresponding assignment.

❐ **Particle filtering** — This algorithm approximates the posterior density of state variables given the evidence of observation variables by keeping track of K particles at a time. Starting from a set of particles C of size K, we run the following 3 steps iteratively:

- Step 1: proposal - For each old particle $x_{t-1} \in C$, sample x from the transition probability distribution $p(x|x_{t-1})$ and add x to a set C'.
- Step 2: weighting - Weigh each x of the set C' by $w(x) = p(e_t|x)$, where e_t is the evidence observed at time t.
- Step 3: resampling - Sample K elements from the set C' using the probability distribution induced by w and store them in C: these are the current particles x_t.

Remark: a more expensive version of this algorithm also keeps track of past particles in the proposal step.

❐ **Maximum likelihood** — If we don't know the local conditional distributions, we can learn them using maximum likelihood.

$$\boxed{\max_{\theta} \prod_{x \in \mathcal{D}_{\text{train}}} p(X = x; \theta)}$$

❐ **Laplace smoothing** — For each distribution d and partial assignment $(x_{\text{Parents}(i)}, x_i)$, add λ to $\text{count}_d(x_{\text{Parents}(i)}, x_i)$, then normalize to get probability estimates.

❐ **Algorithm** — The Expectation-Maximization (EM) algorithm gives an efficient method at estimating the parameter θ through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:

- E-step: Evaluate the posterior probability $q(h)$ that each data point e came from a particular cluster h as follows:

$$\boxed{q(h) = P(H = h|E = e; \theta)}$$

- M-step: Use the posterior probabilities $q(h)$ as cluster specific weights on data points e to determine θ through maximum likelihood.