

# ECE 590.17: Lecture 2 – Introduction to Factor Graphs

Graphical Models and Inference for Machine Learning

Duke University, Spring 2026

**History:** Written by Henry Pfister (2026).

**Last Modified:** 02/13/2026

## Outline of lecture:

2.1	Factor graphs . . . . .	1
2.2	Examples: Markov chains, Sudoku, K-SAT, and ECCs . . . . .	2
2.2.1	Markov chain . . . . .	2
2.2.2	Factor Graphs and Bayesian Networks . . . . .	3
2.2.3	Sudoku as a factor graph . . . . .	4
2.2.4	K-SAT . . . . .	5
2.2.5	Error-correcting codes . . . . .	5
2.3	From factorization to inference tasks . . . . .	6
2.4	Conditional independence structure . . . . .	6
2.5	Belief propagation for discrete variables . . . . .	7
2.6	Trees and beyond . . . . .	9
2.7	Peeling for Sudoku, K-SAT, and codes . . . . .	9
2.8	Connections and roadmap . . . . .	10
2.9	Worked examples . . . . .	11

## 2.1 Factor graphs

A *factor graph* is a bipartite graph that represents how a global function  $f: \mathcal{X} \rightarrow \mathbb{R}$  factors into local terms. Let  $\mathcal{X}_0$  be a finite set and  $x = (x_1, \dots, x_n) \in \mathcal{X}^n$  be a vector over this alphabet. Consider a nonnegative function  $f: \mathcal{X}^n \rightarrow \mathbb{R}$ . A factor graph is a bipartite graph with “variable nodes” indexed by  $[n]$  and “factor nodes” indexed by a finite set  $F$ . For each factor node  $a \in F$ , we use  $\partial a$  to denote the subset of variable nodes attached to it. Using this, one can express  $f$  as

$$f(x) = \prod_{a \in F} f_a(x_{\partial a}),$$

where  $x_{\partial a} \in \mathcal{X}^{|\partial a|}$  denotes the subvector of  $x$  with elements indexed by  $\partial a$ . The graph has edges, for each  $a \in F$ , between the factor node  $a$  to the variable nodes in  $\partial a$ . This representation makes it transparent which variables interact locally, and it is the foundation for message-passing algorithms.

A common variation is that of a *pairwise graphical model* where

$$f(x) = \left( \prod_{i=1}^n f_i(x_i) \right) \left( \prod_{(i,j) \in E} f_{ij}(x_i, x_j) \right),$$

where the  $f_i$  are unary factors and the  $f_{ij}$  are pairwise compatibility factors. In this case, factors affect at most two variable nodes and can be represented by edges in the graph rather than separate factor nodes. Thus, the vertices are indexed by  $[n]$  and the edges  $E \subseteq \binom{[n]}{2}$  index the active pairwise

factors. The unary factors are typically neglected in the graph but are sometimes represented by “half-edges” that attach only to a variable node.

In probabilistic modeling we often work with distributions of the form

$$\mu(x) = \frac{1}{Z} f(x), \quad Z = \sum_{x \in \mathcal{X}^n} f(x).$$

The normalization constant  $Z$  is called the *partition function*. Computing marginals and modes can be difficult in high dimensions, but when  $f$  factors, the factor graph provides a scaffold for efficient inference on trees and useful approximations on loopy graphs.

**Why factor graphs?** They unify problems from machine learning, constraint satisfaction, statistical physics, and error-correcting codes (ECCs). The same local structure that yields sparse constraints also yields scalable algorithms. In later lectures we will connect factor graphs to conditional independence, complexity, and learning.

**Elimination and complexity.** We denote the marginal distribution of  $x_i$  by

$$\mu_i(x') = \frac{1}{Z} \sum_{x \in \mathcal{X}^n} f(x) \delta_{x_i, x'}.$$

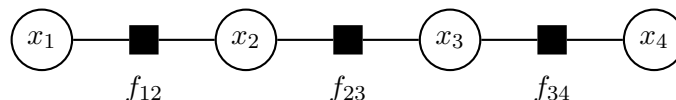
For each  $x' \in \mathcal{X}$ , one must sum  $|\mathcal{X}|^{n-1}$  configurations giving a naive complexity of  $|\mathcal{X}|^n$ . But, in a factor graph, we can eliminate variables one at a time by grouping terms that depend on the eliminated variable, performing a local sum, and introducing a new factor. The size of the largest intermediate factor controls the complexity. On a tree, all intermediate factors remain small and exact inference is linear in the number of edges.

**Partition function as a counting problem.** When  $f$  is the indicator function for satisfying all constraints (e.g., K-SAT, Sudoku, etc...),  $Z$  equals the number of valid solutions. In statistical physics,  $\log Z$  encodes important information about the system and is called the *free entropy*. From the perspective of complexity theory, computing  $Z$  is harder than finding a single solution. Thus, one often focuses on approximate inference and sampling.

## 2.2 Examples: Markov chains, Sudoku, K-SAT, and ECCs

Factor graphs can be used to model many things. The Markov chain example shows how they model well-known objects in probability. Another common case is systems with hard constraints. See the examples below that can be written as a product of indicator factors that enforce local constraints.

### 2.2.1 Markov chain



Consider a Markov chain  $X_1 - X_2 - X_3 - X_4$  where each random variable takes values in the finite alphabet  $\mathcal{X}$ . Its joint distribution factors as

$$p(x_1, x_2, x_3, x_4) = p(x_1, x_2) p(x_3 | x_2) p(x_4 | x_3)$$

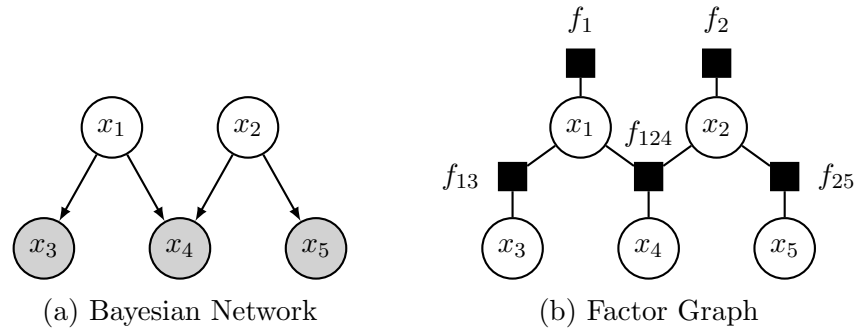


Figure 2.1: Bayesian network (BN) versus factor graph (FG) for small tree.

and this can be represented by a factor graph with four variable nodes and three pairwise factors. For example, we can define

$$f_{12}(x_1, x_2) = p(x_1, x_2), \quad f_{23}(x_2, x_3) = p(x_3 | x_2), \quad f_{34}(x_3, x_4) = p(x_4 | x_3)$$

so that  $p(x_1, x_2, x_3, x_4) = f_{12}(x_1, x_2)f_{23}(x_2, x_3)f_{34}(x_3, x_4)$  via the factor graph.

## 2.2.2 Factor Graphs and Bayesian Networks

A Bayesian network is a directed acyclic graph (DAG) where, for a vertex  $X_i$ , we define the set of *descendants*,  $X_{>i}$ , to be all vertices reachable along directed paths from  $i$ . Similarly, we define the set of *ancestors*,  $X_{<i}$ , to be all vertices with a directed path to  $i$ . The *parents* of  $i$  are the vertices that have a directed edge to  $i$ . Observed variables in a BN are sometimes shaded to indicate that fact.

In a Bayesian network (BN), basic conditional independence is defined by the fact that

$$p_{X_i | X_{<i}}(x_i | x_{<i}) = p_{X_i | X_{\text{pa}(i)}}(x_i | x_{\text{pa}(i)}),$$

which means that all ancestors except the parents are conditionally independent of the child  $X_i$  given the values of the parents  $X_{\text{pa}(i)}$ . This is a kind of causality where the chain rule can be applied by conditioning each variable on its parents.

$$p_{X_1, \dots, X_n}(x_1, \dots, x_n) = \prod_{i=1}^n p_{X_i | X_{\text{pa}(i)}}(x_i | x_{\text{pa}(i)}).$$

A permutation  $\pi: [n] \rightarrow [n]$  defines an ordering of the variables and is called *ancestral* if all parents appear before their child. For any ancestral ordering  $\pi$ , the chain rule decomposition allows one to efficiently draw samples from a BN by sampling  $X_{\pi(i)}$  for  $i = 1, 2, \dots$ . This holds because all unsampled factors marginalize via

$$\sum_{x_i \in \mathcal{X}} p_{X_i | X_{\text{pa}(i)}}(x_i | x_{\text{pa}(i)}) = 1.$$

Figure 2.1 shows a Bayesian network and a factor graph for the same simple tree model. In this case, the BN represents

$$p_{X_1, \dots, X_5}(x_1, \dots, x_5) = P_{X_1}(x_1)P_{X_2}(x_2)P_{X_3|X_1}(x_3 | x_1)p_{X_4|X_1, X_2}(x_4 | x_1, x_2)p_{X_5|X_2}(x_5 | x_2).$$

Likewise, we observe that the factor graph represents the function

$$f(x_1, \dots, x_5) = f_1(x_1)f_2(x_2)f_{13}(x_1, x_3)f_{124}(x_1, x_2, x_4)f_{25}(x_2, x_5).$$

If the factor nodes are chosen to equal the corresponding conditional probabilities, then the two models are identical.

A BN where variable  $i$  has  $d_i - 1$  parents can always be converted into a factor graph as follows. For each variable node (denoted by  $i$ ), introduce a factor node of degree  $d_i$  to represent the conditional probability and connect it to variable node  $i$  and all of its parents. The resulting factor graph may not be a tree but one can still draw samples using any ancestral ordering of the variables. Likewise, a tree factor graph can be always converted into a BN by choosing a root node and converting each factor into directed edges towards the root from each of the variables farther from the root. It is not known how to do this for arbitrary factor graphs with cycles, without blowing up the degrees. If it were, one could solve NP-complete problems such as  $K$ -SAT in polynomial time and collapse the polynomial hierarchy.

### 2.2.3 Sudoku as a factor graph

A standard  $9 \times 9$  Sudoku has 81 variables, each taking values in  $\{1, \dots, 9\}$ . We can index the Sudoku cells using a map  $\phi : [9]^2 \rightarrow [81]$  with

$$\phi(i, j) = 9(i - 1) + j,$$

so that cell  $(i, j)$  corresponds to variable  $x_{\phi(i, j)}$ . This lets us write constraints in terms of a single index set. For example, the  $i$ th row constraint involves variables  $\{x_{\phi(i, 1)}, \dots, x_{\phi(i, 9)}\}$ , the  $j$ th column constraint involves  $\{x_{\phi(1, j)}, \dots, x_{\phi(9, j)}\}$ , and the  $(r, s)$  block constraint (with  $r, s \in \{0, 1, 2\}$ ) involves  $\{x_{\phi(3r+u, 3s+v)} : u, v \in \{1, 2, 3\}\}$ .

The constraints are of three types: each row, each column, and each  $3 \times 3$  subgrid must contain all digits exactly once. Each constraint can be encoded by a factor that evaluates to 1 if its local assignment is valid and 0 otherwise. The global function is the product of these factors, and any satisfying assignment corresponds to a solution. This shows how a combinatorial puzzle becomes a structured inference problem.

We note that the local constraint itself can be represented in many ways. For example, requiring each element to appear exactly once is the same as requiring all distinct positions of the local constraint to take different values. An alternative approach is to use binary indicator functions. In the binary indicator representation, introduce variables  $z_{i, j, d} \in \{0, 1\}$  that indicate whether cell  $(i, j)$  takes the value  $d \in [9]$ . The one-hot indicator constraints are

$$\sum_{d=1}^9 z_{i, j, d} = 1 \quad \text{for each cell } (i, j),$$

and the row/column/block constraints become

$$\sum_{j=1}^9 z_{i, j, d} = 1 \quad \text{for each row } i \text{ and digit } d,$$

with analogous constraints for columns and blocks. Each of these constraints can be encoded as a factor. The resulting graph has more variables but the factor nodes have fewer valid patterns, which can reduce the complexity of belief-propagation.

Different representations trade off graph size and factor complexity and provide a concrete example of modeling choices that affect inference algorithms. Also, the choice of representation can

significantly change the behavior of message passing. In the first representation, each factor node has degree 9 with  $9!$  valid patterns out of  $9^9$  possibilities. In the second, the number of factors and variables both grow by a factor of 9 but the factors themselves only have 9 valid patterns out of  $2^9$  possibilities.

### 2.2.4 K-SAT

In  $K$ -SAT, variables  $x_i \in \{0, 1\}$  must satisfy a collection of clauses, where each clause is an OR of  $K$  literals (variables or their negations). Define a factor  $\psi_a(x_{\partial a})$  that equals 1 if clause  $a$  is satisfied and 0 otherwise. The product of all clause factors encodes the formula. Satisfiability corresponds to whether the partition function is nonzero, while optimization variants (MAX-SAT) maximize the number of satisfied clauses. K-SAT is a key example for complexity and phase transitions.

For example, a 3-SAT formula on 4 variables is given by

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4),$$

where the three clauses are enforced by the factors  $f_A, f_B, f_C$ . Each factor evaluates to 1 unless the inputs match the single falsifying assignment. For instance,

$$f_A(x_1, x_2, x_3) = 1 - (1 - x_1)x_2(1 - x_3),$$

and similarly for the other clauses.

$K$ -SAT is often studied in a random ensemble where  $M = \alpha N$  clauses are chosen uniformly at random. As  $\alpha$  increases, instances undergo a sharp transition from satisfiable to unsatisfiable with high probability. This “phase transition” is a guiding example that shows how typical-case complexity can differ from worst-case complexity.

In complexity theory, a decision problem takes an input string (typically binary) and returns a yes/no answer. Consider the set of decision problems solvable by a Turing machine in time that grows only polynomially in the input length. This set is called P for polynomial. The class NP, which stands for non-deterministic polynomial, is the set of decision problems whose answers can be verified by a Turing machine in time growing polynomially in the input length. Most experts believe that  $P \neq NP$  but the question remains open. A problem is NP complete if any problem in NP can be reduced to it. A problem is NP hard if a polynomial-time solution for that problem provides a polynomial-time solution for all problems in NP.

The decision version of  $K$ -SAT asks whether a satisfying assignment exists. It turns out that  $K$ -SAT is NP-complete for  $K \geq 3$ . The search version returns a single satisfying assignment if one exists. This is NP hard because it naturally solves the decision version. The variation known as MAX-SAT computes the maximum number of satisfied clauses for any assignment and a MAX-SAT solver tries to find an assignment that achieves the maximum. Both are NP hard for  $K \geq 2$ .

### 2.2.5 Error-correcting codes

Linear block codes can be written as factor graphs with parity-check constraints. For a binary code, each parity check enforces that the XOR of a subset of bits is zero. Given a received word  $y$  and a channel likelihood, the posterior distribution over codewords factors into a product of local checks and symbol likelihoods. This representation is central for belief propagation decoding and connects naturally to the graphical models view of inference.

For a linear code, the parity-check matrix  $H$  defines which variables participate in each constraint. Low-density parity-check (LDPC) codes correspond to sparse  $H$  and hence sparse factor graphs. The sparsity enables iterative decoding methods that are practical for large block lengths.

Two decoding tasks mirror the earlier inference tasks: symbol marginals  $\mu_i(x_i)$  support soft decisions and error-rate analysis, while the MAP codeword maximizes  $\mu(x)$ . On tree-like graphs the two can be computed efficiently, but realistic codes use loopy graphs where BP becomes an approximation with excellent empirical performance.

## 2.3 From factorization to inference tasks

Once a problem is written as a product of local factors, two core tasks emerge: marginalization and optimization.

- **Marginalization:** compute  $\mu_i(x')$  for one or more variables. This supports probabilistic decisions and uncertainty quantification.
- **Optimization (MAP):** find  $x$  that maximizes  $f(x)$ , which corresponds to minimizing an energy  $E(x)$  if  $f(x) = e^{-E(x)}$ .

The cost of naive marginalization scales like  $|\mathcal{X}|^n$ , which quickly becomes infeasible. Factorization provides a path to reduce this cost by reordering sums, grouping local terms, and eliminating variables in a graph-structured way. These ideas will reappear under multiple names: variable elimination, junction trees, and belief propagation.

**A simple marginalization example.** Suppose

$$f(x_1, x_2, x_3, x_4) = f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_3, x_4).$$

Then

$$\mu_1(x_1) \propto \sum_{x_2} f_1(x_1, x_2) \left[ \sum_{x_3} f_2(x_2, x_3) \left( \sum_{x_4} f_3(x_3, x_4) \right) \right].$$

The inner sums can be computed once and reused, reducing the total work from  $|\mathcal{X}|^4$  to roughly  $5|\mathcal{X}|^2$ . This is the algebraic heart of message passing.

## 2.4 Conditional independence structure

Conditional independence describes when information about one set of variables does not change the distribution of another set, once some conditioning variables are known. Formally,  $X \perp\!\!\!\perp Y \mid Z$  means, for all  $z$  such that  $p(z) > 0$ , we have

$$p(x, y \mid z) = p(x \mid z)p(y \mid z).$$

Graphical models encode conditional independence structure. Factor graphs give a fine-grained view: variables interact only through shared factors, so edges capture direct dependencies. This structure is crucial for scalable inference and for understanding when message passing is exact.

**Separation in factor graphs.** Let  $A, B, S$  be disjoint sets of variable nodes. If every path in the factor graph from a node in  $A$  to a node in  $B$  passes through a node in  $S$ , then  $X_A \perp\!\!\!\perp X_B \mid X_S$ . This is the natural extension of graph separation to factor graphs and can be proved by marginalizing all variables except  $X_A$  and  $X_B$  in the separated components.

**Application: Markov chain.** In the chain example  $X_1 - X_2 - X_3 - X_4$ , all paths from  $X_1$  to  $X_4$  pass through  $\{X_2, X_3\}$ . Hence  $X_1 \perp\!\!\!\perp X_4 \mid (X_2, X_3)$ . In particular,  $X_1 \perp\!\!\!\perp X_3 \mid X_2$ , which is the standard Markov property.

**Locality and sparsity.** Most problems of interest have sparse interactions. Conditional independence allows one to replace exponential computations with local ones, often converting sums over  $\mathcal{X}^n$  into a sequence of summations over smaller sets.

**Markov random fields.** A Markov random field (MRF) is an undirected graphical model that represents a joint distribution as a product of clique potentials

$$p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C),$$

where  $\mathcal{C}$  are the cliques of the graph. A MRF can be converted into a factor graph by creating one factor per clique. In Markov random fields, separation in the graph corresponds to conditional independence.

**Markov blanket.** For a set  $A$  of variables in a graphical model, the *Markov blanket* is the minimal set  $S \subseteq [n] \setminus A$  of variables such that  $X_A \perp\!\!\!\perp X_B \mid X_S$  for  $X_B = [n] \setminus A \cup S$ . For a factor graph and a set  $A$  of variables, let  $F$  be the set of adjacent factors. The Markov blanket of  $A$  is the set of neighbors of  $F$  that are not in  $A$ . For a Markov random field, the Markov blanket of  $A$  is simply its set of neighbors not in  $A$ .

**Conditional independence in Bayesian networks.** Let  $G$  be a DAG with vertices  $[n]$  and random variables  $X_1, \dots, X_n$  defining a Bayesian network. For disjoint sets  $A, B, S \subseteq [n]$ , an undirected path  $\pi$  between  $A$  and  $B$  is *blocked* by  $S$  if it contains

- a *non-collider* node  $v$ : the arrows on  $\pi$  meet head-to-tail or tail-to-tail at  $v \in S$ , or
- a *collider* node  $v$ : the arrows on  $\pi$  meet head-to-head at  $v \notin S$  with no descendant of  $v$  in  $S$ .

The sets  $A$  and  $B$  are *d-separated* by  $S$  (written  $A \perp\!\!\!\perp_G B \mid S$ ) if every path between a node in  $A$  and a node in  $B$  is blocked by  $S$ . If a distribution  $p(x)$  factorizes according to  $G$ ,

$$p(x) = \prod_{i=1}^n p(x_i \mid x_{\text{pa}(i)}),$$

then  $A \perp\!\!\!\perp_G B \mid S$  implies the conditional independence  $X_A \perp\!\!\!\perp X_B \mid X_S$  (the *global Markov property*). In particular, the variable  $X_i$  is conditionally independent of its non-descendants given its parents because all paths from  $i$  to a non-descendent are blocked by  $i$ .

See the Appendix for some formal statements and proofs related to this section.

## 2.5 Belief propagation for discrete variables

Belief propagation (BP), also known as the sum-product algorithm, is a message-passing method for computing marginals on trees. On a factor graph, messages are passed between variable nodes and factor nodes.

**Variable-to-factor messages.** For variable  $i$  and adjacent factor  $a$ :

$$m_{i \rightarrow a}(x_i) \propto \prod_{b \in F(i) \setminus a} \hat{m}_{b \rightarrow i}(x_i).$$

**Factor-to-variable messages.** For factor  $a$  and adjacent variable  $i$ :

$$\hat{m}_{a \rightarrow i}(x_i) \propto \sum_{x_{\partial a \setminus i}} f_a(x_{\partial a}) \prod_{j \in \partial a \setminus i} m_{j \rightarrow a}(x_j).$$

Given these messages, the marginal of variable  $i$  satisfies

$$\mu_i(x_i) \propto \prod_{a \in F(i)} \hat{m}_{a \rightarrow i}(x_i)$$

and the marginal belief at factor  $a$  satisfies

$$\mu_a(x_{\partial a}) \propto f_a(x_{\partial a}) \prod_{i \in \partial a} m_{i \rightarrow a}(x_i).$$

At BP fixed points, these beliefs must be locally consistent:

$$\sum_{x_{\partial a \setminus i}} \mu_a(x_{\partial a}) = \mu_i(x_i), \quad i \in \partial a.$$

On trees, BP computes exact marginals and beliefs. For loopy graphs, these marginal beliefs are approximate but often effective in practice.

**Normalization and beliefs.** Messages are usually normalized to sum to one; the resulting beliefs approximate marginals. On trees, each message summarizes all information from the corresponding subtree, which is why one forward and one backward pass suffice.

**Binary variables.** For binary variables it is convenient to work with log-likelihood ratios, which turns products into sums and can improve numerical stability. This is essential in coding applications, where messages are updated thousands of times.

**Scheduling.** BP can be run synchronously (all messages updated in parallel) or asynchronously (messages updated one at a time). On trees, the schedule does not affect correctness. On loopy graphs, schedules can affect convergence and accuracy.

**Damping and stability.** On loopy graphs, it is common to damp updates by mixing new and old messages, e.g.,  $m^{(t+1)} = (1 - \lambda)m^{(t)} + \lambda m_{\text{new}}^{(t+1)}$ . This can stabilize oscillations and improve convergence.

**Markov chain example.** For a chain of pairwise factors, BP reduces to forward-backward recursions. The forward pass computes messages from left to right; the backward pass goes right to left. The marginal at  $x_i$  combines one message from each side, which illustrates the intuition that each message summarizes all evidence on one side of the chain.

**Marginalization.** To compute the marginal of  $X_3$ , eliminate variables from left to right:

$$m_{1 \rightarrow 2}(x_2) = \sum_{x_1} f_{12}(x_1, x_2), \quad m_{2 \rightarrow 3}(x_3) = \sum_{x_2} m_{1 \rightarrow 2}(x_2) f_{23}(x_2, x_3),$$

and similarly from the right,

$$m_{4 \rightarrow 3}(x_3) = \sum_{x_4} f_{34}(x_3, x_4).$$

Combining these two messages, we find that

$$p(x_3) \propto m_{2 \rightarrow 3}(x_3) m_{4 \rightarrow 3}(x_3),$$

which is simply the *forward-backward recursion* specialized to four variables. This example illustrates how message passing replaces a global sum over  $|\mathcal{X}|^4$  assignments with a sequence of local sums.

## 2.6 Trees and beyond

When the factor graph is a tree, BP computes exact marginals and the partition function can be computed by local elimination. When cycles are present, exact inference is generally intractable, and approximate methods are needed. Later, we will see that the Bethe free entropy provides a variational interpretation of BP fixed points that connects inference to statistical physics.

**Complexity perspective.** Even when each factor is simple, global inference can be hard due to the combinatorial growth of the configuration space. This tension motivates the study of which graphical structures admit efficient exact inference and which require approximations.

**Loopy graphs.** On graphs with cycles, BP is not guaranteed to converge or to be exact, but it often provides good approximations in sparse, weakly coupled systems. Understanding when and why this happens is a recurring theme that connects to phase transitions and replica methods in physics.

**Energy-based view.** Assume all factors are strictly positive. Define local energies

$$E_a(x_{\partial a}) = -\ln f_a(x_{\partial a}),$$

and the global energy

$$E(x) = \sum_{a \in F} E_a(x_{\partial a}).$$

Then,  $f(x) = \exp[-E(x)]$  reduces exactly to the original factor form. Thus, factor graphs and energy-based models are equivalent descriptions: one emphasizes local compatibilities, the other emphasizes additive costs. The competition between energy and entropy shapes the landscape of  $E(x)$  and influences algorithmic behavior.

**Junction trees.** Exact inference on loopy graphs is possible by clustering variables into larger nodes so that the resulting graph is a tree. The cost grows exponentially with the size of these clusters, which is another view of treewidth. Junction tree methods provide a conceptual bridge between exact algorithms and BP.

## 2.7 Peeling for Sudoku, K-SAT, and codes

Peeling is a simple but powerful algorithm for sparse constraint systems. It repeatedly removes variables that participate in a single constraint (degree-one nodes), solves that constraint locally, and simplifies the remaining system. On tree-like factor graphs, peeling can solve the entire problem. On loopy graphs, it may stall, and its failure reveals the presence of a core subgraph where constraints are intertwined.

**Multiple vs unique solutions.** In K-SAT, each clause allows many satisfying assignments, and the global solution set can be large (or empty). In contrast, Sudoku puzzles are typically designed to have a unique solution, and coding problems often aim to reconstruct a unique transmitted codeword given noisy data. This distinction affects inference: SAT emphasizes existence, whereas Sudoku/coding emphasize accurate identification among competing solutions.

**Overview of general algorithm.** Maintain a queue of variables with degree one. Pop a variable, satisfy its unique constraint, remove the variable and update neighboring degrees. Repeat until no degree-one variables remain. The remaining core captures the hard part of the instance.

On a tree, the queue never empties until all variables are removed. On a loopy graph, the queue can become empty while constraints remain, which signals the onset of a core. This simple behavior makes peeling a useful diagnostic for graph structure.

**Sudoku and constraint propagation.** In Sudoku, peeling resembles constraint propagation: if a cell has only one possible value given current constraints, fix it and update nearby constraints. This process is effective for many instances and its failure indicates when additional global reasoning is needed.

**Unit clause propagation.** In SAT, if a clause has all but one literal falsified, the remaining literal must be true to satisfy the clause. Unit clause propagation iteratively applies this rule, simplifying the formula and often solving easy instances. It is a greedy analogue of peeling and it is a core subroutine in modern SAT solvers.

**K-SAT and core structure.** In random K-SAT, iterative elimination of variables appearing in a single clause uncovers a core. The emergence of a nontrivial core is linked to sharp thresholds for satisfiability and algorithmic hardness.

**Codes and erasures.** For low-density parity-check codes on erasure channels, peeling corresponds to iterative decoding. Degree-one checks reveal erased bits, which are then removed from the graph. This succeeds below a threshold and fails when a stopping set remains.

**Core size as a phase indicator.** In random graph ensembles, the emergence of a linear-size core often coincides with sharp transitions in algorithmic performance. This phenomenon will reappear when we discuss satisfiability thresholds and the limits of BP-based heuristics.

## 2.8 Connections and roadmap

These introductory examples motivate the central themes of the course:

- Factor graphs provide a common language for inference, constraints, and coding.
- Conditional independence structure enables scalable computation.
- Belief propagation is exact on trees and a powerful approximation beyond trees.
- Peeling reveals the boundary between easy and hard instances and connects algorithmic performance to graph structure.

In the next lectures we will formalize graphical model representations, study exact and approximate inference in more depth, and connect these ideas to complexity theory, statistical physics, and machine learning.

**Suggested perspective.** As you read the later chapters, keep track of three recurring objects: the factor graph, the partition function, and local messages. The first encodes structure, the second captures global difficulty, and the third is the computational mechanism by which we turn structure into inference.

**Where this leads.** The course will build from these foundations toward learning graphical models from data, designing approximate inference for large-scale systems, and connecting inference to optimization and statistical physics. We will revisit K-SAT and coding in the context of complexity theory, and revisit Ising-type models to understand phase transitions and the limits of message passing.

## 2.9 Worked examples

### Example 1: Variable elimination and intermediate factor size

Let  $x_i \in \{0, 1\}$  and consider the factorization

$$f(x_1, x_2, x_3, x_4, x_5) = f_{123}(x_1, x_2, x_3) f_{34}(x_3, x_4) f_{45}(x_4, x_5),$$

where  $f_{123}$  is an arbitrary nonnegative function on three binary variables and  $f_{34}, f_{45}$  are arbitrary nonnegative pairwise functions.

We compute the marginal

$$\mu_1(x_1) \propto \sum_{x_2, x_3, x_4, x_5} f(x_1, x_2, x_3, x_4, x_5)$$

by eliminating variables in a favorable order.

**Good elimination order:**  $x_5, x_4, x_2, x_3$ . Eliminate  $x_5$ :

$$g_4(x_4) \triangleq \sum_{x_5} f_{45}(x_4, x_5),$$

so

$$f \propto f_{123}(x_1, x_2, x_3) f_{34}(x_3, x_4) g_4(x_4).$$

Eliminate  $x_4$ :

$$g_3(x_3) \triangleq \sum_{x_4} f_{34}(x_3, x_4) g_4(x_4).$$

Eliminate  $x_2$ :

$$h_{13}(x_1, x_3) \triangleq \sum_{x_2} f_{123}(x_1, x_2, x_3).$$

Finally eliminate  $x_3$ :

$$\mu_1(x_1) \propto \sum_{x_3} h_{13}(x_1, x_3) g_3(x_3).$$

The largest intermediate factor here is  $h_{13}(x_1, x_3)$ , which has size  $2^2$ .

**Bad elimination order: eliminate  $x_3$  first.** If we eliminate  $x_3$  first, then we must form

$$\tilde{h}(x_1, x_2, x_4) \triangleq \sum_{x_3} f_{123}(x_1, x_2, x_3) f_{34}(x_3, x_4),$$

which is a factor on three variables (size  $2^3$ ). This illustrates how the elimination order controls the size of intermediate factors and hence the total complexity.

**Example 2: Exact BP on a tree (BN converted to a factor graph)**

Consider the tree-structured model from above with binary variables  $x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}$  and factors corresponding to a BN:

$$f(x_1, \dots, x_5) = f_1(x_1) f_2(x_2) f_{13}(x_1, x_3) f_{124}(x_1, x_2, x_4) f_{25}(x_2, x_5),$$

where the factor define the conditionals from the BN. Take numerical values:

$$p(x_1 = 1) = 0.6, \quad p(x_2 = 1) = 0.5,$$

$$p(x_3 = 1 \mid x_1) = \begin{cases} 0.9, & x_1 = 1 \\ 0.2, & x_1 = 0, \end{cases} \quad p(x_5 = 1 \mid x_2) = \begin{cases} 0.8, & x_2 = 1 \\ 0.1, & x_2 = 0, \end{cases}$$

and

$$p(x_4 = 1 \mid x_1, x_2) = \begin{array}{c|cc} & x_2 = 0 & x_2 = 1 \\ \hline x_1 = 0 & 0.1 & 0.6 \\ x_1 = 1 & 0.7 & 0.9 \end{array}.$$

Suppose we observe  $x_3 = 1$  and  $x_5 = 0$ . We compute  $p(x_4 = 1 \mid x_3 = 1, x_5 = 0)$ .

**Upward message from  $x_3$  to  $x_1$ .** The factor-to-variable message satisfies

$$\hat{m}_{13 \rightarrow 1}(x_1) \propto p(x_3 = 1 \mid x_1) = \begin{cases} 0.9, & x_1 = 1 \\ 0.2, & x_1 = 0. \end{cases}$$

Thus the (unnormalized) belief at  $x_1$  is

$$b_1(x_1) \propto p(x_1) \hat{m}_{13 \rightarrow 1}(x_1),$$

so

$$b_1(1) \propto 0.6 \cdot 0.9 = 0.54, \quad b_1(0) \propto 0.4 \cdot 0.2 = 0.08.$$

Normalize:

$$b_1(1) = \frac{0.54}{0.62} \approx 0.87097, \quad b_1(0) = \frac{0.08}{0.62} \approx 0.12903.$$

**Upward message from  $x_5$  to  $x_2$ .** Since  $x_5 = 0$ ,

$$\hat{m}_{25 \rightarrow 2}(x_2) \propto p(x_5 = 0 \mid x_2) = \begin{cases} 0.2, & x_2 = 1 \\ 0.9, & x_2 = 0. \end{cases}$$

So

$$b_2(x_2) \propto p(x_2) \hat{m}_{25 \rightarrow 2}(x_2),$$

yielding

$$b_2(1) \propto 0.5 \cdot 0.2 = 0.10, \quad b_2(0) \propto 0.5 \cdot 0.9 = 0.45.$$

Normalize:

$$b_2(1) = \frac{0.10}{0.55} \approx 0.18182, \quad b_2(0) = \frac{0.45}{0.55} \approx 0.81818.$$

**Compute the posterior on  $x_4$ .** On this tree, the posterior for  $x_4$  is obtained exactly by summing over  $x_1, x_2$ :

$$p(x_4 | x_3 = 1, x_5 = 0) = \sum_{x_1, x_2} b_1(x_1) b_2(x_2) p(x_4 | x_1, x_2).$$

For  $x_4 = 1$ , we compute the four contributions:

$$\begin{aligned} (x_1, x_2) = (0, 0) : & \quad 0.12903 \cdot 0.81818 \cdot 0.1 \approx 0.010557, \\ (0, 1) : & \quad 0.12903 \cdot 0.18182 \cdot 0.6 \approx 0.014076, \\ (1, 0) : & \quad 0.87097 \cdot 0.81818 \cdot 0.7 \approx 0.498512, \\ (1, 1) : & \quad 0.87097 \cdot 0.18182 \cdot 0.9 \approx 0.142929. \end{aligned}$$

Summing gives

$$p(x_4 = 1 | x_3 = 1, x_5 = 0) \approx 0.666074.$$

Therefore,

$$p(x_4 = 1 | x_3 = 1, x_5 = 0) \approx 0.666.$$

### Example 3: Forward–backward on a binary Markov chain with evidence

Let  $x_i \in \{0, 1\}$  and consider a Markov chain with transition matrix

$$T = \begin{pmatrix} p(0 \rightarrow 0) & p(0 \rightarrow 1) \\ p(1 \rightarrow 0) & p(1 \rightarrow 1) \end{pmatrix} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}.$$

Suppose we observe  $x_2 = 1$  and  $x_4 = 0$ . Compute  $p(x_3 = 1 | x_2 = 1, x_4 = 0)$ .

**Left (forward) message into  $x_3$ .** Given  $x_2 = 1$ ,

$$m_{2 \rightarrow 3}(x_3) \propto p(x_3 | x_2 = 1) = \begin{cases} 0.2, & x_3 = 0 \\ 0.8, & x_3 = 1. \end{cases}$$

**Right (backward) message into  $x_3$ .** Given  $x_4 = 0$ ,

$$m_{4 \rightarrow 3}(x_3) \propto p(x_4 = 0 | x_3) = \begin{cases} p(0 | 0) = 0.9, & x_3 = 0 \\ p(0 | 1) = 0.2, & x_3 = 1. \end{cases}$$

**Combine and normalize.** Thus

$$p(x_3 | x_2 = 1, x_4 = 0) \propto m_{2 \rightarrow 3}(x_3) m_{4 \rightarrow 3}(x_3),$$

so the unnormalized weights are

$$w(0) = 0.2 \cdot 0.9 = 0.18, \quad w(1) = 0.8 \cdot 0.2 = 0.16.$$

Normalize by  $w(0) + w(1) = 0.34$ :

$$p(x_3 = 1 | x_2 = 1, x_4 = 0) = \frac{0.16}{0.34} \approx 0.4706.$$

**Example 4: Peeling / iterative decoding on the BEC (and a stopping set)**

Consider a binary linear code with parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

The checks are:

$$c_1 : x_1 \oplus x_2 \oplus x_4 = 0, \quad c_2 : x_2 \oplus x_3 \oplus x_5 = 0, \quad c_3 : x_1 \oplus x_3 \oplus x_6 = 0.$$

**Case 1: Peeling succeeds.** Suppose the erasure channel erases  $\{x_1, x_3, x_4\}$  and reveals

$$x_2 = 0, \quad x_5 = 1, \quad x_6 = 0.$$

Then:

- From  $c_2$ :  $x_2 \oplus x_3 \oplus x_5 = 0$  gives

$$0 \oplus x_3 \oplus 1 = 0 \Rightarrow x_3 = 1.$$

- From  $c_3$ :  $x_1 \oplus x_3 \oplus x_6 = 0$  gives

$$x_1 \oplus 1 \oplus 0 = 0 \Rightarrow x_1 = 1.$$

- From  $c_1$ :  $x_1 \oplus x_2 \oplus x_4 = 0$  gives

$$1 \oplus 0 \oplus x_4 = 0 \Rightarrow x_4 = 1.$$

All erased bits are recovered.

**Case 2: A stopping set (peeling stalls).** If instead the erasures are  $\{x_1, x_2, x_3\}$  and  $x_4, x_5, x_6$  are known, then each check contains at least two erased variables:

$$c_1 : \{x_1, x_2\} \text{ erased}, \quad c_2 : \{x_2, x_3\} \text{ erased}, \quad c_3 : \{x_1, x_3\} \text{ erased}.$$

No degree-one check exists, so peeling cannot start. The erased set contains a stopping set, which is exactly the obstruction to iterative decoding on the BEC.

**Appendix: Deferred Proofs**

**Lemma 1** (Independence across connected components). *Let  $G$  be a factor graph whose variable and factor nodes decompose into connected components  $\mathcal{C}_1, \dots, \mathcal{C}_k$  with variable sets  $V_\ell$  and factor sets  $F_\ell$ . If*

$$f(x) = \prod_{a \in F} f_a(x_{\partial a}), \quad \mu(x) = \frac{1}{Z} f(x),$$

then

$$f(x) = \prod_{\ell=1}^k g_\ell(x_{V_\ell}), \quad \mu(x) = \prod_{\ell=1}^k \mu_\ell(x_{V_\ell}),$$

where  $g_\ell(x_{V_\ell}) \triangleq \prod_{a \in F_\ell} f_a(x_{\partial a})$  and  $\mu_\ell(x_{V_\ell}) \propto g_\ell(x_{V_\ell})$ . In particular, variables in distinct connected components are independent.

*Proof.* Since there are no edges between components, each factor in  $F_\ell$  depends only on variables in  $V_\ell$ , and the full product splits as  $f(x) = \prod_{\ell=1}^k g_\ell(x_{V_\ell})$ . Therefore

$$Z = \sum_x f(x) = \prod_{\ell=1}^k \sum_{x_{V_\ell}} g_\ell(x_{V_\ell}),$$

so  $\mu(x)$  factors into a product of normalized component marginals  $\mu_\ell$ . This implies independence across components.  $\square$

**Theorem 2** (Separation implies conditional independence in factor graphs). *Let  $G$  be a factor graph with variable set  $[n]$  and factor set  $F$ , and let*

$$f(x) = \prod_{a \in F} f_a(x_{\partial a}), \quad \mu(x) = \frac{1}{Z} f(x).$$

*Let  $A, B, S \subseteq [n]$  be disjoint. If every path in  $G$  from a node in  $A$  to a node in  $B$  passes through a node in  $S$ , then  $X_A \perp\!\!\!\perp X_B \mid X_S$  under  $\mu$ .*

*Proof.* Remove the variable nodes in  $S$  (and incident edges) and view the remaining graph as a factor graph on  $[n] \setminus S$ . By assumption,  $A$  and  $B$  lie in distinct connected components of this reduced graph. Fixing  $x_S$  multiplies  $f(x)$  by a factor depending only on  $x_S$  and removes all factors incident to  $S$  from the remaining variables. Hence the conditional distribution  $\mu(\cdot \mid x_S)$  is represented by this reduced factor graph up to normalization. By the lemma above, variables in distinct connected components are independent under  $\mu(\cdot \mid x_S)$ , so  $X_A \perp\!\!\!\perp X_B \mid X_S$ .  $\square$

The Hammersley–Clifford theorem gives necessary and sufficient conditions under which a strictly positive probability distribution can be represented as a Markov random field.

**Theorem 3** (Hammersley–Clifford (discrete, positive case)). *Let  $G = (V, E)$  be an undirected graph and let  $\mu$  be a strictly positive distribution on  $X_V = \prod_{i \in V} \mathcal{X}_i$  (with each  $\mathcal{X}_i$  finite). Then the following are equivalent:*

- (Global Markov) *For disjoint  $A, B, S \subseteq V$ , if  $S$  separates  $A$  and  $B$  in  $G$ , then  $X_A \perp\!\!\!\perp X_B \mid X_S$ .*
- (Clique factorization) *The distribution  $\mu(x)$  factorizes over cliques:*

$$\mu(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C),$$

*where  $\mathcal{C}$  is the set of (maximal) cliques of  $G$  and each  $\psi_C > 0$ .*

*Proof.* (Clique factorization  $\Rightarrow$  Global Markov) Fix disjoint  $A, B, S \subseteq V$  with  $S$  separating  $A$  and  $B$ . Remove  $S$  from the graph; let  $U_1, \dots, U_r$  be the connected components of  $G[V \setminus S]$ . Because there are no edges between different  $U_t$ , any clique  $C$  intersects at most one component  $U_t$  (it may also contain vertices in  $S$ ). Therefore we can regroup clique factors as

$$\mu(x) = \frac{1}{Z} \phi_S(x_S) \prod_{t=1}^r \phi_t(x_{U_t}, x_S),$$

for suitable positive functions  $\phi_S, \phi_t$ . Conditioning on  $x_S$  gives

$$\mu(x_{V \setminus S} \mid x_S) \propto \prod_{t=1}^r \phi_t(x_{U_t}, x_S),$$

so the random vectors  $(X_{U_t})_{t=1}^r$  are mutually independent given  $X_S$ . Since  $A$  and  $B$  lie in different unions of these components, this implies  $X_A \perp\!\!\!\perp X_B \mid X_S$ .

(*Global Markov*  $\Rightarrow$  *Clique factorization*) Assume  $\mu(x) > 0$  for all  $x$ . Choose a reference configuration  $x^0$  and write  $F(x) := \log \mu(x)$ . For each subset  $C \subseteq V$ , define

$$U_C(x_C) := \sum_{B \subseteq C} (-1)^{|C|-|B|} F(x_B, x_{V \setminus B}^0).$$

By Mobius inversion (ANOVA decomposition),

$$F(x) = \sum_{C \subseteq V} U_C(x_C).$$

Now let  $C$  be non-clique. Then there exist nonadjacent  $u, v \in C$ . By the global Markov property with  $A = \{u\}$ ,  $B = \{v\}$ , and  $S = V \setminus \{u, v\}$ , we have  $X_u \perp\!\!\!\perp X_v \mid X_S$ . Equivalently, for every fixed  $x_S$ , the mixed log-difference in  $(u, v)$  is zero:

$$\Delta_{uv} F = F(x_u, x_v, x_S) - F(x_u^0, x_v, x_S) - F(x_u, x_v^0, x_S) + F(x_u^0, x_v^0, x_S) = 0.$$

In the Mobius expansion,  $U_C$  is a higher-order mixed interaction term. If  $C$  contains a nonedge  $\{u, v\}$ , that mixed interaction must vanish; hence  $U_C \equiv 0$  for every non-clique  $C$ . Therefore only clique terms remain:

$$F(x) = \text{const} + \sum_{C \in \mathcal{C}} U_C(x_C).$$

Exponentiating gives

$$\mu(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C), \quad \psi_C(x_C) := \exp(U_C(x_C)) > 0.$$

Finally, non-maximal clique factors can be absorbed into maximal-clique factors, so the displayed form over maximal cliques is obtained.  $\square$