What is a probabilistic graphical model (PGM)?

A probabilistic graphical model (PGM) combines probability theory and graph theory. It uses a graph to represent conditional independence assumptions between random variables, allowing a compact description of a joint probability distribution.

Simply put:

- Nodes: represent random variables.
- Edges: represent probabilistic dependencies between variables.

With this representation, a complex joint distribution $P(X_1, X_2, ..., X_n)$ can be factorized into local factors, reducing computational and storage complexity.

1. Core categories

Based on edge direction, graphical models are mainly divided into two categories:

1.1 Bayesian network

- Structure: Directed acyclic graph (DAG).
- Meaning: represents causal or explicit dependency relations.
- Factorization: The joint distribution factorizes into the product of each node's conditional probability given its parents.

  $$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid \mathrm{pa}(X_i))$$

  where $\mathrm{pa}(X_i)$ is the parent set of node X_i.
- Typical applications: disease diagnosis (cause → symptom), hidden Markov models (HMM).

1.2 Markov random field (MRF)

- Structure: Undirected graph.
- Meaning: represents correlations or constraints among variables, without explicit direction (e.g., neighboring pixels in an image).
- Factorization: Defined via maximal cliques and potential functions.

  $$P(X) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_{C(X_C)}$$

  where:

  - $\mathcal{C}$ is the set of all maximal cliques in the graph.
  - $\psi_{C(X_C)}$ is the potential function on clique C, typically requiring $\psi_C \geq 0$.
  - Z is the partition function for normalization:

    $$Z = \sum_X \prod_{C \in \mathcal{C}} \psi_{C(X_C)}$$
- Typical applications: image segmentatiFon, conditional random fields (CRF).

2. The three basic problems in graphical models

In practice, we focus on three core problems:

2.1 Representation

How do we choose the graph structure and parameters to model the real world?

- Choose directed or undirected graphs?
- Define which variables are conditionally independent?
- **Key idea**: use conditional independence $X \perp Y \mid Z$ to sparsify the graph.

2.2 Inference

Given observed variables E (Evidence), compute the posterior $P(Y \mid E)$ of hidden variables Y.

Common algorithms:

- Exact inference:
  - Variable Elimination.
  - Clique Tree / Junction Tree Algorithm.
  - Belief Propagation (BP) (exact on trees).
- Approximate inference (for complex graphs):
  - Variational Inference: fit a simple distribution $Q(Y)$ to approximate $P(Y|E)$.
  - Monte Carlo sampling (MCMC): e.g., Gibbs sampling.

2.3 Learning

Given a dataset, how do we learn the model?

- Parameter learning: with known graph structure, estimate parameters of CPTs or potentials (MLE or EM).
- Structure learning: both parameters and graph structure are unknown and must be inferred from data.

3. Example: a simple Bayesian network

Suppose we have three variables:

- R: Rain
- S: Sprinkler
- G: Grass Wet

Relationships:

1. Rain R affects grass wetness G.
2. Sprinkler S affects grass wetness G.
3. Assume R and S are independent (simplified).

Graph: $R \to G, S \to G$.

The joint distribution is:

$$P(R, S, G) = P(R) \cdot P(S) \cdot P(G \mid R, S)$$

Compared to storing all $2^3 = 8$ combinations, we only store $P(R), P(S)$, and $P(G|R, S)$, greatly reducing parameter count.

4. Summary

Feature	Bayesian network	Markov network
Graph type	Directed acyclic graph (DAG)	Undirected graph
Dependency	Causal relation	Correlation/spatial constraint
Local factor	Conditional probability $P(X \mid Y)$	Potential $\psi(X, Y)$
Normalization	Local normalization	Global normalization (Z)

2. Undirected probabilistic graph (Markov Random Field)

An undirected probabilistic graphical model, commonly called a **Markov random field (MRF)**, uses an undirected graph $G = (V, E)$ to describe the joint distribution of a set of random variables. Unlike Bayesian networks, MRF edges have no arrows, indicating correlation or constraints without explicit causal direction.

2.1 Factorization of the joint distribution

As noted above, an MRF factorizes its joint distribution based on **maximal cliques**.

Let $\mathcal{C}$ be the set of all maximal cliques. Let X_C be the variables in clique C. The joint distribution is a Gibbs distribution:

$$P(X) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_{C(X_C)}$$

where:

- $\psi_{C(X_C)} \geq 0$ are potentials (factors).
- Z is the **partition function**, the hardest part of the model:

$$Z = \sum_{X} \prod_{C \in \mathcal{C}} \psi_{C(X_C)}$$

Note: Computing Z requires summing over all possible configurations. If X has n binary variables, the sum has 2^n terms, which is usually intractable (NP-hard) for large n.

2.2 Markov properties (conditional independence)

In an undirected graph, connectivity directly defines conditional independence. There are three equivalent definitions (for strictly positive $P(X) > 0$):

1. **Pairwise Markov property** Let u and v be two nodes with no edge between them. Given all other nodes $X_{\{V \setminus \{u,v\}\}}$, u and v are independent.

$$X_u \perp X_v \mid X_{\{V \setminus \{u,v\}\}}$$

2. **Local Markov property** A node v is independent of all other nodes given its **neighbors** $N(v)$.

$$X_v \perp X_{\{V \setminus (\{v\} \cup N(v))\}} \mid X_{\{N(v)\}}$$

Here, $N(v)$ is the **Markov blanket** of v. In undirected graphs, it is just the neighbors; in directed graphs, it is more complex (parents, children, and parents of children).

3. **Global Markov property** The most intuitive definition. Let node sets A and B be **separated** by S (all paths from A to B pass through S), then:

$$X_A \perp X_B \mid X_S$$

2.3 Energy models and physics

MRFs are closely related to statistical physics. We can write potentials as exponentials:

$$\psi_{C(X_C)} = \exp\left(-E_{C(X_C)}\right)$$

where E_C is an **energy function**.

Then the joint distribution becomes a Boltzmann distribution:

$$P(X) = \frac{1}{Z} \exp\left(-\sum_{C \in \mathcal{C}} E_{C(X_C)}\right) = \frac{1}{Z} \exp\left(-E_{\text{total}(X)}\right)$$

This explains why we often say: **lower energy means higher probability**.

Classic example: Ising model Used to model ferromagnetism. Nodes are arranged on a grid, each node $x_i \in \{+1, -1\}$. The energy function is:

$$E(x) = -\sum_{(i,j) \in E} J x_i x_j - \sum_i h x_i$$

- The first term says neighboring spins prefer to align (if $J > 0$).
- The second term captures the effect of an external field.

This is a classic pairwise MRF.

2.4 Application scenarios

Because undirected graphs model "neighbor relations" well, they are widely used in:

- **Computer vision**:
  - Image segmentation: neighboring pixels prefer the same label.
  - Image denoising: observed pixels correlate with true pixels, and true pixels vary smoothly.
- **Natural language processing**:
  - Conditional random fields (CRF): sequence labeling (e.g., NER), more flexible than HMM for long-range dependencies.
- **Spatial statistics**:
  - Predict spatially correlated variables.

2.5 Summary: directed vs undirected

Dimension	**Bayesian network (directed)**	**Markov network (undirected)**
Factor definition	Conditional probability $P(X\|Y)$, locally normalized	Potential $\psi(X, Y)$, requires global normalization Z

Dimension	**Bayesian network (directed)**	**Markov network (undirected)**
Independence	Dependence direction (d-separation)	Connectivity (graph separation)
Applicability	Causal inference, logical reasoning	Images, spatial, constraint systems

3. PGM perspective in QEC

Quantum error correction (especially topological codes) can be mapped perfectly to inference on an undirected graphical model (MRF).

3.1 Mapping: Tanner graph and factor graph

In QEC, we often use Tanner graphs to describe codes. This is essentially the factor graph in a graphical model:

- Variable nodes: physical qubit error states. Let $E = \{e_1, e_2, ..., e_n\}$ be error chains, typically $e_i \in \{I, X, Y, Z\}$. In the simplest model, we only consider bit flips, $e_i \in \{0, 1\}$. This corresponds to hidden variables X.
- Factor nodes: stabilizer measurements or parity checks. These nodes define constraints between variables.
- Evidence: the syndrome S. When we measure stabilizers and violate a constraint (e.g., odd parity), we observe a non-zero syndrome.

3.2 From MAP to MMAP

Understanding why QEC is an MMAP problem hinges on the difference between "most likely physical error" and "most likely logical error".

1. MAP - find a specific error If we try to find the single most likely physical error chain E^*:

   $$E^* = \text{argmax}_E P(E \mid S)$$

   This corresponds to what **minimum weight perfect matching (MWPM)** solves (in non-degenerate cases): the shortest path explaining the syndrome.

2. MMAP - find a logical equivalence class In QEC, we apply a correction R. If R differs from the true error E by a stabilizer g (i.e., $R = E \cdot g$), the correction succeeds because stabilizers do not affect the logical state.

   This introduces degeneracy: **many different physical errors E correspond to the same logical class**.

   We group all physical errors by their logical operator homology class $|(L)$. What we care about is: **which logical class is most likely?**

   We sum probabilities over all physical errors in the same class, then take the maximum:

   $$|(L)^* = \text{argmax}_{|(L)} \sum_{E \in \, |(L)} P(E \mid S)$$

   This is the standard MMAP problem:

   - Marginal: sum (marginalize) over physical details we do not care about.

- MAP: maximize at the logical-class level.

3.3 Physical meaning: Ising model and partition function

This mapping is physically elegant. For the surface code:

- The error model maps to a random-bond Ising model.
- Solving MMAP is equivalent to computing the free energy of this statistical system.

To compare two logical classes $|(L)_1$ and $|(L)_2$, we compute their partition function ratio:

$$Z_1 = \sum_{E \in |(L)_1} e^{-\beta H(E)}, \quad Z_2 = \sum_{E \in |(L)_2} e^{-\beta H(E)}$$

If $Z_1 > Z_2$, we infer logical class 1.

Exact computation of Z is #P-COMPLETE, which is why QEC decoders based on tensor networks or renormalization group aim to approximate this sum efficiently.

4. QEC decoding algorithm analysis

In QEC, decoding is the problem of using the syndrome S to recover logical information. For degenerate codes, this is exactly the MMAP problem:

$$\hat{L} = \text{argmax}_L \sum_{E \in L} P(E \mid S)$$

Here, "sum" handles degeneracy (different physical errors with the same logical effect), and "max" selects the most likely logical class.

4.1 Baseline: Belief Propagation (BP)

Before evaluating other algorithms, we must clarify BP's role.

4.1.1 Two modes of BP

BP passes messages on a factor graph to update beliefs.

- Sum-Product BP: computes marginals $P(x_i \mid S)$.
  - **In QEC**: computes error probability for each physical qubit (summing over all error combinations, i.e., "Sum").
- Max-Product BP: computes the global MAP configuration.
  - **In QEC**: finds the most likely specific error chain (similar to MWPM).

4.1.2 Limitations of BP in QEC

Although BP is $O(N)$ and efficient, it faces three challenges in QEC:

1. Loop problem (short cycles): quantum codes (especially high-threshold codes) often have many short cycles, causing BP to oscillate or fail to converge.
2. MMAP compatibility: standard BP is either all Sum or all Max. MMAP requires Sum over some variables (physical degeneracy) and Max over others (logical classes). Using Sum-Product BP with hard decisions is a heuristic MMAP approximation.
3. Validity failure: the converged BP result may not satisfy parity checks (i.e., $H\hat{e} \neq S$).

4.2 Deep evaluation of AI-recommended approaches

Given QEC requirements (high throughput, low latency, degeneracy), we analyze AI-proposed solutions one by one.

Category 1: Exact inference

(A1) Junction Tree / (A2) AND/OR Search

- Analysis: These methods decompose by treewidth.
- QEC applicability: **Very low**.
  - High-performance quantum codes (Surface Code, Expander Codes) are designed with high connectivity and treewidth to avoid logical errors.
  - Exact methods are exponential: $O(\exp(\text{treewidth}))$.
- Conclusion: Only suitable for very small codes (5-qubit, 7-qubit), not practical.

Category 2: Approximate with bounds

(B1) Mini-Bucket / (B2) TRW (Tree-Reweighted) / (B3) LP relaxation

- (B1) Mini-Bucket & (B2) TRW:
  - Idea: relax graph structure or use convex optimization to compute upper/lower bounds on Z.
  - Pros: better convergence than standard BP, less oscillation.
  - Cons: higher computational cost and harder to parallelize.
  - QEC potential: TRW can outperform BP on some hard codes, but speed is a bottleneck.
- (B3) LP (linear programming) / ADMM:
  - Idea: model decoding as an LP. MWPM can be seen as a dual LP.
  - QEC potential: **Medium**. For non-degenerate codes, LP decoders have guarantees, but for degenerate codes (needs Sum), LP often ignores entropy benefits from degeneracy.

Category 3: Fast heuristic approximations

This is the most active area in QEC research, and the direction you should focus on.

- (C1) Sum-Max BP (Mixed-Product BP):
  - Idea: theoretically closest to MMAP. Define two variable classes and use Sum for one and Max for the other.
  - QEC issue: numerically unstable. Mixing Sum and Max in floating point often causes underflow or precision loss. Rarely successful in QEC.
- (C2) BP + Decimation:
  - Idea: run BP → fix the most confident bits → simplify the graph → repeat.
  - QEC applicability: **High**. It breaks symmetry and loops, helping BP converge. A standard enhancement.
- (C3) GBP (Generalized BP):
  - Idea: pass messages between regions, not just nodes.
  - QEC applicability: **Very high (for topological codes)**.
  - Surface Code has many short cycles (plaquettes). GBP explicitly handles these loops and significantly improves thresholds. Complexity grows exponentially with region size.
- (C4) BP + OSD (Ordered Statistics Decoding):
  - Idea: run Sum-Product BP to get soft information, then apply OSD as post-processing. OSD orders by reliability and flips the least reliable bits to satisfy checks.

- QEC applicability: **Current SOTA (best practice)**.
- Reason: combines BP speed (soft info) with linear-algebra rigor (guaranteed valid correction). The standard benchmark for LDPC codes is BP+OSD.

4.3 Summary and roadmap

If your goal is efficient MMAP decoding, a recommended path is:

Phase	**Algorithm choice**	**Reason**
1. Start	Sum-Product BP	Implement basic message passing as a baseline. Use log-domain computation to avoid underflow.
2. Intermediate (practical)	BP + OSD (Order-0 / Order-E)	This is the industry standard. It fixes BP non-convergence or invalid outputs. Higher OSD order improves accuracy but costs speed.
3. Code-specific	GBP (generalized BP)	If you mainly study surface codes or color codes (many short cycles), GBP outperforms BP + OSD.
4. Exploratory	Neural BP / Weighted BP	Train weights in BP using ML to approximate MMAP.

Core conclusion: Do not attempt exact Junction Tree or direct Sum-Max BP. Start with BP + OSD. It uses OSD post-processing to compensate for BP's inability to fully solve MMAP, and is the most cost-effective MMAP approximation today.

Note

1. What is a maximal clique?

In an undirected graph, the concept of a clique is the basis for defining local relationships.

Definitions and Distinctions

1. **Clique**: A subset of nodes such that every pair of nodes in the subset is connected by an edge (a fully connected subgraph).
2. **Maximal clique**: A clique that cannot be expanded by adding any other node while remaining a clique.

Example: Suppose an undirected graph G has node connections $A - B, B - C, A - C, B - D$.

- $\{A, B\}$ is a clique.
- $\{A, B, C\}$ is a clique (a triangle), and you cannot add D (since neither A nor C connects to D), so $\{A, B, C\}$ is a **maximal clique**.
- $\{B, D\}$ is also a **maximal clique**.

Role: In a Markov random field, maximal cliques define the smallest units for factorizing the probability distribution. If two variables are not in the same maximal clique, it means they have no direct strong coupling (or their direct relation is already contained in a larger clique).

2. What is a potential function?

Because an undirected graph has no direction, we cannot define conditional probabilities like $P(A|B)$ (since A and B are on equal footing). Instead, we use a potential function to quantify compatibility between variables.

Properties and Meaning

Definition: A non-negative real-valued function $\psi_{C(X_C)}$ defined on a maximal clique C, where X_C is the set of variables in the clique.

Key features:

1. **Non-negativity**: $\psi_{C(X_C)} \geq 0$.
2. **Not a probability**: A potential function is not a probability and does not need to be normalized (the sum does not have to be 1).
3. **Intuition**:
   - A larger ψ value indicates a more likely configuration.
   - It can be seen as the inverse of energy. Often $\psi_{C(x)} = \exp(-E(x))$, where $E(x)$ is an energy function. Lower energy implies a more stable system and a higher probability.

Example: In image processing, adjacent pixels x_i, x_j tend to have similar colors. We can define a potential $\psi(x_i, x_j)$ that takes a large value when $x_i \approx x_j$, and a small value when they differ.

3. Why can the joint probability factor into a product of potentials?

This is one of the deepest results in graphical models, known as the **Hammersley-Clifford theorem**.

Hammersley-Clifford Theorem

Question: We want the joint distribution $P(X)$ to satisfy the conditional independence implied by the graph structure (Markov property).

Theorem: If a distribution $P(X) > 0$ (strictly positive) satisfies the local Markov property defined by an undirected graph G, then it can be factorized as a product of potential functions over all maximal cliques in the graph:

$$P(X) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_{C(X_C)}$$

Why is this true? (Intuition)

1. **Localization principle**: Graph theory tells us that direct interactions are limited to nodes connected by an edge. Maximal cliques contain all groups of variables with direct mutual influence.
2. **Independence is reflected**: If two variables x_i and x_j never appear in the same potential ψ_C, it means there is no direct interaction between them. This matches the absence of an edge in the graph and ensures conditional independence (given a separating set).
3. **Necessity of a product form**: When we consider two independent subsystems (a disconnected graph), the joint probability should factor as $P(A, B) = P(A)P(B)$. A product of potentials naturally satisfies this, while a sum does not.

4. Partition function Z in detail: what are "all possible variable configurations"?

In the definition of a Markov random field, the partition function Z is written as:

$$Z = \sum_{X} \prod_{C \in \mathcal{C}} \psi_{C(X_C)}$$

The term $\sum_X$ is often the hardest part for beginners. It is not a sum over numeric values, but an enumeration over all possible world states.

Definition of a Configuration

A "configuration" (assignment) means assigning a specific value to every random variable in the graph at the same time.

If the graph has n variables $X_1, X_2, \ldots, X_n$, then "all possible configurations" are the Cartesian product of their values.

4.1 Example: understanding Z with 3 variables

Suppose we have a very simple model with only 3 variables A, B, C.

- Assume they are all binary (e.g., heads/tails), with values $\{0, 1\}$.

- Assume the model defines a global potential $\psi(A, B, C)$ (for simplicity, no clique decomposition).

Then there are $2^3 = 8$ possible configurations. Computing Z means summing the "unnormalized probabilities" from these 8 parallel worlds.

We list a "truth table" to show how Z is computed:

Variable A	**Variable B**	**Variable C**	**"Score" of this configuration** \ $s_i = \psi(A, B, C)$
0	0	0	$s_1 = \psi(0, 0, 0)$
0	0	1	$s_2 = \psi(0, 0, 1)$
0	1	0	$s_3 = \psi(0, 1, 0)$
0	1	1	$s_4 = \psi(0, 1, 1)$
1	0	0	$s_5 = \psi(1, 0, 0)$
1	0	1	$s_6 = \psi(1, 0, 1)$
1	1	0	$s_7 = \psi(1, 1, 0)$
1	1	1	$s_8 = \psi(1, 1, 1)$
		Partition function $Z =$	$s_1 + s_2 + s_3 + s_4 + s_5 + s_6 + s_7 + s_8$

Table 1: Computing Z requires traversing all rows

After computing Z, if we want the probability of a specific state (for example all 1s), we divide that row's score by the total:

$$P(A = 1, B = 1, C = 1) = \frac{s_8}{Z}$$

4.2 Why is this hard? (The Partition Function Problem)

Exponential blowup

In the example above, 3 variables require 8 sums. It seems easy for a computer.

But in real applications (e.g., image processing):

- A small image might have 100×100 pixels.
- The number of variables is $n = 10,000$.
- Each pixel has 2 values (black/white).
- Total configurations = $2^{10,000}$.

$2^{10,000} \approx 10^{3000}$.

For comparison, the total number of atoms in the observable universe is about 10^{80}. This means no computer can compute Z exactly by "traversing all configurations".

That is why in undirected models (such as CRFs and RBMs) we usually cannot do exact inference, and instead use:

1. Sampling: e.g., MCMC, which "walks" around high-probability configurations without traversing all of them.

2. Approximate inference: e.g., variational inference, which fits a simple distribution to approximate a complex one.

5. Relationship between Tanner graphs, factor graphs, and undirected graphs

In QEC and probabilistic inference, these terms are often used interchangeably, but they live at different levels of abstraction. We can see them as a relationship of containment and concretization.

5.1 Conceptual hierarchy

In QEC and probabilistic inference, these three concepts represent different levels from an "abstract mathematical model" to a "concrete implementation structure". We compare them visually.

We use a simple three-variable correlation model as an example: variables A, B, C, as shown below.

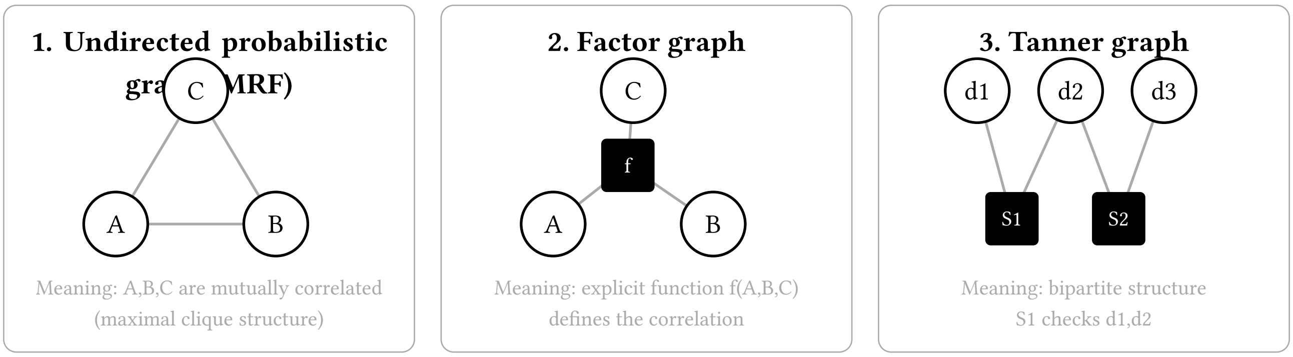

1. Undirected probabilistic graph (MRF)
C
A
B
Meaning: A,B,C are mutually correlated
(maximal clique structure)
2. Factor graph
C
f
A
B
Meaning: explicit function f(A,B,C)
defines the correlation
3. Tanner graph
d1
d2
d3
S1
S2
Meaning: bipartite structure
S1 checks d1,d2

Figure 1: Structural comparison of three graph models

A simple containment relation is:

$$\text{Tanner Graph} \subset \text{Factor Graph} \subset \text{Representation of MRF}$$

Core differences among the three

1. Undirected probabilistic graph (MRF): This is the mathematical model. It abstractly describes correlations between variables (via maximal cliques), but does not necessarily specify the exact factor structure.
2. Factor graph: This is a fine-grained representation of an MRF. It introduces extra "factor nodes" to explicitly show the scope of potential functions, resolving ambiguity in complex clique structures.
3. Tanner graph: This is a special case of a factor graph in coding theory. It is used to describe linear block codes (such as LDPC codes or stabilizer codes in QEC).

5.2 Deep dive

5.2.1 From an undirected graph (MRF) to a factor graph

In a standard undirected graph, if three variables A, B, C form a triangle (a clique), we only know they are related, but not whether the relation is pairwise or joint.

A factor graph removes this ambiguity by turning the graph into a bipartite graph:

- Circle nodes: variables X_i.
- Square nodes: factors f_j.
- Edges: a variable X_i connects to a factor f_j only if X_i is an argument of f_j.

Formula correspondence:

$$P(X) = \frac{1}{Z} \prod_j f_{j(X_{\text{scope}(j)})}$$

5.2.2 From a factor graph to a Tanner graph

A Tanner graph is essentially a factor graph with binary variables, but with specific physical/logical meanings:

- Variable nodes: codeword bits (Data Qubits).
- Factor nodes: check bits (Check Qubits / Stabilizers / Parity Checks).
- Mathematical form of factors: In a Tanner graph, the “potential” is usually a hard constraint or indicator function.

  For example, for a parity check, the factor f is defined as:

  $$f(x_1, x_2, x_3) = \begin{cases} 1, \text{if } x_1 + x_2 + x_3 = 0 \pmod 2 \\ 0, \text{otherwise} \end{cases}$$

  In QEC, this means if the physical error E commutes with the stabilizer S, the probability (weight) is non-zero; otherwise it is forbidden (or extremely small in soft decoding).

5.3 Summary table

Name	Structural features	QEC correspondence
Undirected probabilistic graph (MRF)	Ordinary graph, nodes are variables and edges are dependencies	Describes entanglement/statistical correlations between qubits
Factor graph	Bipartite graph, explicitly separates “variables” and “functions”	A general decoding framework, the carrier for Belief Propagation (BP)
Tanner graph	Subset of factor graphs, with factors as parity checks	Describes stabilizer code structure: variables = physical qubits, factors = stabilizer measurements

6. Differences between MAP and MMAP: from intuition to math

In probabilistic inference, MAP and MMAP differ by only one letter, but their logic for finding the “best answer” is completely different. For QEC, understanding this is crucial.

6.1 Definition comparison

Suppose we have two variables:

- X: the variable we care about (e.g., logical error type).
- Y: the variable we do not care about but that does exist (e.g., physical error details).

- E: observed evidence (e.g., syndrome).

Mathematical definitions

1. MAP (Maximum A Posteriori) Find the most likely specific global configuration (x, y).

$$(x^*, y^*) = \text{argmax}_{x,y} P(x, y \mid E)$$

"Find the single most likely microscopic scenario."

2. MMAP (Marginal MAP) Find the most likely target variable configuration x, marginalizing over y.

$$x^* = \text{argmax}_x \sum_y P(x, y \mid E) = \text{argmax}_x P(x \mid E)$$

"First add up all possibilities that share the same x, then see which group has the largest total probability."

6.2 Intuitive example: election and votes

To see why MAP and MMAP give different answers, consider an "election paradox" example.

Suppose a class elects a leader. Candidates belong to two groups: Logical party (L) and Physical party (P).

- The Logical party has one candidate: L_1.
- The Physical party has three candidates: P_1, P_2, P_3.

Vote shares (posterior probabilities) are:

Group (macro)	Candidate (micro)	Vote share (probability)
Logical party	L_1	**40%**
Physical party	P_1	25%
	P_2	20%
	P_3	15%

Table 2: Decision difference between MAP and MMAP

Inference comparison:

1. MAP view (highest single point):
   - Who is the highest-vote individual?
   - Answer: L_1 (40%).
   - **Conclusion**: MAP says the Logical party wins.
2. MMAP view (group total):
   - Which group has the highest total vote share?
   - Logical party total: 40%.
   - Physical party total: $25\% + 20\% + 15\% = 60\%$.
   - **Conclusion**: MMAP says the Physical party wins.

Key insight

> MAP is easily attracted to a "spike" (a single high-probability state).
>
> MMAP focuses on probability mass: even if each state is not large, if there are many such states (high degeneracy), their total probability can win.

6.3 Meaning in QEC

This explains why MMAP is the correct answer in quantum error correction, while MAP (often approximated by MWPM) is only an approximation.

- Physical errors (micro-state): there can be thousands of concrete error paths (e.g., a chain going slightly left or right), corresponding to different Y.
- Logical errors (macro-state): all these paths lead to the same logical operation (e.g., logical X flip), corresponding to X.

QEC status:

- MWPM (minimum weight perfect matching): effectively does MAP, finding the single most probable error chain. When the error distribution is sharp (low-temperature limit), MAP and MMAP are close.
- Tensor Network / BP decoders: attempt to compute MMAP by summing all equivalent error chains (entropy effect). At higher noise, MMAP has a significantly higher decoding threshold than MAP.

7. What are degenerate codes?

In classical error-correcting codes, each error usually corresponds to a unique syndrome, so the decoder only needs to find that unique error. In quantum error correction, things get more interesting and complex.

> **Definition**
>
> Degeneracy means multiple different physical error patterns E_1, E_2 have exactly the same logical effect on the quantum state (or are equivalent).
>
> Mathematically, if $E_2 = E_1 \cdot S$, where S is an element of the stabilizer group, then E_1 and E_2 are degenerate. The stabilizer acts as the identity I on the code space.

Why is it important?

- Bad: It makes MAP (maximum a posteriori) invalid. MAP tries to distinguish E_1 and E_2, which is meaningless and wastes computation.
- Good: It increases the probability of successful correction. We need MMAP, i.e., sum the probabilities of all equivalent errors $E_1, E_2, \ldots$. This "entropy gain" makes quantum codes more robust than expected.

8. Quantum codes and high-threshold codes

8.1 Quantum codes vs classical codes

Quantum codes (especially CSS codes) are usually built from two classical codes: one corrects X errors, the other corrects Z errors. The core constraint is the no-cloning theorem: we cannot copy qubits to check errors, we can only measure stabilizers (parity checks) to obtain information indirectly.

8.2 What is a threshold?

This is a key metric for code performance.

- If the physical error rate $p < p_{\text{th}}$, then as code length $N \to \infty$, the logical error rate $P_L \to 0$.
- If $p > p_{\text{th}}$, increasing code length increases logical error rate.

8.3 Common high-threshold codes

Type	**Representative: Surface Code**	**Representative: quantum LDPC codes**
Structure	2D grid, only nearest-neighbor interactions	Sparse random graphs with complex long-range connections
Threshold	High ($\approx 1\%$ under circuit noise)	Medium/high (depends on construction)
Code rate	Very low ($\frac{1}{N}$, decreases with size)	Finite constant ($\frac{k}{N}$ stays constant)
Decoding difficulty	Lower (MWPM, Union-Find)	Higher (requires BP+OSD)

9. The loop problem (Short Cycles / Loops)

This is the biggest enemy of BP on quantum codes.

> **Why are loops deadly?**
>
> BP assumes the graph is tree-like (no cycles).
>
> 1. Echo chamber effect: Suppose node A tells node B: “I think I have an error”. If there is a loop $A \to B \to C \to A$, that message comes back to A as: “C says you might also be in error”. A mistakes this as independent confirmation and becomes overconfident.
> 2. Positive feedback oscillation: This self-reinforcing loop causes probabilities to oscillate between 0 and 1, preventing convergence.

The QEC dilemma: Degenerate quantum codes (especially topological codes like the surface code) intrinsically contain many short loops (because stabilizers must commute, which geometrically forms closed shapes). This makes standard BP perform poorly and forces the use of GBP (handling loops) or OSD (breaking loop effects).

10. BP algorithm explained (an intuitive view for beginners)

Belief Propagation (BP), also called the sum-product algorithm, is essentially a “telephone game”.

10.1 Core roles

- Variable nodes (V): physical qubits. They want to know if they are in error (0 or 1).
- Check nodes (C): stabilizers. They enforce parity checks (e.g., an even number of 1s among neighbors).

10.2 Algorithm flow (iterative)

We can understand it using log-likelihood ratios (LLR): positive means a tendency toward 0, negative means a tendency toward 1.

1. Initialization (Input): Each variable node V has an initial belief (prior) based on channel noise (e.g., error rate 0.1%).
2. Check node update ("work with me"): Check node C tells each connected variable V_i: **"Based on the states of the other variables connected to me, what state should you be in to satisfy my parity check?"**

   Rule: If the other variables are confident about 0, you must comply. If they are uncertain, your message is also uncertain.
3. Variable node update ("I listen to everyone"): Variable node V collects suggestions from all connected checks C_k, adds its own prior, and simply sums them. **"C1 says I might be 1, C2 says I am definitely 0, and I already think I am 0, so my combined belief is..."**
4. Decision: Check the final LLR sign. If negative, declare the bit flipped.
5. Syndrome check: Check whether the current correction satisfies all parity checks. If yes, stop; otherwise return to step 2 (until timeout).

One-sentence summary of BP: Everyone (qubits) keeps updating their belief based on neighbors (checks) until the network reaches consensus.

11. Surface Code

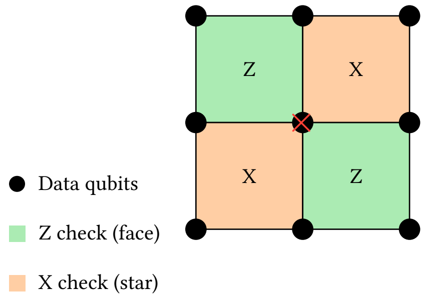
The surface code is currently the most promising candidate for universal quantum computing. It is a typical topological code.

11.1 Structure and diagram

Imagine a chessboard:

- Data qubits: located at the intersections (or edges) of the grid.
- Stabilizers: located in the plaquettes. Two types: one checks Z errors (face operators), the other checks X errors (vertex/star operators).

Surface Code (Rotated Surface Code) diagram



Error detection mechanism: The central data qubit has an error (red cross), which triggers both adjacent X and Z checks. These paired "hot spots" are the syndrome.

11.2 Key properties

1. Locality: the biggest engineering advantage. Each qubit only interacts with its neighbors. No need to connect the upper-left qubit to the lower-right.
2. High threshold: about 1% physical error rate still correctable. Very forgiving.
3. Zero code rate: the biggest downside.
   - No matter how large the board is, e.g., 1000 × 1000 physical qubits, it typically encodes one logical qubit.
   - Resource overhead is huge.

11.3 Why does BP perform poorly on the surface code?

From the diagram, each plaquette (check) shares data qubits with neighboring plaquettes. On the Tanner graph, this forms many length-4 short cycles.

- Standard BP cannot converge correctly in such dense short-cycle structures (positive feedback oscillation).
- Therefore surface codes typically use MWPM (minimum weight perfect matching) or Union-Find decoders.

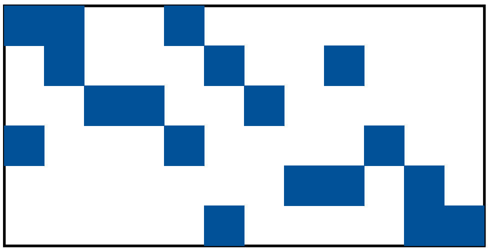
12. LDPC codes (Low-Density Parity-Check Codes)

LDPC codes generalize surface codes. In classical communications (e.g., 5G, Wi-Fi) they are already standard. Quantum LDPC (qLDPC) is a hot research topic in recent years.

12.1 What does "low density" mean?

"Low density" means the parity-check matrix H is sparse.

LDPC sparse matrix H diagram



Features:

1. Most cells are empty (white).
2. Each row/column has only a few non-zero elements (blue).
3. This means each check involves only a few bits, and each bit participates in only a few checks.

- Row weight: number of bits involved in each check (usually a small constant like 6).
- Column weight: number of checks each bit participates in (also a small constant like 3 or 4).
- Non-locality: unlike the surface code, LDPC codes allow long-range connections. Points in the matrix can be far apart, not just neighbors.

12.2 Why are qLDPC codes so exciting?

They solve the biggest pain of surface codes - code rate.

Performance comparison

- **Surface code**: uses N physical qubits to encode 1 logical qubit. $\frac{k}{N} \to 0$.
- **Good qLDPC codes**: use N physical qubits to encode $k = \frac{N}{10}$ logical qubits. $\frac{k}{N} = \text{const}$.

Example: to protect 100 logical qubits.

- Surface code may need 100,000 physical qubits.

> - qLDPC may need 1,000 physical qubits.

12.3 Decoding: BP's home field

Because qLDPC codes are typically constructed from expander graphs, they can still have cycles, but the cycles are usually long (large girth) or sufficiently random.

- Standard BP performs well on LDPC codes.
- BP + OSD is the standard decoding approach for qLDPC.

12.4 Summary comparison table

Feature	**Surface Code**	**Quantum LDPC code**
Geometry	2D planar grid, local connections	Complex network, long-range connections
Short-cycle problem	Very severe (many 4-cycles)	Milder (designed to avoid short cycles)
Encoding efficiency	Very low ($k = 1$)	High (finite code rate)
Applicable decoders	MWPM, Union-Find (BP needs modifications)	BP, BP+OSD
Engineering difficulty	Low (simple wiring)	High (requires long wires/multi-layer routing)

13. BP algorithm deep dive

Belief Propagation (BP) is the standard algorithm for exact inference on trees or approximate inference on graphs with cycles. In QEC, we often use log-domain BP for numerical stability.

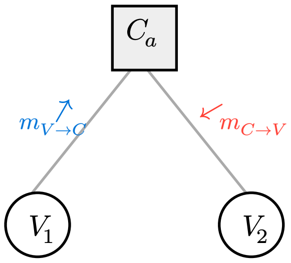
13.1 Why use log-likelihood ratios (LLR)?

Directly passing probabilities $p \in [0, 1]$ can underflow to 0 after repeated multiplications. We define LLR:

$$L(x) = \ln\left(\frac{P(x = 0)}{P(x = 1)}\right)$$

- $L > 0$: more likely 0 (no error).
- $L < 0$: more likely 1 (error).
- $|L|$: confidence magnitude.

Message passing on a Tanner graph



Core principle (Extrinsic Principle):
The message V_2 sends to C_a cannot include the information that C_a just sent to V_2.

Plainly:
"Apart from what you told me, here is what I think..."

13.2 Core update formulas

The algorithm alternates message passing between variable nodes V_i and check nodes C_j.

1. Variable to Check ($V \to C$) V_i tells C_j the probability of being 0 or 1. This equals its initial observation plus all other check nodes (except C_j) suggestions.

$$m_{i\to j} = L_i^{\text{init}} + \sum_{k\in N(i)\setminus j} m_{k\to i}$$

2. Check to Variable ($C \to V$) C_j tells V_i what it must be to satisfy parity. This involves a nonlinear "tanh" operation (a soft version of XOR).

$$m_{j\to i} = 2\tanh^{-1}\left((-1)^{S_j} \prod_{k\in N(j)\setminus i} \tanh\left(\frac{m_{k\to j}}{2}\right)\right)$$

- $S_j \in \{0, 1\}$ is the syndrome value of the check node.
- If $S_j = 1$ (check fails), the sign term $(-1)^1 = -1$ flips the message sign, encouraging a flip.
- If some $m_{\{ktoj\}}$ is near 0 (uncertain), the product is near 0, and C_j gives a near-zero suggestion.

13.3 Code logic (Pythonic pseudocode)

In practice (C++ or Python), the code structure is typically:

```
# Initialization
# channel_probs: physical error rate per bit
llr = log((1 - channel_probs) / channel_probs) # initial LLR
check_to_var_msg = zeros(num_checks, num_vars) # store C->V messages

for iter in range(max_iters):

    # --- Step 1: Variable node processing (V -> C) ---
    # For each edge (i, j), compute the message from V_i to C_j
    # Trick: compute total once, then subtract the message from j (O(deg) -> O(1))
    var_to_check_msg = zeros(num_checks, num_vars)
    for i in range(num_vars):
        incoming_sum = llr[i] + sum(check_to_var_msg[:, i]) # includes all C info
        for j in neighbors(i):
            # Extrinsic principle: total minus the message from j
            var_to_check_msg[j, i] = incoming_sum - check_to_var_msg[j, i]

    # --- Step 2: Check node processing (C -> V) ---
    # Core challenge: implement the tanh product formula
    # Optimization: usually use lookup tables or min-sum approximation
    for j in range(num_checks):
        syndrome_sign = -1 if syndrome[j] == 1 else 1

        # Compute tanh product over all variables connected to this check
        prod_tanh = 1.0
        for i in neighbors(j):
            prod_tanh *= tanh(var_to_check_msg[j, i] / 2)

        # Update C -> V messages
        for i in neighbors(j):
            # Extrinsic principle: divide by current variable's tanh (multiplicative
inverse)
            # Note division by 0 (numerical stability)
            extrinsic_val = prod_tanh / tanh(var_to_check_msg[j, i] / 2)
            extrinsic_val = clip(extrinsic_val, -0.999999, 0.999999)
            check_to_var_msg[j, i] = syndrome_sign * 2 * atanh(extrinsic_val)

    # --- Step 3: Soft decision and check ---
    new_llr = llr.copy()
    for i in range(num_vars):
        new_llr[i] += sum(check_to_var_msg[:, i])

    hard_decision = (new_llr < 0).astype(int) # LLR < 0 -> 1 (error)

    if (H @ hard_decision % 2 == syndrome).all():
        return hard_decision # converged

return fail # timeout
```