Master's Thesis (Academic Year 2025)

Implementation and Resource Estimation of Decoded Quantum Interferometry

Keio University
Graduate School of Media and Governance

Natchapol Patamawisut

Master's Thesis　Academic Year 2025

Implementation and Resource Estimation of Decoded Quantum Interferometry

Summary

In this thesis, I investigate the quantum optimization algorithm Decoded Quantum Interferometry (DQI), which prepares a quantum state whose amplitudes encode a polynomial representation of the objective function, allowing us to sample high-quality solutions from measurement outcomes. Existing work on DQI has been largely theoretical, without a concrete circuit-level blueprint or systematic implementation. I move DQI toward implementation by specifying its realization as a quantum circuit, evaluating its behavior in simulation, and studying how its resource requirements scale with problem size.

To this end, I construct explicit circuit families for DQI by decomposing the high-level algorithm into elementary operations. The central challenging part of implementing this algorithm is the decoder: I implement and compare several realizations, including a Gauss Jordan Elimination (GJE) decoder, a lookup-table-based decoder, and a Belief Propagation with Quantum Messages (BPQM) decoder, and analyze how each choice affects the number of qubits, the circuit depth, and the counts of each gate type. Building on these designs, I perform detailed resource estimates and evaluate the algorithm on simulators. The implementation is released as open-source code to support further exploration and benchmarking of DQI.

These circuit constructions and resource estimates identify the main implementation bottlenecks in DQI and quantify how decoder choices contribute to overheads in qubits, gates, and depth. I also outline directions for future work, including exploring additional decoder families, studying larger problem instances and their scaling behavior, and extending these simulations to hardware-aware analyses and experiments on real quantum devices.

Keywords:

Decoded Quantum Interferometry, Quantum Algorithms, Optimization Algorithms, Implementation, Resource Estimation

Keio University Graduate School of Media and Governance

Natchapol Patamawisut

PUBLISHED CONTENT

N. Patamawisut, N. Benchasattabuse, M. Hajdušek, and R. Van Meter, "Quantum Circuit Design for Decoded Quantum Interferometry," in *Proc. 2025 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE Quantum Week), Albuquerque, NM, USA, Aug. 31–Sep. 5, 2025, pp. 291–301, doi: 10.1109/QCE65121.2025.00041. IEEE Xplore: 11250226.

Award: 2nd Place – QALG Best Paper Awards.

N.P. conceived the project; designed the end-to-end DQI circuits; performed resource estimation and simulations; coordinated experiments; and led the writing of the manuscript.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization, broadly understood (and often formalized as operations research), has been studied for decades because of its pervasive usefulness, from logistics and transportation to finance and engineering design. A key challenge in many optimization problems is that they are inherently hard to parallelize: even in today's high-compute era, where large-scale parallel hardware greatly accelerates tasks such as machine learning, many optimization procedures remain fundamentally sequential or suffer from poor scaling with problem size.

Quantum computing offers a potential way to overcome some of these limitations by exploiting computational resources that are not available in classical models. Quantum optimization aims to use quantum devices to tackle hard optimization problems more effectively than classical algorithms. Until recently, the main algorithmic frameworks in this area have been variational or adiabatic in nature, with the Quantum Approximate Optimization Algorithm (QAOA) [1] and related short-path algorithms [2] serving as prominent examples that have shown promising performance and, in some regimes, indications of quantum advantage.

More recently, Google Quantum AI introduced a new algorithm called Decoded Quantum Interferometry (DQI) [3], which takes a different route to quantum optimization. Rather than directly encoding the cost function into a variational ansatz or an adiabatic Hamiltonian, DQI first uses a Quantum Fourier Transform (QFT) to reduce the optimization instance to a bounded-distance decoding problem over an appropriate code. A decoder unitary is then applied to steer amplitude toward the target state specified by the objective function. This framework provides a qualitatively new route to quantum advantage for optimization, explicitly linking quantum algorithms with ideas from classical error correction and decoding.

1.1 Motivation

Since its introduction, work on DQI has been mostly theoretical [3, 4], focusing on its formal structure and asymptotic guarantees. Existing papers describe the algorithm at a high level and outline its relation to error-correcting codes, but they do not provide an explicit, end-to-end circuit implementation or a detailed resource study. In particular, the decoding step—solving the associated bounded-distance decoding problem—is treated abstractly, without a comparison of different decoding strategies at the circuit level.

The motivation of this thesis is to close this gap by moving DQI from a purely theoretical proposal toward an implementation-oriented perspective. I start from the fundamentals of quantum computation and quantum optimization, introduce the basic ideas behind DQI and its connection to error correction and bounded-distance decoding, and then ask: How can the decoding problem be solved not only classically but also as a quantum circuit? Which decoder realizations are the most resource-efficient, and what trade-offs do they introduce? How can the full DQI pipeline be implemented step by step, and how does the cost of each stage scale with problem size?

Answering these questions requires bringing together tools from classical coding theory, decoder design, and quantum circuit synthesis. The goal of this thesis is to make this connection explicit, quantify the associated resource costs, and thereby provide a clearer picture of what it would take to implement DQI in practice.

1.2 Contributions

This thesis makes the following contributions toward an implementation-oriented understanding of Decoded Quantum Interferometry (DQI):

1. **End-to-end DQI circuit design.** I specify an explicit realization of DQI.
2. **Full circuit implementations.** I implement the complete DQI pipeline as modular circuit families to enable consistent comparisons across designs.
3. **Resource estimation.** I derive qubit counts, circuit depth, and per-gate-type count equations for each DQI stage.
4. **Simulation benchmarks.** I evaluate the implemented circuits on classical simulators and provide a small, reproducible benchmark suite.
5. **Open-source release.** I release a unified codebase to facilitate reproduction and future comparisons.
6. **Bottlenecks and design guidance.** I identify key implementation bottlenecks and distill practical guidance for future improvements.

Chapter 2

Background

2.1 Quantum Computing

QQuantum technology has been part of everyday life for decades, often invisibly—for example, lasers power barcode scanners, fiber-optic links, and optical drives. Magnetic-resonance imaging (MRI) exploits the nuclear magnetic resonance of atomic nuclei to produce detailed, non-ionizing medical images. Global navigation systems (GPS) depend on atomic clocks, whose quantized energy-level transitions keep time so precisely that satellite positioning and even financial networks can be synchronized. The entire semiconductor ecosystem—from transistors and integrated circuits to LEDs—rests on quantum band theory, which explains how electrons occupy energy bands in solids [5, 6, 7].

These examples are often described as the first quantum revolution: technologies that crucially rely on quantum mechanics but do not require fine-grained control of individual quantum systems. The second quantum revolution [8] is characterized by the ability to coherently prepare, manipulate, and measure individual quantum states to unlock new capabilities, including quantum sensing (e.g., NV-center magnetometry), quantum communication (e.g., QKD), and quantum computing [9, 10].

Quantum computing is one of the new quantum technologies attracting intense interest as conventional CMOS scaling encounters physical and economic limits. Moore's law—the historical observation [11] that transistor counts double roughly every two years—has slowed as Dennard scaling [12](which once kept power density constant while shrinking transistors) broke down in the mid-2000s, exposing power and thermal walls and increasing leakage and other short-channel quantum effects at ever-smaller nodes. As a result, further gains from simply shrinking transistors have become harder, motivating exploration of new computing paradigms, including quantum computing, that exploit phenomena unavailable to classical devices.

Importantly, quantum computing is not simply a "faster" or "better" version of classical computing; it is a fundamentally different model of computation based on the principles of quantum mechanics. In classical computing, information is stored in bits that take values 0 or 1, and processors manipulate these bits using logical operations. In quantum computing, information is stored in quantum bits, or qubits, which can exist in superpositions of 0 and 1, and collections of qubits can become entangled, leading to correlations with no classical counterpart.

From this perspective, quantum computing is the study of how to build and program quantum processors—quantum processing units (QPUs)—that exploit quantum properties to perform computations in ways that classical processors cannot efficiently emulate. A QPU is a physical device (for example, based on superconducting circuits, trapped ions, or spins in semiconductors) whose quantum states encode and process information according to the laws of quantum mechanics, rather than classical transistor logic.

Because the underlying hardware and physics are different, the algorithms must also be fundamentally different. Quantum computing is not "just using better algorithms on a classical CPU," and in general one cannot take a quantum algorithm and run it on a classical machine while retaining its advantages. Instead, one requires quantum algorithms—such as Shor's factoring algorithm [13] or Grover's search algorithm [14]—that are specifically designed to exploit quantum effects like interference to amplify correct answers and suppress incorrect ones.

In this section, I briefly review the basic mathematical framework and notation underlying quantum computation [15]. I focus on the description of quantum states, their evolution, and measurement; specific computational models will be introduced in a later subsection.

2.1.1 Hilbert Spaces and Dirac Notation

Quantum states are described by vectors in a complex Hilbert space $\mathcal{H}$, which is a complex vector space equipped with an inner product that is linear in the second argument and conjugate-linear in the first.

Dirac's notation is used throughout this thesis. A *ket* $|\psi\rangle$ denotes a column vector in $\mathcal{H}$, while a *bra* $\langle\phi|$ denotes the conjugate transpose of $|\phi\rangle$, i.e.,

$$\langle\phi| \equiv |\phi\rangle^\dagger . \tag{2.1}$$

The inner product between two states $|\phi\rangle$ and $|\psi\rangle$ is written as

$$\langle\phi|\psi\rangle \in \mathbb{C}$$

and induces the norm $\| \, |\psi\rangle \, \| = \sqrt{\langle\psi|\psi\rangle}$. Physical pure states are represented by unit vectors, i.e., $\langle\psi|\psi\rangle = 1$.

Given two states $|\phi\rangle$ and $|\psi\rangle$, the *outer product*

$$|\phi\rangle\langle\psi|$$

denotes the linear operator $\mathcal{H} \to \mathcal{H}$ that maps $|\xi\rangle \mapsto |\phi\rangle \langle\psi|\xi\rangle$. In particular, $|\psi\rangle\langle\psi|$ is the rank-one projector onto the span of $|\psi\rangle$.

2.1.2 Qubits and Quantum States

The basic unit of information in quantum computing is the *quantum bit*, or *qubit*. Mathematically, a single qubit is represented by a unit vector in a two-dimensional complex Hilbert space $\mathcal{H} \cong \mathbb{C}^2$. The standard computational basis is

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

representing a qubit in the states $|0\rangle$ and $|1\rangle$, respectively. These are the most common basis states and can be viewed as the quantum analogues of the classical bit values 0 and 1. In contrast to a classical bit, however, a qubit is not restricted to just these two states: a general pure state of a single qubit is a superposition

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \qquad \alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1. \tag{2.2}$$

When both α and β are nonzero, the qubit is said to be in a superposition of $|0\rangle$ and $|1\rangle$, meaning that measurements in the computational basis yield 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$.

Up to an overall global phase, any single-qubit state can be represented as a point on the Bloch sphere:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle\,. \tag{2.3}$$

for real parameters $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$.

Here, the angle θ controls the relative weights of $|0\rangle$ and $|1\rangle$, while ϕ controls the *relative phase* between them. A global phase factor $e^{i\gamma}$ multiplying the entire state,

$$|\psi\rangle \sim e^{i\gamma}\,|\psi\rangle\,. \tag{2.4}$$

is physically irrelevant because it leaves all measurement probabilities and expectation values unchanged. In contrast, the phase between the components, encoded in $e^{i\phi}$, is *physically meaningful*: it affects interference and can change the outcome statistics of subsequent operations.

2.1.3 Composite Systems and Tensor Products

When we combine multiple quantum systems, their joint state space is given by the *tensor product* of the individual Hilbert spaces. Given two Hilbert spaces $\mathcal{H}_A$ and $\mathcal{H}_B$, their tensor product $\mathcal{H}_A \otimes \mathcal{H}_B$ is a Hilbert space spanned by formal linear combinations of vectors of the form $|\phi\rangle_A \otimes |\psi\rangle_B$, where $|\phi\rangle_A \in \mathcal{H}_A$ and $|\psi\rangle_B \in \mathcal{H}_B$, subject to the usual bilinearity rules. If $\{|a_i\rangle\}$ is an orthonormal basis for $\mathcal{H}_A$ and $\{|b_j\rangle\}$ is an orthonormal basis for $\mathcal{H}_B$, then $\{|a_i\rangle \otimes |b_j\rangle\}_{i,j}$ is an orthonormal basis for $\mathcal{H}_A \otimes \mathcal{H}_B$.

In the context of qubits, a two-qubit system is described by $\mathbb{C}^2 \otimes \mathbb{C}^2 \cong \mathbb{C}^4$. The computational basis is

$$|00\rangle = |0\rangle \otimes |0\rangle, \quad |01\rangle = |0\rangle \otimes |1\rangle, \quad |10\rangle = |1\rangle \otimes |0\rangle, \quad |11\rangle = |1\rangle \otimes |1\rangle.$$

A general two-qubit pure state can then be written as

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle, \qquad \sum_{x\in\{0,1\}^2} |\alpha_x|^2 = 1.$$

More generally, an n-qubit register is described by the tensor product of n single-qubit spaces,

$$\mathcal{H}_n = \mathcal{H}^{\otimes n} \cong (\mathbb{C}^2)^{\otimes n}. \tag{2.5}$$

The computational basis states are written as

$$|x_1 x_2 \dots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle,$$

where each $x_j \in \{0,1\}$. A general pure state of n qubits can be expressed as

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle, \qquad \sum_x |\alpha_x|^2 = 1. \tag{2.6}$$

For brevity, I will often omit the explicit $\otimes$ symbol and write, for example, $|a\rangle|b\rangle$ instead of $|a\rangle \otimes |b\rangle$, and similarly for operators.

2.1.4 Unitary Evolution and Quantum Gates

To use quantum systems for information processing, we must be able to *transform* their states in a controlled way. In the simplest picture, a pure state of a single qubit is a column vector in $\mathbb{C}^2$, and a quantum gate acting on that qubit is represented by a 2×2 matrix that multiplies this vector.

For example, the Pauli-X gate is the matrix

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

and its action on the computational basis states is

$$X\left|0\right\rangle = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}0\\ 1\end{bmatrix} = \left|1\right\rangle, \qquad X\left|1\right\rangle = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \begin{bmatrix}1\\ 0\end{bmatrix} = \left|0\right\rangle.$$

More generally, for a superposition $\left|\psi\right\rangle = \alpha\left|0\right\rangle + \beta\left|1\right\rangle$, we have

$$X\left|\psi\right\rangle = \alpha X\left|0\right\rangle + \beta X\left|1\right\rangle = \alpha\left|1\right\rangle + \beta\left|0\right\rangle,$$

so the X gate simply swaps the amplitudes of $\left|0\right\rangle$ and $\left|1\right\rangle$. In this way, quantum gates act as matrices that map input state vectors to output state vectors.

In the general formalism, the time evolution of a closed, noiseless quantum system is described by a *unitary* operator U acting on the state:

$$\left|\psi'\right\rangle = U\left|\psi\right\rangle. \tag{2.7}$$

Here $U^\dagger$ denotes the adjoint (conjugate transpose) of U. An operator U on a Hilbert space $\mathcal{H}$ is called *unitary* if

$$U^\dagger U = UU^\dagger = I. \tag{2.8}$$

where I is the identity operator on $\mathcal{H}$.

Single-qubit Rotation Operators

A particularly important class of single-qubit unitaries are *rotation operators* on the Bloch sphere. Using the Pauli matrices

$$X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, \quad Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}, \quad Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix},$$

we define rotations about the x-, y-, and z-axes by

$$R_x(\theta) = e^{-i\theta X/2}, \qquad R_y(\theta) = e^{-i\theta Y/2}, \qquad R_z(\theta) = e^{-i\theta Z/2}. \tag{2.9}$$

for a real rotation angle θ. These are 2×2 unitary matrices, and up to a global phase any single-qubit unitary can be written as a product of such rotations.

As a simple example, the $R_y(\theta)$ gate acts on $\left|0\right\rangle$ as

$$R_y(\theta)\left|0\right\rangle = \cos\left(\frac{\theta}{2}\right)\left|0\right\rangle + \sin\left(\frac{\theta}{2}\right)\left|1\right\rangle,$$

creating a controllable superposition between $\left|0\right\rangle$ and $\left|1\right\rangle$. Thus rotation operators provide a convenient way to move points on the Bloch sphere by specified angles.

Multi-qubit Unitary Operators

For a register of n qubits, the state space is the tensor product $\mathcal{H}_n = \mathcal{H}^{\otimes n} \cong (\mathbb{C}^2)^{\otimes n}$, and quantum gates are represented by $2^n \times 2^n$ unitary matrices acting on this space. The simplest multi-qubit unitaries are tensor products of single-qubit gates. For example, applying H to the first qubit and X to the second qubit of a two-qubit register corresponds to the operator

$$H \otimes X,$$

which acts as

$$(H \otimes X)\big(|a\rangle\,|b\rangle\big) = (H\,|a\rangle) \otimes (X\,|b\rangle) \quad \text{for } a, b \in \{0, 1\}.$$

More interesting behaviour arises from *entangling* multi-qubit gates, which cannot be written as simple tensor products. A standard example is the controlled-NOT (CNOT) gate, which flips the target qubit if and only if the control qubit is in state $|1\rangle$:

$$\text{CNOT}\,|00\rangle = |00\rangle\,, \quad \text{CNOT}\,|01\rangle = |01\rangle\,, \quad \text{CNOT}\,|10\rangle = |11\rangle\,, \quad \text{CNOT}\,|11\rangle = |10\rangle\,.$$

Starting from $|00\rangle$, the circuit

$$(\text{CNOT})(H \otimes I)\,|00\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big)$$

prepares a maximally entangled Bell state. In this way, combinations of single-qubit rotations and multi-qubit entangling gates generate the unitary evolutions used in quantum algorithms.

2.1.5 Measurement

Quantum measurements are modeled by a collection of measurement operators $\{M_m\}$ acting on the state space and satisfying

$$\sum_m M_m^\dagger M_m = I. \tag{2.10}$$

Given a pre-measurement state ρ, the probability of obtaining outcome m is

$$p(m) = \mathrm{Tr}\big(M_m^\dagger M_m\,\rho\big). \tag{2.11}$$

and the post-measurement state, conditioned on observing outcome m, is

$$\rho_m = \frac{M_m \rho M_m^\dagger}{p(m)}. \tag{2.12}$$

This is the most general description of a measurement compatible with quantum mechanics (a POVM implemented via measurement operators).

In many computational settings, we restrict attention to projective measurements in the computational basis. For a single qubit, this corresponds to the projectors

$$P_0 = |0\rangle\langle 0| , \qquad P_1 = |1\rangle\langle 1| . \tag{2.13}$$

which satisfy $P_0 + P_1 = I$ and $P_j^2 = P_j$. For a pure state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, the Born rule gives

$$p(0) = \langle\psi|P_0|\psi\rangle = |\alpha|^2, \qquad p(1) = \langle\psi|P_1|\psi\rangle = |\beta|^2.$$

so outcome 0 occurs with probability $|\alpha|^2$ and outcome 1 with probability $|\beta|^2$. The corresponding post-measurement states are $|0\rangle$ and $|1\rangle$, respectively.

2.1.6 Quantum Circuits

The mathematical ingredients introduced above—Hilbert spaces, tensor products, unitary evolution, and measurement—are organized in practice into several equivalent models of quantum computation. The most widely used is the *gate-based* or *circuit* model, in which a computation is described as a sequence of quantum gates acting on qubits, followed by measurements [15].

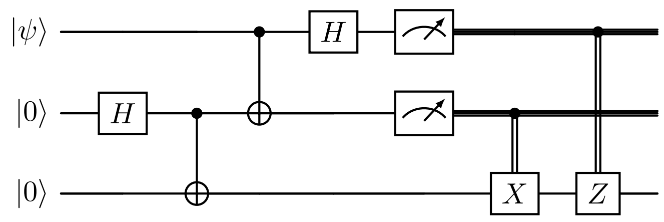


Figure 2.1: Example of a quantum circuit in the gate-based model. Horizontal lines represent qubits, with time flowing from left to right. Single-qubit gates are drawn as boxes on a wire, while two-qubit controlled gates are shown by a filled control dot connected to a $\oplus$ symbol on the target wire. Meter symbols denote projective measurements in the computational basis, and double wires indicate classical bits carrying the measurement outcomes that can be used to classically control later quantum gates.

Other models that are computationally equivalent (up to polynomial overhead) include measurement-based quantum computation [16], and adiabatic quantum computation [17], but throughout this thesis I will express all constructions in the gate-based circuit model.

In the circuit model, an n-qubit computation is visualized as a set of horizontal lines, one for each qubit, with time flowing from left to right. Gates are drawn as symbols placed on these lines. An example is shown in Figure 2.1, which depicts the standard *quantum teleportation* circuit. The top wire carries the unknown state $|\psi\rangle$ to be teleported, while the two lower wires start in $|0\rangle$. A Hadamard gate followed by a controlled-NOT (CNOT) on the lower two qubits prepares an entangled Bell pair. The first two qubits are then processed by a CNOT (control on the top wire, target on the middle wire) and a Hadamard on the top wire, implementing a Bell-basis measurement. Meter symbols on the first two wires indicate projective measurements in the computational basis, and the resulting classical bits are propagated along double wires to control Pauli X and Z gates on the bottom qubit. Regardless of the measurement outcomes, the net effect of the circuit is to reproduce the input state $|\psi\rangle$ on the bottom wire, while the measured qubits end in classical states.

More generally, a circuit on n qubits with T time steps consists of a sequence of elementary gates $U_1, U_2, \dots, U_T$ drawn from a fixed universal gate set. The overall evolution (before measurement) is given by the unitary operator

$$U = U_T \cdots U_2 U_1. \tag{2.14}$$

acting on some initial state (typically $|0\rangle^{\otimes n}$), followed by measurements of a subset of the qubits in a chosen basis. The resulting classical bit string is interpreted as the output of the quantum algorithm.

2.2 Error-Correcting Codes

Physical information carriers are inevitably subject to noise. In a classical digital system, transmitted bits may be flipped by thermal fluctuations, imperfect hardware, crosstalk between wires, or electromagnetic interference on a communication channel. Similar effects occur in storage media, where charge leakage, defects, or aging can corrupt stored data over time. Rather than trying to build perfectly noiseless hardware, modern communication and storage systems rely on *error control*: the sender deliberately adds structured redundancy so that the receiver can detect and, within limits, correct errors introduced by the channel [18].

Error-correcting codes provide a systematic way to implement this idea. A message is first encoded into a longer codeword that satisfies a set of parity constraints; after transmission through a noisy channel, the receiver uses these constraints to infer which symbols were likely corrupted and to reconstruct the original information. In this section, I focus exclusively on *classical* error-correcting codes and decoding, with a particular emphasis on block codes and on linear codes defined by sparse parity-check matrices, such as low-density parity-check (LDPC) codes [19]. These notions are purely classical, but they will later serve as building blocks for more advanced constructions and for the decoders that appear inside the quantum algorithms studied in this thesis.

2.2.1 Block Codes and Hamming Distance

In classical error-control coding, we consider a simple communication model with three main components: an *encoder*, a *noisy channel*, and a *decoder* [20]. A sender wishes to transmit a finite-length binary message

$$m \in \{0,1\}^k$$

over a channel that may flip some of the bits. Instead of sending m directly, the encoder maps it to a longer binary string $c \in \{0,1\}^n$ that contains redundant information. This redundancy allows the receiver to detect and, within limits, correct errors introduced by the channel.

A (binary) *block code* of length n is a specified set

$$C \subseteq \{0,1\}^n$$

of allowed length-n binary strings, called *codewords.* The encoder is a mapping

$$\mathrm{Enc} : \{0,1\}^k \to C,$$

so that each k-bit message m is associated with a unique codeword $c = \mathrm{Enc}(m)$. During transmission, the channel corrupts some positions in c, and the receiver observes a *received word* $r \in \{0,1\}^n$, which may differ from c. The decoder then applies a mapping

$$\mathrm{Dec} : \{0,1\}^n \to \{0,1\}^k$$

(or equivalently $\mathrm{Dec} : \{0,1\}^n \to C$) that attempts to reconstruct the original message or codeword from r.

To reason about the robustness of a code, we need a way to quantify how different two length-n binary strings are. For $x, y \in \{0,1\}^n$, the *Hamming distance* $d(x,y)$ is defined as the number of positions in which they differ:

$$d(x,y) := \big|\{\, i \in \{1,\dots,n\} : x_i \neq y_i \,\}\big|. \tag{2.15}$$

Closely related is the *Hamming weight* of a vector x, defined as $w(x) = d(x,0)$, i.e., the number of 1's in x.

The *minimum distance* of a block code C is

$$d_{\min}(C) := \min_{\substack{x,y\in C \\ x\neq y}} d(x,y). \tag{2.16}$$

the smallest Hamming distance between any two distinct codewords. This single parameter captures the basic error-detecting and error-correcting capability of the code:

1. A code with minimum distance $d_{\min}$ can *detect* any pattern of up to $d_{\min} - 1$ bit-flip errors in a block, because such an error cannot transform one valid codeword into a different valid codeword.

2. In principle, the same code can *correct* up to

$$t := \left\lfloor \frac{d_{\min}(C) - 1}{2} \right\rfloor . \tag{2.17}$$

   bit-flip errors, for example by *nearest-neighbor decoding*: given a received word r, the decoder chooses the codeword $c \in C$ that is closest to r in Hamming distance and outputs c as its estimate of the transmitted word.

In many practical systems, the block code C is chosen to have additional algebraic structure (typically a linear subspace of $\mathbb{F}_2^n$). This allows compact descriptions via generator and parity-check matrices and enables efficient decoding algorithms, which I discuss in the next subsection.

2.2.2 Linear Block Codes

In many practical systems it is advantageous to restrict attention to block codes with additional algebraic structure. A (binary) *linear block code* of length n and dimension k, usually denoted by an $[n, k]$ code, is a k-dimensional linear subspace

$$C \subseteq \mathbb{F}_2^n,$$

where $\mathbb{F}_2 = \{0, 1\}$ is the field with two elements and $\mathbb{F}_2^n$ denotes the set of all length-n binary vectors viewed as a vector space over $\mathbb{F}_2$. Linearity means that the all-zero vector 0^n is a codeword and that C is closed under addition (bitwise XOR): if $c_1, c_2 \in C$, then $c_1 + c_2 \in C$. The parameter k is called the *dimension* of the code, and the ratio

$$R := \frac{k}{n}. \tag{2.18}$$

is the *code rate*, measuring how much redundancy is introduced by the encoding.

A linear code can be described compactly by a *generator matrix* G, which is a $k \times n$ matrix over $\mathbb{F}_2$ whose rows form a basis of C. The encoding operation is then a linear map

$$\mathrm{Enc} : \mathbb{F}_2^k \to \mathbb{F}_2^n, \qquad \mathrm{Enc}(m) = mG,$$

so that each k-bit message $m \in \mathbb{F}_2^k$ is mapped to a length-n codeword $c = mG \in C$.

Equivalently, a linear code can be specified by a *parity-check matrix* H, which is an $(n - k) \times n$ matrix over $\mathbb{F}_2$. The code is then given as the null space of H:

$$C := \{\, c \in \mathbb{F}_2^n : Hc^{\mathsf{T}} = 0 \,\}. \tag{2.19}$$

The generator and parity-check matrices are related by

$$HG^{\mathsf{T}} = 0. \tag{2.20}$$

expressing the fact that every encoded word $c = mG$ automatically satisfies all parity constraints imposed by H.

2.2.3 Syndromes and Decoding

When a codeword $c \in C$ is transmitted over a noisy channel, the receiver observes

$$r = c + e. \tag{2.21}$$

where $e \in \mathbb{F}_2^n$ is the (unknown) error pattern and addition is bitwise modulo 2. The goal of decoding is to use r to reconstruct the original codeword c (and hence the original message), or at least to output an estimate $\hat{c}$ that equals c with high probability. Equivalently, one can view decoding as trying to infer the most likely error pattern e consistent with r and then subtract it. For a linear code with parity-check matrix H, each row of H represents a parity constraint (check equation) that every valid codeword must satisfy. Concretely, if the i-th row of H has entries $H_{i1}, \ldots, H_{in}$, then a vector $c \in \mathbb{F}_2^n$ is a valid codeword only if

$$\sum_{j=1}^{n} H_{ij} c_j = 0 \pmod 2$$

for every row i. In this sense, H encodes all the parity checks that define the code.

Given a received word r, we can test these parity constraints by multiplying r by H. The *syndrome* of r is defined as

$$s := Hr^{\mathsf{T}}. \tag{2.22}$$

where the i-th component s_i is the parity (mod 2) of the bits of r involved in the i-th check equation. If $s_i = 0$, then that check is satisfied by r; if $s_i = 1$, the check is violated. Thus the syndrome is a compact summary of which parity checks pass and which fail for the received word.

Using $r = c + e$ and $Hc^{\mathsf{T}} = 0$ for all $c \in C$, we have

$$\begin{aligned} s &= H(c+e)^{\mathsf{T}} \\ &= Hc^{\mathsf{T}} + He^{\mathsf{T}} \\ &= He^{\mathsf{T}}. \end{aligned} \tag{2.23}$$

In other words, the syndrome depends only on the error pattern e and is independent of the unknown transmitted codeword c. Different error patterns may sometimes share the same syndrome, but any two received words with the same syndrome violate exactly the same set of parity checks. From the decoder's point of view, the syndrome is therefore the "signature" of the error as seen through the parity-check constraints: it tells us how the received word fails to look like a codeword, and decoding amounts to choosing a plausible error pattern consistent with that signature.

The purpose of introducing the syndrome is to separate the influence of the channel errors from the unknown codeword and to reduce the decoding problem to an *error-estimation* problem. Given r and $s = Hr^{\mathsf{T}}$, the decoder seeks an error pattern $\hat{e}$ such that $H\hat{e}^{\mathsf{T}} = s$ and then forms

$$\hat{c} = r + \hat{e}. \tag{2.24}$$

which is its estimate of the transmitted codeword. Different error patterns that produce the same syndrome s are indistinguishable at the level of parity checks; a decoding rule chooses one of them (typically the most likely under the assumed channel model).

A simple conceptual decoding approach is a lookup table. In its idealized form, one precomputes a table that maps each possible syndrome s to a "preferred" error pattern $\hat{e}(s)$ (for example, the lowest-weight error consistent with s under the assumed channel model). Upon receiving r, the decoder computes $s = Hr^{\mathsf{T}}$, looks up $\hat{e}(s)$, and outputs $\hat{c} = r+\hat{e}(s)$ as its estimate of the transmitted codeword. For codes with large block length n, storing a full syndrome table is impractical and exact nearest-neighbor decoding is computationally expensive, so one typically uses more structured decoding algorithms.

2.3 Low-Density Parity-Check (LDPC) Codes

Low-density parity-check (LDPC) codes form an important family of binary linear block codes [19]. LDPC codes are now widely used in modern communication and storage systems because, together with iterative decoding algorithms, they can achieve performance close to channel capacity for large block lengths [21]. More recently, LDPC ideas have also become central in *quantum* error correction: many leading quantum code constructions are quantum LDPC codes, where the sparsity of the (classical) parity-check matrix translates into low-weight stabilizer checks that are easier to implement on noisy quantum hardware.

2.3.1 Definition via Sparse Parity-Check Matrices

Let H be an $(n-k) \times n$ parity-check matrix over $\mathbb{F}_2$. The associated linear code is

$$C \;=\; \{\, c \in \mathbb{F}_2^n \;:\; Hc^{\mathsf{T}} = 0 \,\}. \tag{2.25}$$

The code C is called a *low-density parity-check* (LDPC) code if the matrix H is *sparse*:

the number of nonzero entries in H scales only linearly with n, and each row and column of H has relatively small Hamming weight compared to n. Intuitively, each parity-check equation involves only a few codeword bits, and each codeword bit participates in only a few parity checks.

This sparsity is the defining feature of LDPC codes. It distinguishes them from general linear codes, whose parity-check matrices are typically dense and do not admit efficient iterative decoding algorithms.

2.3.2 Tanner Graph Representation

The structure of an LDPC code is often visualized using its *Tanner graph* [22]. For a code with parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, the Tanner graph is a bipartite graph with two types of nodes:

1. n *variable nodes*, one for each codeword bit c_j ($j = 1, \ldots, n$), corresponding to the columns of H; and
2. $n - k$ *check nodes*, one for each parity-check equation, corresponding to the rows of H.

There is an edge between variable node j and check node i if and only if $H_{ij} = 1$.

Figure 2.2 shows a small example Tanner graph.

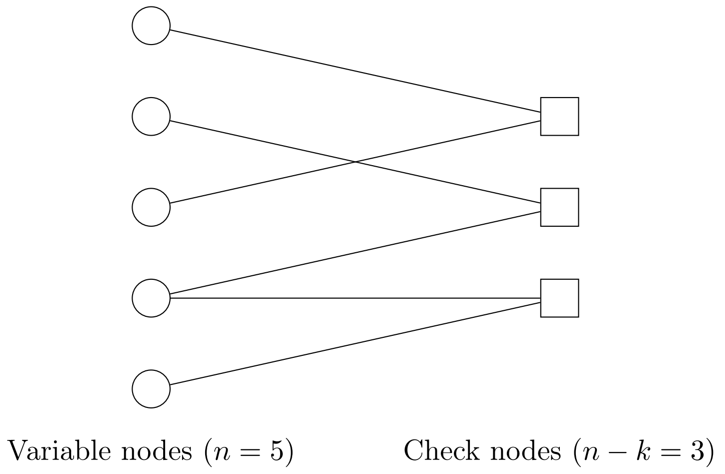


Figure 2.2: Example Tanner graph for a binary linear [5, 2] code with parity-check matrix $H \in \mathbb{F}_2^{3\times 5}$. Circles represent variable nodes corresponding to the $n = 5$ columns of H, squares represent check nodes corresponding to the $n-k = 3$ rows of H, and an edge connects a check node to a variable node whenever the corresponding entry of H is 1.

In this representation, variable nodes are drawn as circles (one for each codeword bit) and check nodes as squares (one for each parity constraint). An edge between a variable node and a check node indicates that the corresponding bit participates in that parity check, i.e., the corresponding entry of H is equal to 1. The *degree* of a variable node is the number of check nodes it is connected to, and the degree of a check node is the number of variable nodes it involves. Because H is sparse in an LDPC code, both variable and check nodes have relatively small degree, so the Tanner graph is locally sparse.

From this viewpoint, the matrix H and the Tanner graph encode the same incidence structure: H records which bits enter which parity equations, while the Tanner graph displays the same information as edges between variable and check nodes. This graphical representation will be important later, since many decoding algorithms for LDPC codes, including belief propagation and other message-passing schemes, can be interpreted as passing messages along the edges of the Tanner graph, with variable and check nodes updating their beliefs based on the parity constraints.

2.3.3 Syndromes and Iterative Decoding

The decoding process begins with computing the syndrome s as in (2.22).

Critically, s depends only on the error pattern e via $s = He^{\mathsf{T}}$ and is independent of the transmitted codeword c. The overall goal of decoding is to recover the original codeword with high probability, which can be approached in two equivalent ways:

1. *Codeword Viewpoint:* Identify the codeword $\hat{c} \in C$ that is most likely given the received word r.

2. *Error Viewpoint:* Find the most likely error pattern $\hat{e}$ that satisfies the syndrome equation $H\hat{e}^{\mathsf{T}} = s$, and then set the estimated codeword $\hat{c} = r + \hat{e}$.

A simple, though impractical, method is syndrome decoding using a lookup table. This involves precomputing a preferred error pattern $\hat{e}(s)$ (e.g., the one with the lowest Hamming weight consistent with s) for every possible syndrome s. The decoder then computes $s = Hr^{\mathsf{T}}$, looks up $\hat{e}(s)$, and outputs $\hat{c} = r + \hat{e}(s)$. While optimal for small block lengths, this method is computationally and memory-intensive for practical, long Low-Density Parity-Check (LDPC) codes (where n is in the hundreds or thousands) because the table size and complexity grow exponentially with $n - k$ (the number of syndromes is 2^{n-k}).

Belief Propagation

Practical decoders for large LDPC codes, such as the widely used Belief Propagation (BP) or sum–product algorithm, overcome the scalability challenge by exploiting the sparsity of H and its associated Tanner graph. BP, originally developed for probabilistic graphical models, interprets the Tanner graph as a factor graph for the posterior distribution of the codeword bits c given the channel observations r [23].

The posterior distribution can be factorized as:

$$P(c \mid r) \propto \left(\prod_{j=1}^{n} P(r_j \mid c_j)\right)\left(\prod_{i=1}^{n-k} (H_i c^{\mathsf{T}} + 1) \pmod 2\right). \tag{2.26}$$

where $P(r_j \mid c_j)$ is the single-bit channel likelihood.

Belief propagation approximates the marginal posteriors $P(c \mid r)$ by passing messages along the edges of the Tanner graph. The two distinct phases of this message passing are illustrated in Figure 2.3. On each iteration, the messages are typically calculated as Log-Likelihood Ratios (LLRs): $L(x) = \ln \frac{P(x=0)}{P(x=1)}$ [24].

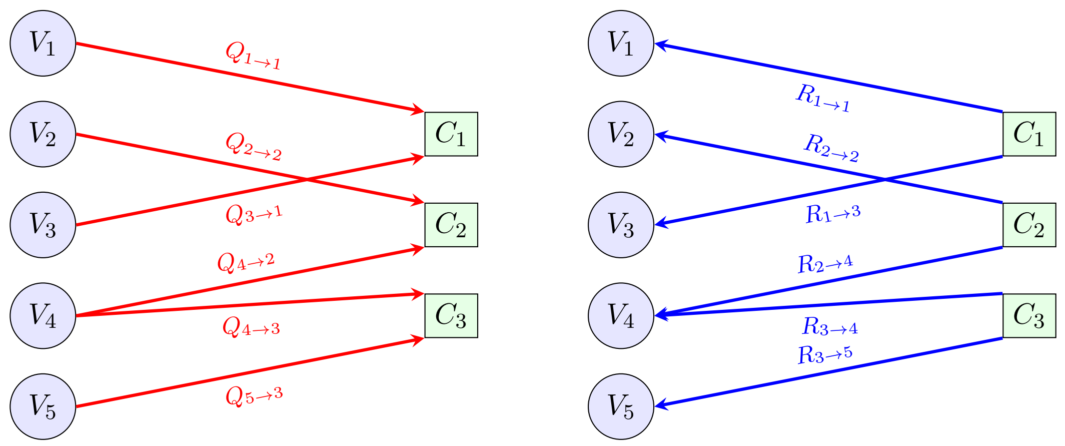
1. *Variable-to-Check Messages (Q):* Each variable node j sends to each neighboring check node i a message summarizing its current "belief" about c_j, based on the channel observation r_j and all incoming messages from other check nodes. This phase is shown in Figure 2.3a. The outgoing message, $Q_{j\to i}$, is calculated as the sum of the channel LLR, L_j^{ch}, and all incoming check-to-variable messages ($R_{i'\to j}$), *excluding* the message from i:

$$Q_{j\to i} = L_j^{\text{ch}} + \sum_{i' \in N(j)\setminus\{i\}} R_{i'\to j}. \tag{2.27}$$

2. *Check-to-Variable Messages (R):* Each check node i sends to each neighboring variable node j a message describing how the parity constraint at node i favors $c_j = 0$ or 1, given the messages received from the other variables participating in that check. This phase is shown in Figure 2.3b. The outgoing message, $R_{i\to j}$, is calculated from the incoming variable-to-check messages ($Q_{j'\to i}$), *excluding* the message from j:

$$R_{i\to j} = 2\tanh^{-1}\left(\prod_{j' \in N(i)\setminus\{j\}} \tanh\left(\frac{Q_{j'\to i}}{2}\right)\right). \tag{2.28}$$

These messages are real-valued quantities (often log-likelihood ratios) and are updated according to local sum–product rules derived from the factorization of $P(c \mid r)$. After a fixed number of iterations, or once the messages have approximately converged, each variable node computes its final posterior LLR, $L_j^{\text{post}} = L_j^{\text{ch}} + \sum_{i\in N(j)} R_{i\to j}$, and forms a local decision $\hat{c}_j$ by choosing the more likely bit value ($\hat{c}_j = 0$ if $L_j^{\text{post}} > 0$, and $\hat{c}_j = 1$ otherwise). The decoder then assembles $\hat{c} = (\hat{c}_1, \ldots, \hat{c}_n)$ and checks whether $H\hat{c}^{\mathsf{T}} = 0$; if so, a valid codeword has been found.



(a) Variable nodes V_j send messages $Q_{j\to i}$ to their neighboring check nodes C_i.

(b) Check nodes C_i send messages $R_{i\to j}$ back to their neighboring variable nodes V_j.

Figure 2.3: Belief Propagation Message Phases on the Example Tanner Graph

From the perspective of this thesis, the important points are that (i) LDPC codes admit sparse Tanner graphs, which make belief propagation computationally efficient (with complexity roughly linear in the block length for a fixed number of iterations), and (ii) the decoding problem can be phrased entirely in terms of parity-check constraints and local message updates on the graph. Later, when we embed decoding tasks inside quantum algorithms, we will reuse these syndrome- and Tanner-graph–based viewpoints to design and compare different decoder realizations.

2.4 Quantum Optimization

2.4.1 Fundamentals and Classical Optimization

Optimization is a core methodological pillar of Operations Research (OR), a discipline concerned with the development of models and algorithms for structured decision-making under objectives and constraints [25]. In its standard mathematical form, an optimization problem seeks a decision vector $\vec{x}$ that minimizes (or maximizes) an objective function while satisfying feasibility requirements:

$$\min_{\vec{x}\in\mathcal{X}} C(\vec{x}) \quad \text{subject to} \quad g_i(\vec{x}) \leq 0,\ h_j(\vec{x}) = 0, \tag{2.29}$$

where $\mathcal{X}$ denotes the search space (continuous, discrete, or mixed), and $\{g_i\}$ and $\{h_j\}$ encode inequality and equality constraints.

The classical optimization landscape is broad and is often organized by the structure of $\mathcal{X}$ and the analytical properties of $C(\cdot)$ [25]. Three widely used categories are:

1. **Continuous optimization (linear/convex/nonconvex):** When $\vec{x}$ ranges over a continuous domain, problems include linear programming (LP), quadratic programming (QP), and general nonlinear programming. Convex structure is particularly important because it enables polynomial-time algorithms and global optimality guarantees under suitable conditions. In contrast, nonconvex problems can exhibit multiple local minima and typically require additional assumptions or specialized methods.

2. **Discrete and combinatorial optimization:** When decision variables are discrete (e.g., binary or integer), the search space grows exponentially in the worst case. Many canonical problems in this class are NP-hard, motivating both exact algorithms (e.g., branch-and-bound, cutting planes) and approximate approaches.

3. **Heuristics and metaheuristics:** For large-scale or highly nonconvex instances where provable guarantees are difficult or exact approaches are computationally prohibitive, heuristic strategies are frequently employed. Metaheuristics such as simulated annealing and evolutionary algorithms aim to discover high-quality solutions using stochastic exploration and problem-independent search mechanisms, typically trading worst-case guarantees for practical performance.

2.4.2 General Optimization Framework

Despite the diversity of problem classes and algorithms, optimization methods generally follow a common iterative structure [26]. At a high level, one specifies:

1. **Problem formulation (model):** Identify decision variables $\vec{x}$, define the objective $C(\vec{x})$, and encode feasibility through constraints.

2. **Search space definition:** Specify the domain $\mathcal{X}$ (continuous, discrete, or mixed) and any structural properties that can be exploited (e.g., sparsity, convexity, decomposability).

3. **Algorithm selection:** Choose an algorithm appropriate to the problem structure (e.g., first/second-order methods for smooth continuous problems, combinatorial methods for discrete problems, heuristics for large or nonconvex instances).

4. **Evaluation and update rule:** Iteratively generate candidate solutions and evaluate $C(\vec{x})$ (and feasibility), then update the candidate according to the method's rule (e.g., gradient steps, neighborhood moves, acceptance criteria, or sampling).

5. **Termination criteria:** Stop when a budget is exhausted or when progress meets a convergence criterion (e.g., small improvement, stationarity, or feasibility/optimality thresholds).

This perspective is useful for quantum optimization because many quantum approaches preserve the same outer-loop logic (evaluate, update, terminate), while changing how candidates are represented and how exploration and evaluation are performed.

2.4.3 Optimization on Quantum Computers

Quantum optimization studies how quantum hardware can be used as a computational primitive inside an optimization workflow. In most approaches, the task is specified by an objective function (and possibly constraints), while the quantum device is used to (i) represent candidate solutions in quantum states and (ii) evaluate or bias the search via quantum dynamics and measurement [27]. Conceptually, quantum methods preserve the classical outer-loop structure (model $\rightarrow$ evaluate $\rightarrow$ update), but differ in how the objective is embedded into a physical process and how information is extracted from measurements.

From objective functions to observables A common design principle is to encode the objective into an operator whose measurement statistics reflect solution quality. For many discrete optimization problems, this is achieved by constructing a *cost Hamiltonian H_C* that is diagonal in the computational basis, so that each bitstring $\mathbf{z} \in \{0,1\}^n$ corresponds to a basis state $|\mathbf{z}\rangle$ with an associated energy proportional to the classical cost [28]. In this setting, optimization can be viewed as an energy-minimization task: low-energy states (or samples concentrated at low energies) correspond to high-quality candidate solutions.

A widely used modeling form is Quadratic Unconstrained Binary Optimization (QUBO), which provides a unifying representation for many combinatorial problems [29, 28]. QUBO objectives can be mapped to Ising-type Hamiltonians through a standard binary-to-spin transformation, yielding an operator of the form

$$H_C = \sum_i h_i \sigma_i^z + \sum_{i<j} J_{ij}\, \sigma_i^z \sigma_j^z + E_{\text{offset}}, \tag{2.30}$$

where (h_i, J_{ij}) depend on the problem instance and E_{offset} is an additive constant. The optimum of the encoded objective corresponds to the ground-state energy in the idealized model; practical algorithms aim to prepare low-energy states and/or obtain low-energy samples under hardware constraints [27].

Algorithmic paradigms Two paradigms are especially prominent:

1. **Adiabatic computation and quantum annealing.** In adiabatic approaches, the system is evolved under a time-dependent Hamiltonian that interpolates from

an initial Hamiltonian with a known ground state to the problem Hamiltonian H_C. Under suitable conditions, adiabatic evolution motivates ending in a state with high overlap on low-energy solutions [17]. This paradigm is closely related to analog quantum annealing implementations.

2. **Gate-based variational algorithms (VQAs).** For gate-model devices, hybrid variational methods prepare a parameterized state $\left|\psi(\vec{\theta})\right\rangle$ and use repeated measurements to estimate a cost such as $\langle H_C \rangle$, while a classical routine updates $\vec{\theta}$ to reduce the measured cost [30]. QAOA is a structured VQA tailored to combinatorial objectives, alternating between unitaries generated by H_C and a mixing Hamiltonian [1].

Practical considerations on NISQ hardware On NISQ devices, limited coherence and imperfect gate fidelities impose strict circuit-depth budgets, while restricted qubit connectivity can add compilation overhead (e.g., SWAP insertion) that further amplifies noise [31]. In variational and Hamiltonian-based workflows, the dominant wall-clock cost is often *measurement*: estimating objectives such as $\langle H_C \rangle$ to useful precision may require many circuit repetitions, especially when H_C decomposes into many terms, motivating grouping and other measurement-reduction strategies as well as approaches such as classical shadows [32]. Moreover, finite-shot variance and device noise make objective evaluations stochastic and biased; error-mitigation techniques (e.g., readout mitigation and zero-noise extrapolation) can reduce bias but typically increase sampling overhead, with fundamental tradeoffs limiting achievable accuracy at fixed experimental cost [33, 34, 35]. Consequently, practical quantum optimization pipelines commonly combine shallow ansätze with problem-tailored encodings and substantial classical preprocessing/postprocessing to obtain robust performance under realistic hardware constraints [27].

Chapter 3

Decoded Quantum Interferometry

3.1 Motivation and high-level idea

Since the early development of quantum algorithms for optimization, a large portion of the literature has concentrated on a small number of recurring paradigms: Hamiltonian-based methods such as adiabatic/annealing approaches, and hybrid variational frameworks such as VQE and QAOA [27, 30, 1]. These frameworks have been widely explored on NISQ hardware [31], but identifying clear, superpolynomial quantum advantage for practically relevant optimization tasks remains challenging and largely open [27].

Rather than tuning an ansatz through a quantum–classical training loop, DQI leverages structure: a QFT-based interference step converts the optimization objective into an induced decoding task, and a bounded-distance decoder is embedded coherently as a subroutine. The algorithm's success is therefore tied to the availability and quality of the decoder, which directly mediates the amplification of near-optimal solutions in the final measurement distribution [3, 36].

For certain highly structured problem families, DQI can offer particularly strong guarantees. One prominent example is the *Optimal Polynomial Intersection* (OPI) problem over finite fields, where the objective's algebraic structure induces a decoding instance that can be handled by efficient decoding primitives (e.g., Reed–Solomon / Reed–Muller decoding in the polynomial-fitting setting). More broadly, the DQI viewpoint highlights that many combinatorial objectives exhibit sparsity or exploitable structure in their Fourier spectrum, and that QFT-based interference can concentrate probability mass on high-quality solutions. At the same time, whether DQI yields advantages beyond such structured families remains an active research direction; recent analyses suggest performance can depend critically on instance structure and decoder capability [37].

3.2 Problem setting and notation

3.2.1 Max-XORSAT

In this thesis we focus on Max-XORSAT, specified by a binary matrix $B \in \{0,1\}^{m\times n}$ and a binary vector $v \in \mathbb{F}_2^m$. An assignment is $x \in \mathbb{F}_2^n$, and the ith XOR-constraint is

$$b_i \cdot x \;=\; v_i \pmod 2, \qquad i \in [m],$$

where $b_i \in \mathbb{F}_2^n$ is the ith row of B and $b_i \cdot x := \sum_{j=1}^n b_{i,j}x_j \pmod 2$. DQI targets this setting by reducing the optimization to a (coherently implemented) decoding subroutine.

Objective. Following the DQI formulation, we define the Max-XORSAT objective as

$$f(x) \;:=\; \sum_{i=1}^{m}(-1)^{\,v_i+b_i\cdot x}, \qquad x \in \mathbb{F}_2^n, \tag{3.1}$$

where each constraint is specified by a row vector $b_i \in \mathbb{F}_2^n$ and a bit $v_i \in \mathbb{F}_2$. Collecting $\{b_i\}_{i=1}^m$ into a matrix $B \in \mathbb{F}_2^{m\times n}$ and $\{v_i\}_{i=1}^m$ into $v \in \mathbb{F}_2^m$, the ith constraint is $b_i \cdot x = v_i \pmod 2$, i.e., $Bx = v$ over $\mathbb{F}_2$. Define the violation vector

$$y(x) \;:=\; Bx \oplus v \in \mathbb{F}_2^m, \tag{3.2}$$

so $y_i(x) = 1$ iff constraint i is violated. Writing $\mathrm{sat}(x)$ for the number of satisfied constraints,

$$\mathrm{sat}(x) = m - \mathrm{wt}(y(x)) \qquad \text{and} \qquad \mathrm{sat}(x) = \frac{m+f(x)}{2}. \tag{3.3}$$

Max-XORSAT asks for an x maximizing $f(x)$ (equivalently, maximizing $\mathrm{sat}(x)$).

Example. Consider a toy instance with $n = 4$ variables and $m = 3$ constraints:

$$B = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}, \qquad v = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \in \mathbb{F}_2^4.$$

The constraints expand to

$$\begin{aligned} c_1 &: x_1 \oplus x_2 \oplus x_4 = 0, \\ c_2 &: x_2 \oplus x_3 = 1, \\ c_3 &: x_1 \oplus x_3 \oplus x_4 = 0. \end{aligned}$$

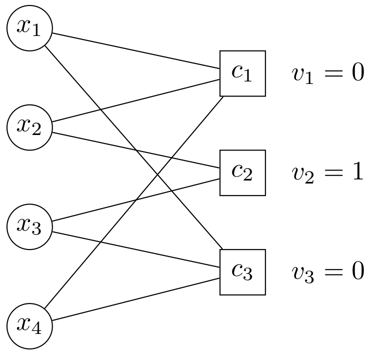


Figure 3.1: Tanner graph for the running Max-XORSAT instance (B, v): variable nodes (circles) and constraint nodes (squares), with an edge wherever $B_{i,j} = 1$.

For this instance,

$$f(x) = (-1)^{x_1+x_2+x_4} + (-1)^{1+x_2+x_3} + (-1)^{x_1+x_3+x_4},$$

and the violation vector is

$$y(x) = Bx \oplus v = \begin{pmatrix} x_1 \oplus x_2 \oplus x_4 \\ 1 \oplus x_2 \oplus x_3 \\ x_1 \oplus x_3 \oplus x_4 \end{pmatrix}.$$

The corresponding Tanner graph is shown in Fig. 3.1.

3.2.2 Coding viewpoint and induced decoding task.

The violation vector $y(x)$ defined in (3.2) marks which constraints are violated by an assignment x. Since $\mathrm{sat}(x) = m - \mathrm{wt}(y(x))$ (cf. (3.3)), near-optimal assignments correspond to *low-weight* violation patterns.

DQI leverages this by shifting attention from x to an (implicit) superposition over low-weight strings $y \in \mathbb{F}_2^m$. The induced coding-theoretic object is the *dual* code

$$\mathcal{C}^{\perp} := \{ y \in \mathbb{F}_2^m : B^T y = 0 \},$$

for which $B^T \in \mathbb{F}_2^{n \times m}$ serves as a parity-check matrix. The DQI circuit computes the associated syndrome

$$s := B^T y \in \mathbb{F}_2^n. \tag{3.4}$$

and the central primitive is *bounded-distance syndrome decoding*: given s (and a target radius ℓ), recover an error pattern $\hat{y}$ such that $B^T \hat{y} = s$ and $\mathrm{wt}(\hat{y}) \leq \ell$.

In the coherent (circuit) setting, the decoder is embedded as a reversible subroutine with an interface of the form

$$|s\rangle_S |0\rangle_E |0\rangle_A \longmapsto |s\rangle_S |\hat{y}(s)\rangle_E |0\rangle_A,$$

on the subset of syndromes that correspond to decodable errors within radius ℓ (with work register A uncomputed). This coherent decoder is then used within the DQI pipeline to map syndrome information back into an explicit low-weight error pattern y, enabling the amplitude concentration mechanism that biases measurement outcomes toward assignments with many satisfied constraints [3, 36].

3.3 DQI algorithm overview

Focusing on Max-XORSAT, DQI can be viewed as a *state-preparation* procedure whose goal is to produce a quantum state in which the amplitude of each assignment $x \in \mathbb{F}_2^n$ is a (polynomially amplified) function of the objective value $f(x)$. Fix a *normalized* polynomial function $P^{(\ell)} : \mathbb{R} \to \mathbb{R}$ of degree at most ℓ. The *target objective state* is

$$\left|P^{(\ell)}(f)\right\rangle := \sum_{x \in \mathbb{F}_2^n} P^{(\ell)}\big(f(x)\big) \left|x\right\rangle . \tag{3.5}$$

Measuring in the computational basis samples x with probability $P^{(\ell)}\big(f(x)\big)^2$.

The amplification strength is controlled by the degree parameter ℓ: larger ℓ can concentrate more probability mass on high-quality assignments, but it also corresponds to a larger bounded-distance decoding radius in the induced decoding subroutine used by DQI, and thus to higher circuit cost.

3.3.1 Unitary evolution for Max-XORSAT

We describe DQI as an explicit unitary evolution on named quantum registers. Let

Y be an m-qubit register, $\quad$ S be an n-qubit register, $\quad$ A be auxiliary work registers,

where Y indexes strings $y \in \mathbb{F}_2^m$ and S stores the induced syndrome $s = B^T y \in \mathbb{F}_2^n$. The goal is to implement a reversible transformation whose net effect (after a final Hadamard transform) prepares the polynomial-amplified objective state $|P(f)\rangle$.

Optimal weight selection and initial superposition. DQI begins with a *classical preprocessing step* in which a vector of weights $\mathbf{w} = (w_0, \ldots, w_\ell)^\mathsf{T}$ is computed as a function of the decoding radius ℓ. These weights are chosen to maximize the *expected fraction of satisfied constraints* under the DQI measurement distribution, subject to the restriction that only violation patterns of Hamming weight at most ℓ are coherently decoded.

In the ideal-decoder setting analyzed in [3], this optimization reduces to a Rayleigh–quotient problem of the form

$$\max_{\|\mathbf{w}\|_2 = 1} \mathbf{w}^\mathsf{T} A^{(m,\ell,d)} \mathbf{w},$$

where $A^{(m,\ell,d)} \in \mathbb{R}^{(\ell+1)\times(\ell+1)}$ is a symmetric tridiagonal matrix whose structure reflects interference between Hamming-weight layers under the quantum Fourier transform. Explicitly,

$$A^{(m,\ell,d)} = \begin{pmatrix} 0 & a_1 & & & \\ a_1 & d & a_2 & & \\ & a_2 & 2d & \ddots & \\ & & \ddots & \ddots & a_\ell \\ & & & a_\ell & \ell d \end{pmatrix},$$

with

$$a_k = \sqrt{k(m-k+1)}, \qquad d = \frac{p-2r}{\sqrt{r(p-r)}}.$$

The optimal weight vector $\mathbf{w} = (w_0, \dots, w_\ell)^T$ specifies the relative amplitudes assigned to each Hamming-weight layer k (i.e., to violation patterns of weight k) in the low-weight superposition up to the decoding radius ℓ, with $\sum_{k=0}^{\ell} |w_k|^2 = 1$. It is chosen as the normalized eigenvector corresponding to the largest eigenvalue of $A^{(m,\ell,d)}$. This choice maximizes the expected objective value achievable within decoding radius ℓ and uniquely determines the initial amplitude profile used by the algorithm.

Once the weights $\{w_k\}$ are computed, they are encoded into an initial quantum state $\sum_{k=0}^{\ell} w_k |k\rangle$, which then controls the subsequent preparation of the Dicke-state superposition over weight-k strings. We refer the reader to [3] for the full derivation of $A^{(m,\ell,d)}$, its spectral properties, and its asymptotic behavior.

Dicke-state preparation. Next we coherently prepare a superposition over low-weight violation patterns by attaching an m-qubit Dicke state of weight k on register Y. We model this by a controlled state-preparation unitary

$$U_{\text{Dicke}}: \quad |k\rangle |0^m\rangle_Y \longmapsto |k\rangle |D_{m,k}\rangle_Y, \qquad 0 \le k \le \ell, \tag{3.6}$$

where

$$|D_{m,k}\rangle = \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\substack{y \in \mathbb{F}_2^m \\ \text{wt}(y)=k}} |y\rangle.$$

Applying U_{Dicke} produces the joint state

$$|\psi_1\rangle = \sum_{k=0}^{\ell} w_k \; |k\rangle |D_{m,k}\rangle_Y |0\rangle_S |0\rangle_A,$$

i.e., the coefficients w_k control the relative amplitude assigned to each Hamming-weight layer in register Y.

Phase encoding of constraint right-hand side. Next we apply a diagonal phase operator that depends on the constraint vector $v \in \mathbb{F}_2^m$ in the Max-XORSAT instance $Bx = v \pmod 2$:

$$U_v : |y\rangle_Y \longmapsto (-1)^{v \cdot y} |y\rangle_Y , \qquad v \cdot y := \sum_{i=1}^{m} v_i y_i \pmod 2.$$

Acting on $|\psi_1\rangle$, this yields

$$|\psi_2\rangle = \sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\mathrm{wt}(y)=k} (-1)^{v \cdot y} |y\rangle_Y |0\rangle_S |0\rangle_A . \tag{3.7}$$

This phase is the point at which the Max-XORSAT instance offsets enter the interference pattern used by DQI.

Reversible syndrome computation. We then compute the syndrome associated with y into an n-qubit register S using the reversible linear map induced by B^T:

$$U_{\mathrm{syn}} : |y\rangle_Y |s\rangle_S \longmapsto |y\rangle_Y \left| s \oplus (B^T y) \right\rangle_S ,$$

so that in particular $|y\rangle_Y |0\rangle_S \mapsto |y\rangle_Y \left|B^T y\right\rangle_S$. Applying U_{syn} to (3.7) gives

$$|\psi_3\rangle = \sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\mathrm{wt}(y)=k} (-1)^{v \cdot y} |y\rangle_Y \left|B^T y\right\rangle_S |0\rangle_A . \tag{3.8}$$

Coherent bounded-distance syndrome decoding and uncomputation. The central primitive is a *coherently embedded* bounded-distance decoder for the (classical) code with parity-check matrix B^T. On syndromes that are decodable within radius ℓ, it is implemented as a reversible map

$$U_{\mathrm{dec}} : \quad |s\rangle_S |0\rangle_E |0\rangle_A \longmapsto |s\rangle_S |\hat{y}(s)\rangle_E |0\rangle_A , \qquad B^T \hat{y}(s) = s, \ \mathrm{wt}\big(\hat{y}(s)\big) \le \ell, \tag{3.9}$$

with all work space returned to $|0\rangle$.

After applying U_{dec}, we erase the original Y register by bitwise XOR,

$$|y\rangle_Y |y\rangle_E \longmapsto |0\rangle_Y |y\rangle_E ,$$

and then uncompute any remaining temporary registers (including E if desired), leaving an effective state supported on the syndrome register:

$$|\psi_4\rangle = \sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\mathrm{wt}(y)=k} (-1)^{v \cdot y} |0\rangle_Y \left|B^T y\right\rangle_S |0\rangle_A . \tag{3.10}$$

Interference and measurement. Finally we apply the Hadamard transform to the syndrome register:

$$|\psi_5\rangle = \left(I_Y \otimes H_S^{\otimes n} \otimes I_A\right) |\psi_4\rangle . \tag{3.11}$$

The DQI interference identity shows that the S-register of $|\psi_5\rangle$ is exactly the polynomial-amplified objective state $\left|P^{(\ell)}(f)\right\rangle$, with $P^{(\ell)}$ normalized as in the original analysis. Measuring in the computational basis therefore yields assignments biased toward larger objective values, with the bias–complexity tradeoff governed by the decoding radius ℓ (and the associated optimal weights w).

3.4 Estimating expected satisfaction

A distinctive advantage of DQI, compared to many heuristic optimization methods, is its *predictability*: under an idealized (perfect-decoding) model, one can estimate the expected satisfaction achieved by the algorithm *before* running any quantum circuit. This kind of predictability has been highlighted as a useful lens when discussing what constitutes meaningful quantum advantage in broad "real-world" settings. [38]

Concretely, Jordan *et al.* derive a closed-form approximation for the *expected satisfaction fraction* obtained by measuring the final DQI state, as a function of the number of constraints m, the baseline random-satisfaction rate r/p, and the DQI parameter ℓ (under perfect decoding up to radius ℓ).

In the general MAX-LINSAT setting over a finite field $\mathbb{F}_p$, each constraint is specified by a subset $F_i \subseteq \mathbb{F}_p$ of size r, so a uniformly random assignment satisfies a fraction r/p of constraints in expectation. Let $\langle s\rangle$ denote the expected number of satisfied constraints (out of m) for the string produced by measuring the DQI output state.

The parameter ℓ plays *two roles* in DQI: (i) it controls the degree of the polynomial amplifier (equivalently, the maximum Hamming weight used in the low-weight superposition), and (ii) it is the bounded-distance decoding radius required of the (classical) decoder embedded coherently in the circuit. Intuitively, increasing ℓ can yield stronger amplification toward good solutions, but it demands a more powerful decoder and higher circuit cost.

For decoding up to ℓ errors in the induced dual code $\mathcal{C}^\perp = \{d \in \mathbb{F}_p^m : B^T d = 0\}$, the achievable satisfaction fraction is approximated by

$$\frac{\langle s\rangle}{m} = \left(\sqrt{\frac{\ell}{m}\left(1-\frac{r}{p}\right)} + \sqrt{\frac{r}{p}\left(1-\frac{\ell}{m}\right)}\right)^2, \qquad \text{if } \frac{r}{p} \le 1 - \frac{\ell}{m}, \tag{3.12}$$

and otherwise $\langle s\rangle/m = 1$. Following [3], we informally refer to (3.12) as the *semicircle law*, because it becomes an actual semicircle in the balanced case $r/p = 1/2$.

3.4.1 Specialization to Max-XORSAT.

For Max-XORSAT we have $p = 2$ and $r = 1$, hence $r/p = 1/2$. Substituting into (3.12) yields (for $\ell/m \leq 1/2$)

$$\frac{\langle s \rangle}{m} = \frac{1}{2} + \sqrt{\frac{\ell}{m}\left(1 - \frac{\ell}{m}\right)}. \tag{3.13}$$

Equivalently, letting $t := \ell/m$ and $u := \langle s \rangle/m - 1/2$, we obtain the geometric form

$$\left(t - \frac{1}{2}\right)^2 + u^2 = \frac{1}{4}, \tag{3.14}$$

which is a semicircle of radius 1/2 centered at $(t, u) = (1/2, 0)$.

MaxCut as a Max-XORSAT instance. MaxCut is a structured special case of Max-XORSAT in which every constraint has degree 2. Given a graph $G = (V, E)$, assign a bit $x_u \in \mathbb{F}_2$ to each vertex $u \in V$. An edge $e = (u, v)$ is cut iff $x_u \oplus x_v = 1$. Thus MaxCut can be written as the Max-XORSAT system with one XOR constraint per edge,

$$x_u \oplus x_v = 1 \qquad \text{for each } (u, v) \in E,$$

i.e., $m = |E|$ constraints, each row of B having Hamming weight 2, and offset bits $v_e = 1$ for all edges. In this encoding, the number of satisfied constraints equals the cut size, so the satisfaction fraction $\langle s \rangle/m$ is exactly the *expected cut fraction* under the DQI output distribution.

3.5 From optimization to decoding

A central conceptual move in Decoded Quantum Interferometry (DQI) is to replace "searching over assignments x" with a *decoding* task over low-weight *violation patterns.* At a high level, DQI uses a Fourier/Hadamard interference step (a finite-field analogue of a QFT) to convert optimization structure into an *in-circuit* bounded-distance decoding problem. This perspective is closely aligned with the broader theme that *quantum Fourier/interference primitives can enable problem reductions* by turning global structure into locally checkable algebraic constraints—an idea that also appears in cryptographic settings where quantum advantage can be obtained without relying on visible instance structure. [3, 37, 39]

3.5.1 The induced decoding instance

For Max-XORSAT with constraint matrix $B \in \mathbb{F}_2^{m \times n}$ and offset vector $v \in \mathbb{F}_2^m$, DQI introduces the violation vector $y(x) = Bx \oplus v \in \mathbb{F}_2^m$, where $y_i(x) = 1$ indicates that constraint i is violated. Near-optimal assignments correspond to *low-weight* violation patterns y. DQI therefore shifts attention from x to a superposition over low-weight strings $y \in \mathbb{F}_2^m$ with $\mathrm{wt}(y) \leq \ell$.

The interference step leads to a decoding viewpoint in which $B^T \in \mathbb{F}_2^{n\times m}$ acts as a parity-check matrix for the induced (dual) code, and the circuit computes the associated syndrome

$$s \ := \ B^T y \in \mathbb{F}_2^n. \tag{3.15}$$

Thus, the *new* computational task inside DQI is: given s, recover a low-weight vector y consistent with it.

3.5.2 Bounded-distance syndrome decoding

In Max-XORSAT we typically have $m > n$, so $B^T \in \mathbb{F}_2^{n\times m}$ is *not square* and has no ordinary matrix inverse. The map $y \mapsto s = B^T y$ is therefore many-to-one: for a given syndrome s, there are generally many strings y satisfying $B^T y = s$, so recovering y from s is an *underdetermined* linear-algebra problem.

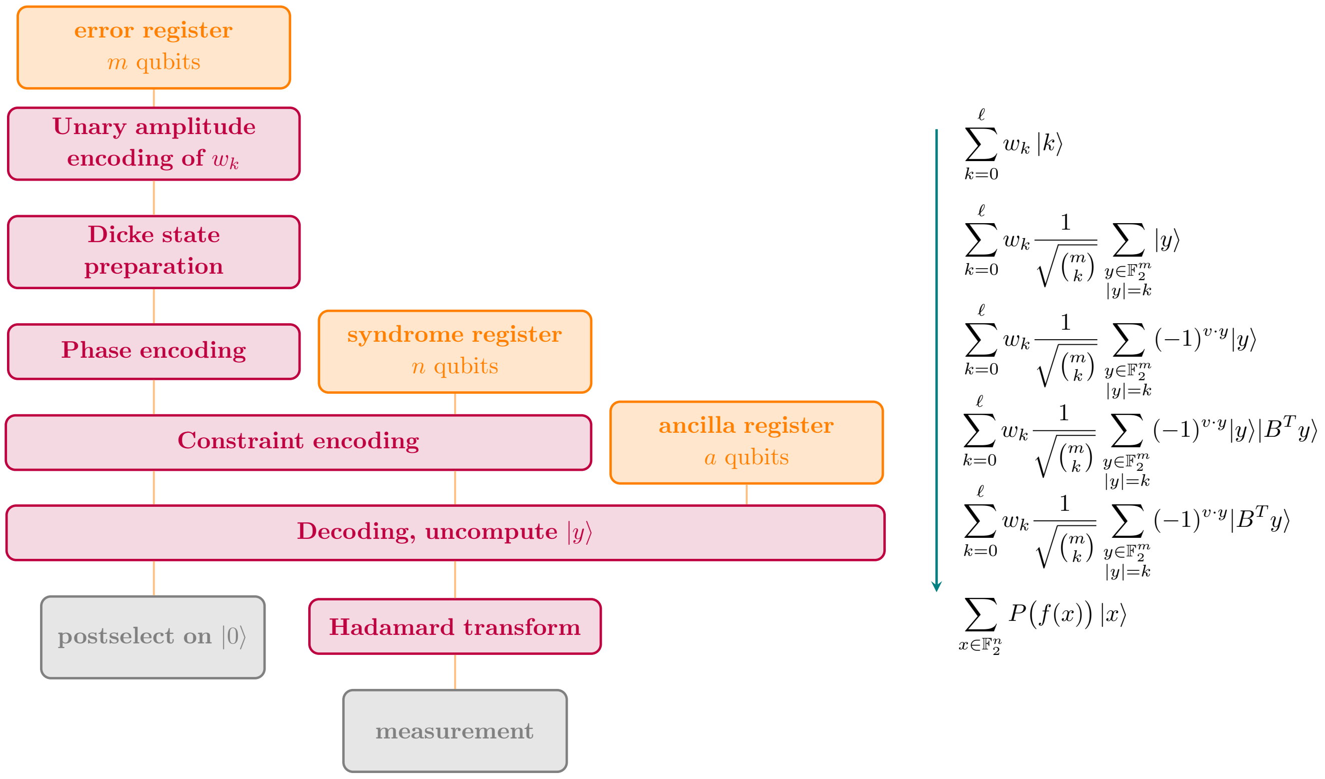
Because B^T is nonsquare and the solution is selected by the low-weight constraint, the *choice of decoding algorithm* (and how well it succeeds within radius ℓ) directly impacts DQI's practical performance and resource cost: stronger amplification requires larger ℓ, which corresponds to solving a harder decoding problem.

$$\text{find } \hat{y} \in \mathbb{F}_2^m \text{ such that } B^T \hat{y} = s \quad \text{and} \quad \mathrm{wt}(\hat{y}) \leq \ell. \tag{3.16}$$

In this bounded-distance regime, the decoder behaves like a *partial inverse*: it maps the syndrome s to the unique (or preferred) low-weight representative $\hat{y}$ whenever such a representative exists and is decodable. This is exactly the sense in which the decoder "inverts" the syndrome map on the subset of syndromes arising from ℓ-bounded errors.

Chapter 4

Implementation



error register
m qubits
Unary amplitude encoding of w_k
Dicke state preparation
Phase encoding
syndrome register
n qubits
Constraint encoding
ancilla register
a qubits
Decoding, uncompute $|y\rangle$
postselect on $|0\rangle$
Hadamard transform
measurement
$\sum_{k=0}^{\ell} w_k \, |k\rangle$
$\sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\substack{y \in \mathbb{F}_2^m \\ |y|=k}} |y\rangle$
$\sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\substack{y \in \mathbb{F}_2^m \\ |y|=k}} (-1)^{v \cdot y} |y\rangle$
$\sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\substack{y \in \mathbb{F}_2^m \\ |y|=k}} (-1)^{v \cdot y} |y\rangle |B^T y\rangle$
$\sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\substack{y \in \mathbb{F}_2^m \\ |y|=k}} (-1)^{v \cdot y} |B^T y\rangle$
$\sum_{x \in \mathbb{F}_2^n} P\big(f(x)\big) \, |x\rangle$

Figure 4.1: DQI algorithm state evolution diagram.

This chapter describes an explicit, circuit-level implementation of the DQI pipeline for Max-XORSAT. Figure 4.1 (state evolution diagram) provides a high-level view of how the quantum state is transformed from the initial all-zero state into an "objective-amplified" state whose measurement is biased toward assignments with larger Max-XORSAT objective value.

Registers. We consider a Max-XORSAT instance specified by a binary matrix $B \in \mathbb{F}_2^{m\times n}$ and an offset vector $v \in \mathbb{F}_2^m$, defining m XOR constraints on n binary variables via

$$Bx = v \pmod 2, \qquad x \in \mathbb{F}_2^n,$$

where each row of B corresponds to one constraint and each column corresponds to one variable. Throughout the implementation we use the following named registers:

1. **Error register Y (size m):** stores bit strings $y \in \mathbb{F}_2^m$ that index Hamming-weight layers (and later serve as the "error" to be decoded).
2. **Syndrome register S (size n):** stores the induced syndrome $s = B^T y \in \mathbb{F}_2^n$ computed from the Max-XORSAT instance matrix B.
3. **Ancillas A:** workspace qubits used only by certain decoding subroutines (and their uncomputation). Other components of our design require no ancillas. Ancillas are returned to $|0\rangle$ whenever possible.

After the final Hadamard transform, we interpret the syndrome register S as holding the candidate assignment $x \in \{0,1\}^n$ to be measured.

Algorithm stages (as in Fig. 4.1). We consider a Max-XORSAT instance specified by $B \in \mathbb{F}_2^{m\times n}$ and $v \in \mathbb{F}_2^m$ defining XOR constraints $Bx = v \pmod 2$ on $x \in \mathbb{F}_2^n$. The end-to-end circuit is organized into:

1. **Unary amplitude encoding of $\{w_k\}_{k=0}^{\ell}$.** Starting from $|0\rangle^{\otimes m}$ in the error register Y, apply a unitary that prepares $\sum_{k=0}^{\ell} w_k |k\rangle$ (in unary/amplitude form on Y), encoding the optimized weights $\{w_k\}$. In the ideal analysis these coefficients are derived from the principal eigenvector of an $(\ell{+}1)\times(\ell{+}1)$ matrix (see the DQI analysis referenced in the circuit paper).
2. **Dicke state preparation.** Transform the amplitude-encoded state into a superposition over Hamming-weight layers,

$$\sum_{k=0}^{\ell} w_k |D(m,k)\rangle ,$$

where $|D(m,k)\rangle$ is the uniform superposition over all $y \in \{0,1\}^m$ with $|y| = k$.
3. **Phase encoding.** Apply a diagonal phase so that each basis state $|y\rangle$ acquires the phase $(-1)^{v\cdot y}$, where $v \in \mathbb{F}_2^m$ is the instance offset vector (the right-hand side of $Bx = v$).

4. **Constraint (syndrome) encoding.** Reversibly compute the syndrome $s = B^T y \in \mathbb{F}_2^n$ into the syndrome register S, producing $|y\rangle_Y \left|B^T y\right\rangle_S$.

5. **Decoding and uncomputation of $|y\rangle$ (uses ancillas).** Apply a coherent decoding unitary acting on (Y, S, A) that, given $\left|B^T y\right\rangle_S$ (and any required workspace), computes a decoded representation of y and then uses controlled-XOR operations to uncompute the error register Y. In our design, ancillas A are introduced only in this decoding stage (and its uncomputation) and are returned to $|0\rangle$ whenever possible.

6. **Hadamard transform (Max-XORSAT QFT step).** Apply $H^{\otimes n}$ to the syndrome register. For Max-XORSAT, this Fourier/Hadamard interference step yields a superposition of the form $\sum_{x\in\mathbb{F}_2^n} P(f(x))\,|x\rangle$, with P defined (and normalized) as in the DQI analysis.

7. **Measurement with postselection.** Measure all registers and post-select on the error register being $|0\rangle^{\otimes m}$; the observed string in the (post-Hadamard) syndrome register is taken as the output assignment x.

4.1 Unary amplitude encoding (UAE)

To coherently weight the Hamming-weight layers in the subsequent Dicke-state preparation step, we first encode a set of real coefficients $\{w_k\}_{k=0}^{\ell}$ (assumed normalized, $\sum_{k=0}^{\ell} w_k^2 = 1$) into a *unary (thermometer) basis* on $(\ell+1)$ qubits. More generally, preparing a target amplitude vector can be viewed as a standard *state preparation* task, for which many generic synthesis techniques are known [40, 41]. In Jordan *et al.*, one suggested approach is to use the state-preparation technique of Low, Kliuchnikov, and Schaeffer [42]. However, in our implementation we adopt a deterministic Dicke-state preparation procedure [43], and this subroutine naturally accepts a unary-weighted superposition over Hamming-weight layers. As a result, we do not require a fully general state-preparation routine for $\{w_k\}$; instead, it suffices to prepare the unary superposition using a fan-out-based construction (i.e., controlled operations with shared controls), which fits our circuit structure and resource goals [44].

We define the unary basis states as

$$|k\rangle_{\text{unary}} := \left|\underbrace{11\cdots 1}_{k}\underbrace{00\cdots 0}_{(\ell+1-k)}\right\rangle, \qquad k = 0, 1, \ldots, \ell, \tag{4.1}$$

and aim to prepare the weighted superposition

$$|\psi_{\text{UAE}}\rangle = \sum_{k=0}^{\ell} w_k\,|k\rangle_{\text{unary}}\,. \tag{4.2}$$

Circuit structure. UAE can be implemented using a simple ladder of R_y and controlled-R_y rotations. The first qubit is rotated by $R_y(\beta_0)$, and each subsequent qubit i is rotated by a controlled-$R_y(\beta_i)$ conditioned on qubit $i-1$ being $|1\rangle$. This "fanout ladder" structure is shown in Fig. 4.2.

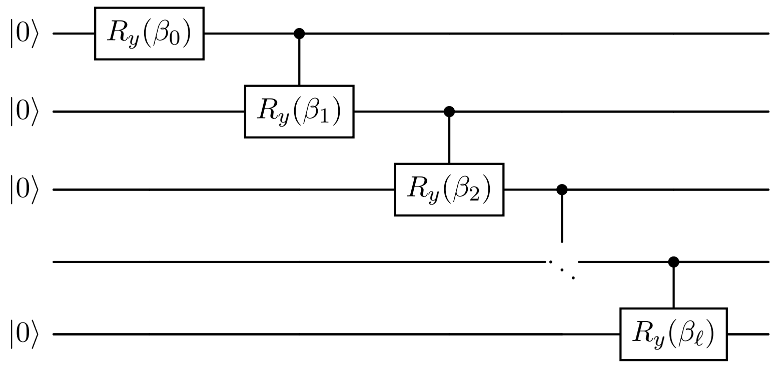


Figure 4.2: Unary amplitude encoding (UAE) ladder. The cascade of R_y and controlled-R_y gates prepares $\sum_{k=0}^{\ell} w_k |k\rangle_{\text{unary}}$ where $|k\rangle_{\text{unary}} = \left|1^k 0^{\ell+1-k}\right\rangle$.

Rotation angles from $\{w_k\}$. Let

$$R_k := 1 - \sum_{j<k} w_j^2 \tag{4.3}$$

denote the *remaining probability mass* before allocating weight k. On the branch where the first k qubits are already $|1\rangle$, the level-k rotation splits the remaining amplitude into a "stop" branch (yielding $|k\rangle_{\text{unary}}$) and a "continue" branch (propagating to $|k+1\rangle_{\text{unary}}$). Choosing β_k such that

$$\cos^2\left(\frac{\beta_k}{2}\right) = \frac{w_k^2}{R_k} \qquad \Longleftrightarrow \qquad \beta_k = 2\arccos\left(\frac{w_k}{\sqrt{R_k}}\right) \tag{4.4}$$

ensures

$$\Pr\Big(|k\rangle_{\text{unary}}\Big) = R_k \cos^2\left(\frac{\beta_k}{2}\right) = w_k^2, \tag{4.5}$$

so the amplitude of $|k\rangle_{\text{unary}}$ equals w_k (up to an overall global phase).

4.2 Dicke state preparation

Dicke states [45] are a canonical family of highly entangled *symmetric* multi-qubit states: for fixed (m,k), $|D_{m,k}\rangle$ is the equal-amplitude superposition over all m-bit strings with exactly k ones. Beyond being a natural structured target for state-preparation research, Dicke states appear as useful resources in several settings, including multiparty quantum networking protocols, quantum metrology, and quantum algorithms for constrained combinatorial optimization [43, 46, 47, 48]. In particular, they can serve as starting states that represent a uniform superposition over a hard-constraint feasible set (e.g., fixed Hamming weight), which is exactly the structure we exploit here.

After Unary Amplitude Encoding (UAE), the next step is to convert the prepared unary superposition into a coherent superposition of Dicke states on the same m-qubit error register Y. Recall the Dicke state of weight k on m qubits,

$$|D_{m,k}\rangle := \binom{m}{k}^{-\frac{1}{2}} \sum_{\substack{y\in\{0,1\}^m \\ \mathrm{wt}(y)=k}} |y\rangle , \qquad k = 0, 1, \ldots, m, \tag{4.6}$$

which is the equal-amplitude superposition over all bit strings of Hamming weight k. In DQI, we require a *low-weight* superposition over Dicke layers,

$$|\psi_1\rangle = \sum_{k=0}^{\ell} w_k |D_{m,k}\rangle , \tag{4.7}$$

where the coefficients $\{w_k\}_{k=0}^{\ell}$ are determined classically (e.g., from the principal eigenvector described in the previous subsection).

Key simplification: no extra control register is required. A direct approach would be to store k in a separate register and apply a k-controlled Dicke-preparation circuit. In our implementation this is unnecessary. UAE already prepares a superposition over *unary basis states* on the same m-qubit register Y:

$$|\psi_{\mathrm{UAE}}\rangle = \sum_{k=0}^{\ell} w_k |u_k\rangle , \qquad |u_k\rangle := \left|1^k\, 0^{m-k}\right\rangle \tag{4.8}$$

(up to a fixed qubit-ordering convention). Therefore, if we have a single deterministic unitary $U_{m,\ell}$ that maps each unary basis state to the corresponding Dicke state,

$$U_{m,\ell} |u_k\rangle = |D_{m,k}\rangle \qquad \text{for all } k \in \{0, 1, \ldots, \ell\}, \tag{4.9}$$

then by linearity we obtain the desired superposition with *one* application of $U_{m,\ell}$:

$$U_{m,\ell} |\psi_{\mathrm{UAE}}\rangle = \sum_{k=0}^{\ell} w_k\, U_{m,\ell} |u_k\rangle = \sum_{k=0}^{\ell} w_k |D_{m,k}\rangle . \tag{4.10}$$

This removes the need for an additional "weight register" and avoids implementing controlled versions of different k-dependent Dicke circuits, thereby reducing the overall qubit footprint of state preparation.

4.2.1 Deterministic Dicke-state preparation protocol

To implement $U_{m,\ell}$, we adopt the deterministic construction of Bärtschi and Eidenbenz [43]. Their approach is based on an inductive "split" of amplitudes that mirrors the well-known Dicke recursion:

$$|D_{n,h}\rangle = \sqrt{\frac{n-h}{n}}\, |0\rangle |D_{n-1,h}\rangle + \sqrt{\frac{h}{n}}\, |1\rangle |D_{n-1,h-1}\rangle , \qquad 0 \le h \le n, \tag{4.11}$$

and packages the required amplitude splits and routing operations into modular unitaries called *Split-and-Cyclic-Shift* (SCS) blocks. Intuitively, each SCS block performs (i) an amplitude split with the correct combinatorial ratio and (ii) a cyclic shift that routes excitations so that repeated application produces a uniform superposition over all placements of the h ones.

A useful way to view the amplitude split is through a single-qubit R_y rotation: since $R_y(\theta)|0\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle$, matching the coefficients in (4.11) corresponds to choosing angles of the form

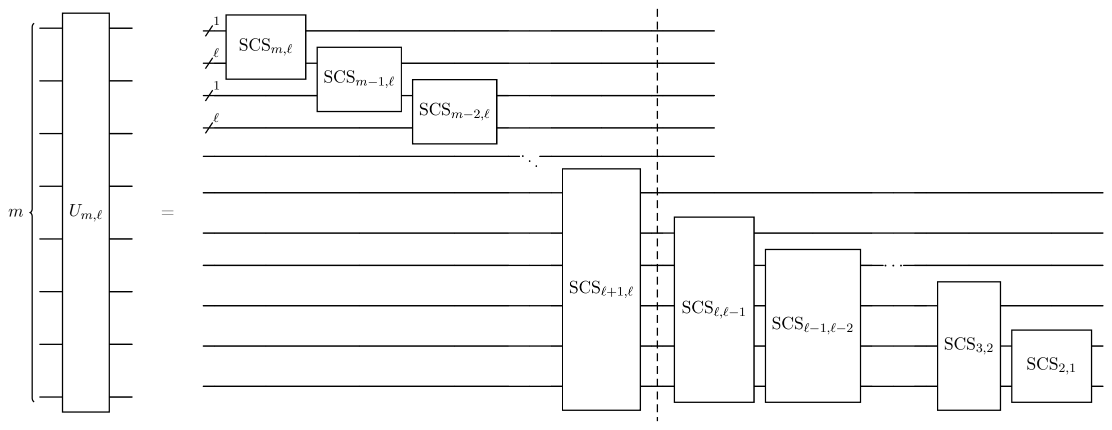
$$\theta_{n,h} = 2 \arccos\left(\sqrt{\frac{n-h}{n}}\right) = 2 \arcsin\left(\sqrt{\frac{h}{n}}\right), \tag{4.12}$$

with additional conditional logic ensuring that the split is applied only on the appropriate branches. In [43], each $\mathrm{SCS}_{n,h}$ is further decomposed into two elementary building blocks: a three-qubit primitive SCS_3 used repeatedly for routing/mixing, followed by a two-qubit primitive SCS_2 that completes the step.

Inductive construction of $U_{m,\ell}$ and how to read Fig. 4.3. Figure 4.3 expands the abstract Dicke unitary $U_{m,\ell}$ into a structured sequence of SCS modules. The construction proceeds in two stages:

1. **Stage I (expand from ℓ to m).** Apply $\mathrm{SCS}_{q,\ell}$ for $q = m, m-1, \dots, \ell+1$ on overlapping subsets of the Y register. This stage "grows" the effective support of the ℓ excitations across the full m qubits while preserving the correct combinatorial splitting implied by (4.11).
2. **Stage II (refine/symmetrize within the low-weight sector).** Apply $\mathrm{SCS}_{q,q-1}$ for $q = \ell, \ell-1, \dots, 2$. This stage removes residual ordering bias inherited from the unary input and completes the symmetrization, ensuring that, for each $k \le \ell$, the resulting state in the weight-k subspace is the uniform Dicke state $|D_{m,k}\rangle$.

Because the overall procedure is unitary, it is *deterministic* (no postselection). Combined with (4.10), the same fixed unitary $U_{m,\ell}$ transforms the entire UAE superposition into the desired Dicke-layer superposition (4.7).



m
$U_{m,\ell}$
$=$
1
ℓ
$\mathrm{SCS}_{m,\ell}$
$\mathrm{SCS}_{m-1,\ell}$
$\mathrm{SCS}_{m-2,\ell}$
$\mathrm{SCS}_{\ell+1,\ell}$
$\mathrm{SCS}_{\ell,\ell-1}$
$\mathrm{SCS}_{\ell-1,\ell-2}$
$\mathrm{SCS}_{3,2}$
$\mathrm{SCS}_{2,1}$

Figure 4.3: Inductive construction of $U_{m,\ell}$ as a sequence of Split-and-Cyclic-Shift (SCS) modules. The first stage applies $\mathrm{SCS}_{q,\ell}$ for $q = m, m-1, \ldots, \ell+1$, followed by a refinement stage $\mathrm{SCS}_{q,q-1}$ for $q = \ell, \ell-1, \ldots, 2$.

SCS Primitives. The efficiency of the $U_{m,\ell}$ unitary is derived from its modular decomposition into two primary building blocks: the SCS_2 (splitting) and SCS_k (routing) primitives [43]. Unlike generic rotation gates, these primitives are designed to act only on specific subspaces of the unary encoding while leaving others invariant, ensuring the deterministic nature of the Dicke state preparation.

The SCS_2 primitive acts on two qubits $(n-1, n)$ to perform the fundamental amplitude split. As shown in Fig. 4.4(a), it is defined by the following transformations:

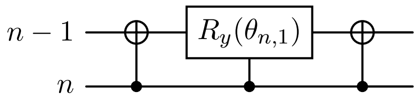
$$\begin{aligned} |00\rangle_{n-1,n} &\to |00\rangle_{n-1,n} \\ |11\rangle_{n-1,n} &\to |11\rangle_{n-1,n} \\ |01\rangle_{n-1,n} &\to \sqrt{\frac{1}{n}}\,|01\rangle_{n-1,n} + \sqrt{\frac{n-1}{n}}\,|10\rangle_{n-1,n} \end{aligned} \tag{4.13}$$

This mapping ensures that if no excitation is present, or if the qubits are already saturated, the state remains unchanged. The split only occurs when a single excitation is found in the lower-index qubit.

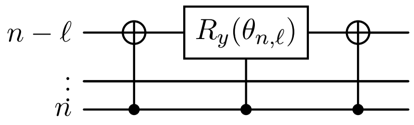
For cases where $k > 1$ excitations exist, the SCS_k primitive (effectively an SCS_3 interaction) routes existing ones to prevent collisions. Acting on qubits $(n-\ell, n-\ell+1, \ldots, n)$, it selectively transforms the state when an excitation is detected in the "overflow" position:

$$|01\rangle_{n-\ell}\,|1\rangle_n \to \sqrt{\frac{\ell}{n}}\,|01\rangle_{n-\ell}\,|1\rangle_n + \sqrt{\frac{n-\ell}{n}}\,|11\rangle_{n-\ell}\,|0\rangle_n\,. \tag{4.14}$$

As illustrated in Fig. 4.4(b).



(a) The SCS$_2$ primitive.



(b) The SCS$_3$ primitive.

Figure 4.4: Implementation of the SCS primitives as used in the Dicke state preparation protocol.

4.3 Problem encoding for Max-XORSAT

After the low-weight state preparation stage (UAE followed by deterministic Dicke preparation), the error register Y holds a superposition over low-weight violation patterns

$$|\psi_1\rangle = \sum_{k=0}^{\ell} w_k \, |D_{m,k}\rangle_Y \, |0^n\rangle_S \,, \tag{4.15}$$

where $|D_{m,k}\rangle$ is the m-qubit Dicke state of Hamming weight k.

Up to this point, the quantum circuit is essentially *instance-independent*: it prepares a low-weight superposition over candidate violation patterns y, controlled by (m, ℓ) and the chosen weights $\{w_k\}$, but it does not yet depend on any particular Max-XORSAT instance. We therefore refer to the next stage as *problem encoding* because it is the first place where the concrete instance data (B, v) is injected into the quantum program. Concretely, we encode (B, v) by (i) imprinting the right-hand side vector v as a diagonal phase on the error pattern register, and (ii) applying a reversible linear map determined by B to compute the induced syndrome into an auxiliary register. After these two steps, subsequent interference and decoding operate on the specific problem instance we want to solve, rather than on a generic low-weight superposition.

4.3.1 Phase encoding of the instance offsets v

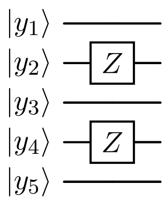


Figure 4.5: Example structure of the phase-encoding block U_v for a case where $v_2 = v_4 = 1$ (all other $v_i = 0$). In general, apply Z on Y_i iff $v_i = 1$.

The Max-XORSAT offsets enter DQI as a *sign* (a ± 1 phase) attached to each violation pattern $y \in \{0,1\}^m$. Define the diagonal unitary

$$U_v : \ |y\rangle_Y \longmapsto (-1)^{v \cdot y} |y\rangle_Y , \qquad v \cdot y := \sum_{i=1}^{m} v_i y_i \ (\text{mod } 2). \tag{4.16}$$

Acting on (4.15), this produces

$$|\psi_2\rangle = \sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\substack{y \in \{0,1\}^m \\ \text{wt}(y)=k}} (-1)^{v \cdot y} |y\rangle_Y |0^n\rangle_S . \tag{4.17}$$

Gate-level realization. For the binary case, (4.16) is implemented with single-qubit Z gates on those positions where $v_i = 1$ 4.5:

$$U_v = \prod_{i:\, v_i = 1} Z_{Y_i}. \tag{4.18}$$

This works because $Z|y_i\rangle = (-1)^{y_i}|y_i\rangle$, so multiplying Z_{Y_i} over all indices with $v_i = 1$ yields the global phase $(-1)^{\sum_i v_i y_i}$.

4.3.2 Constraint encoding via reversible syndrome computation

The second encoding step computes the induced syndrome associated with a violation pattern y under the Max-XORSAT constraint matrix B. We call this *constraint* encoding because, in Max-XORSAT, the matrix B is precisely the instance object that specifies the set of XOR constraints: each row of B selects which variables participate in one linear mod-2 equation, and together (B, v) defines the constraint system $Bx = v$ to be satisfied as much as possible.In other words, B is the natural "constraint matrix" representation of the XORSAT instance.

Viewing $B^T \in \mathbb{F}_2^{n \times m}$ as a parity-check matrix, we define the reversible map

$$U_{\text{syn}} : \ |y\rangle_Y |s\rangle_S \longmapsto |y\rangle_Y \left|s \oplus (B^T y)\right\rangle_S . \tag{4.19}$$

Starting from $s = 0^n$, the syndrome register becomes

$$|\psi_3\rangle = \sum_{k=0}^{\ell} w_k \frac{1}{\sqrt{\binom{m}{k}}} \sum_{\substack{y \in \{0,1\}^m \\ \text{wt}(y)=k}} (-1)^{v \cdot y} |y\rangle_Y \left|B^T y\right\rangle_S . \tag{4.20}$$

This is the point at which the DQI pipeline has fully embedded the Max-XORSAT instance into the quantum state: the register Y indexes low-weight violation patterns, while S holds the corresponding syndromes that will be fed into the coherent decoder.

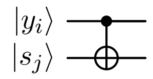


Figure 4.6: Elementary building block for syndrome computation. For every nonzero entry $B_{i,j} = 1$, apply a CNOT from Y_i to S_j. The full network is the collection of such CNOTs over the Tanner-graph edges of (B, v).

Gate-level realization. Equation (4.19) is a linear reversible transformation over $\mathbb{F}_2$ and can be implemented using CNOTs that accumulate parities into S. Writing $(B^T y)_j = \bigoplus_{i=1}^{m} B_{i,j} y_i$, the circuit performs, for each pair (i, j) with $B_{i,j} = 1$, a CNOT with control Y_i and target S_j 4.6:

$$\forall (i, j) \text{ s.t. } B_{i,j} = 1 : \quad \mathrm{CNOT}(Y_i \to S_j). \tag{4.21}$$

Since CNOT implements $s_j \leftarrow s_j \oplus y_i$, composing these updates over all ones in B yields $s \leftarrow s \oplus B^T y$ exactly.

4.4 Coherent bounded-distance decoding

After phase and constraint encoding (Section 4.3), the joint state of the error register Y and syndrome register S takes the form

$$|\psi_3\rangle = \sum_{\substack{y \in \{0,1\}^m \\ \mathrm{wt}(y) \le \ell}} \alpha_y \, (-1)^{v \cdot y} \, |y\rangle_Y \, |s(y)\rangle_S \,, \qquad s(y) := B^T y \in \mathbb{F}_2^n, \tag{4.22}$$

where α_y is inherited from the Dicke-layer superposition and $B \in \mathbb{F}_2^{m \times n}$ is the Max-XORSAT constraint matrix. The role of the decoder is to invert the linear map $s = B^T y$ *under the promise* that the true y has small Hamming weight (bounded-distance / low-weight syndrome decoding), i.e.,

$$\text{given } s \in \mathbb{F}_2^n, \ \text{ find } \hat{y} \in \mathbb{F}_2^m \text{ such that } B^T \hat{y} = s \ \text{ and } \ \mathrm{wt}(\hat{y}) \le \ell. \tag{4.23}$$

DQI leverages the fact that many powerful classical decoding primitives exist for sparse parity-check matrices (e.g., belief propagation for LDPC-like structure) and that its performance can be understood in terms of the empirical success of the chosen decoder. The central implementation requirement for DQI, however, is that decoding must be performed *coherently* on a superposition.

Coherent decoder interface. We implement decoding as a reversible map that (optionally) writes a candidate error pattern into an output register $\widehat{Y}$ together with a success flag F:

$$U_{\mathrm{dec}} : \ |s\rangle_S \, |0^m\rangle_{\widehat{Y}} \, |0\rangle_F \longmapsto |s\rangle_S \, |\hat{y}(s)\rangle_{\widehat{Y}} \, |f(s)\rangle_F \,, \tag{4.24}$$

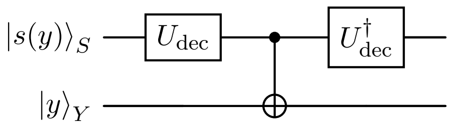


Figure 4.7: Decode–cancel–uncompute pattern. The decoder is applied in-place to map $s(y)$ to $\hat{y}$, then CNOTs cancel the error pattern in Y (when successful), and finally $U_{\text{dec}}^\dagger$ restores the syndrome register.

where $f(s) = 1$ indicates "decoding success" (e.g., $\text{wt}(\hat{y}(s)) \leq \ell$ and $B^T\hat{y}(s) = s$) and $f(s) = 0$ indicates failure.

Decode–cancel–uncompute pattern (in-place use of the decoder). In the DQI pipeline we often use the decoder as an *in-place* transform on the syndrome register, so that (on successful instances) it maps the syndrome into the corresponding low-weight error pattern:

$$U_{\text{dec}} : \ |s\rangle_S \longmapsto |\hat{y}(s)\rangle_S \,, \qquad \text{(with any work/flags kept reversible).} \tag{4.25}$$

Applied to (4.22), this yields

$$\sum_y \alpha_y (-1)^{v\cdot y} |y\rangle_Y |s(y)\rangle_S \xrightarrow{U_{\text{dec}}} \sum_y \alpha_y (-1)^{v\cdot y} |y\rangle_Y |\hat{y}(s(y))\rangle_S \,. \tag{4.26}$$

We then apply bitwise CNOTs from S (now holding $\hat{y}$) into Y to "cancel" the error pattern:

$$|y\rangle_Y |\hat{y}\rangle_S \xrightarrow{\text{CNOT}(S\to Y)} |y \oplus \hat{y}\rangle_Y |\hat{y}\rangle_S \,. \tag{4.27}$$

When decoding succeeds and $\hat{y}(s(y)) = y$, the error register is mapped to $|0^m\rangle_Y$. Finally, we apply $U_{\text{dec}}^\dagger$ on S to return $|\hat{y}\rangle_S$ back to $|s\rangle_S$:

$$|y \oplus \hat{y}\rangle_Y |\hat{y}\rangle_S \xrightarrow{U_{\text{dec}}^\dagger} |y \oplus \hat{y}\rangle_Y |s(y)\rangle_S \,. \tag{4.28}$$

This "decode–cancel–uncompute" sequence uses the decoder coherently while restoring the syndrome register, and it is particularly convenient for integrating different decoding strategies into a single end-to-end circuit.

We implemented three decoding strategies under this coherent interface: a lookup-table decoder (baseline), a reversible Gauss–Jordan elimination (GJE) decoder, and a BPQM-based message-passing decoder. The remainder of this section describes each implementation.

4.4.1 Lookup-table decode

For small parameter regimes, we implement a baseline decoder as a reversible lookup table [49]. Offline, we enumerate all error patterns $y \in \{0,1\}^m$ and compute their syndromes

$$s = B^T y \pmod 2 \in \{0,1\}^n, \tag{4.29}$$

thereby constructing a syndrome-to-pattern map

$$\mathcal{T}: \{0,1\}^n \to \{0,1\}^m, \qquad s \mapsto \hat{y}(s). \tag{4.30}$$

When multiple y yield the same syndrome s (which can occur in general), we store a single deterministic representative, e.g., the first encountered under a fixed enumeration order. The coherent application of this table is performed by a sequence of pattern-match-and-write steps, summarized in Algorithm 1.

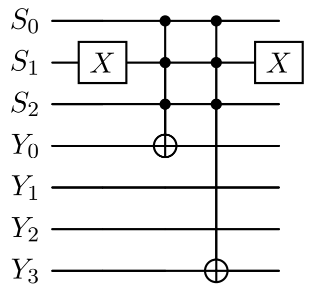


Figure 4.8: Example of a single table entry (s, e) implemented as an n-bit pattern match followed by conditional writes. Here $s = 101$, so we temporarily apply X on S_1 to convert the condition $S = s$ into an all-ones control. Then we apply C^3X onto the output bits where e has 1 (here $e = 1001$, so targets are Y_0 and Y_3), and finally uncompute the temporary X on S_1.

Circuit idea: n-bit pattern match + m conditional writes. For each table entry $(s, e) \in \mathcal{T}$, we implement the conditional map $|s\rangle_S |0^m\rangle_Y \mapsto |s\rangle_S |e\rangle_Y$ using n-controlled Pauli-X gates, denoted C^nX (i.e., the target flips iff all n controls are $|1\rangle$); see Fig. 4.8 for an explicit example. First, we turn the equality test "$S = s$" into an all-ones control condition by applying X to each syndrome qubit S_i with $s_i = 0$. Then, conditioned on *all* n syndrome qubits, we flip each output qubit Y_j for which $e_j = 1$ by applying $C^nX(S[0..n-1] \to Y[j])$. Equivalently, a single table entry is realized by up to m gates of the form C^nX sharing the same n controls but acting on different targets. Finally, we uncompute the temporary X gates on S to restore the syndrome register. Each C^nX can be decomposed into elementary gates using standard multi-controlled Toffoli constructions (with or without ancillas), as studied extensively in the literature.

Algorithm 1 Lookup-table syndrome decoder

```
Require: Syndrome register $S \in \{0,1\}^n$, output register $Y$ initialized to $|0^m\rangle$, and a
    table $\mathcal{T} \subseteq \{0,1\}^n \times \{0,1\}^m$ with *at most one* entry per syndrome $s$.
Ensure: If $(s,e) \in \mathcal{T}$ matches the current syndrome in $S$, the circuit flips $Y$ to $|e\rangle$ (via
    XOR into $Y$).
 1: for all $(s,e) \in \mathcal{T}$ do
 2:     for $i \leftarrow 0$ to $n-1$ do
 3:         if $s_i = 0$ then
 4:             Apply $X$ to $S[i]$.
 5:         end if
 6:     end for
 7:     for $j \leftarrow 0$ to $m-1$ do
 8:         if $e_j = 1$ then
 9:             Apply $C^nX\big(S[0..n-1] \to Y[j]\big)$.
10:         end if
11:     end for
12:     for $i \leftarrow 0$ to $n-1$ do
13:         if $s_i = 0$ then
14:             Apply $X$ to $S[i]$.
15:         end if
16:     end for
17: end for
```

4.4.2 Reversible Gauss–Jordan elimination (GJE) decoder

We now describe the structured decoder used throughout our circuit-level implementation: a *reversible Gauss–Jordan elimination* (GJE) routine applied to the linear system

$$B^T y = s \quad (\text{over } \mathbb{F}_2), \qquad B^T \in \mathbb{F}_2^{n\times m}, \tag{4.31}$$

where $s \in \mathbb{F}_2^n$ is the syndrome computed in Section 4.3.2. Gauss–Jordan elimination extends Gaussian elimination by continuing past row-echelon form (REF) to produce the *reduced* row-echelon form (RREF), i.e., it eliminates nonzero entries both below and above each pivot [50, 51]. Over $\mathbb{F}_2$, row addition is XOR (denoted $\oplus$), so elimination reduces to row swaps and XOR row-additions.

In the decoder we apply elimination to the *augmented* matrix $[\,B^T \mid s\,]$, i.e., every row operation applied to B^T is also applied to the right-hand side s. Reducing to RREF is convenient because, in the square full-rank case, the left block becomes the identity and the solution vector is read off directly from the rightmost column. As a concrete *example*, consider the system $B^T y = s$ over $\mathbb{F}_2$ with

$$B^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}, \qquad s = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

Forming the augmented matrix and applying Gauss–Jordan elimination yields

$$\big[\, B^T \mid s \,\big] = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathrm{RREF}\big(\big[\, B^T \mid s \,\big]\big) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

so that y is read off as the last column (here $y = (1, 0, 0, 1, 0)^T$). More generally, if the system is underdetermined, the reduced augmented form expresses pivot variables as affine $\mathbb{F}_2$-linear functions of the free variables; adopting a fixed convention for free variables yields a canonical solution $\hat{y}(s)$.

This decoder was introduced as the coherent decoding core in our prior DQI circuit design work [36], and it follows the same circuit paradigm used in quantum-circuit Gaussian/Gauss–Jordan elimination for ISD-style constructions (row swaps via SWAP networks and row XOR-additions via CNOT networks) [52]. Concretely, we implement a quantum-circuit-friendly GJE that iterates over pivot columns, swaps in a pivot row when needed, and then clears that pivot column in all other rows using conditional row additions:

Because the Max-XORSAT constraint matrix B (hence B^T) is *fixed* for a given instance, we avoid any data-dependent branching in the quantum circuit by computing the elimination schedule *classically* once and then replaying it coherently. Importantly, since the only instance-dependent quantum data during decoding is the right-hand side (the syndrome contents), we record and replay only the row operations *as they act on the augmented/right-hand-side columns* (i.e., on the evolving syndrome/solution data), rather than maintaining a full quantum register for the left block B^T.

With this convention, we do not represent the full augmented matrix $[\, B^T \mid s \,]$ in a quantum register. Instead, we run Gauss–Jordan elimination *classically* on the fixed matrix B^T and record only the resulting sequence of elementary row operations *as they act on the right-hand-side column(s).* We then replay this recorded schedule directly on the syndrome register. Equivalently, the decoding unitary implements the map that "inverts" the constraint encoding on the syndrome register, converting the state $\left|B^T y\right\rangle$

Algorithm 2 Quantum-circuit-friendly Gauss–Jordan elimination

Require: Augmented system register A encoding $\left[\, B^T \mid s \,\right]$ over $\mathbb{F}_2$.

Ensure: (i) A reduced to (a chosen) RREF form; (ii) a recorded list $\mathcal{L}$ of row operations to replay coherently.

```
 1: $\mathcal{L} \leftarrow [\,]$
 2: for $j = 1$ to $m$ do
 3:     Find a pivot row $i$ with $A_{i,j} = 1$ (according to a fixed convention).
 4:     if $i \neq j$ then
 5:         Swap rows $i$ and $j$ (SWAP network).
 6:         Append SWAP$(i, j)$ to $\mathcal{L}$.
 7:     end if
 8:     for each row $k \neq j$ do
 9:         if $A_{k,j} = 1$ then
10:             $A_{k,*} \leftarrow A_{k,*} \oplus A_{j,*}$
11:             Append XORROW$(k \leftarrow k \oplus j)$ to $\mathcal{L}$.
12:         end if
13:     end for
14: end for
15: Output: RREF$(A)$ and the operation schedule $\mathcal{L}$.
```

back to $|y\rangle$ (square full-rank case), as described in our DQI circuit design.

$$U_{\mathrm{GJE}} : \ \left|B^T y\right\rangle_S \ \longmapsto \ |y\rangle_S \,. \tag{4.32}$$

After obtaining $|y\rangle$ on the syndrome register, we use CNOTs to uncompute the error register and, when needed, apply $U_{\mathrm{GJE}}^{\dagger}$ to return $|y\rangle_S$ back to $\left|B^T y\right\rangle_S$ as part of the overall DQI uncomputation flow.

4.4.3 Belief Propagation with Quantum Messages (BPQM) decoder

Belief propagation (BP) is a local message-passing procedure for computing marginal probabilities in graphical models whose joint distribution factorizes into small terms [23, 53, 54]. On a *factor graph* (or equivalently, a Tanner graph for linear codes) [22], BP iteratively exchanges "messages" along edges between variable nodes and factor/check nodes; each message is a function of the neighboring variable (often represented as probabilities or log-likelihood ratios (LLRs)) summarizing the sender's current belief about that variable given all information *except* what came from the recipient.

In coding theory, a binary linear code specified by a parity-check matrix can be represented as a Tanner graph with variable nodes (codeword bits) and check nodes (parity constraints). Given channel outputs, the decoding task is to infer the posterior bit marginals (or an MAP estimate) under these parity constraints. BP performs this inference efficiently because each update depends only on local neighborhoods; one iteration costs $\mathcal{O}(|E|)$ message updates where $|E|$ is the number of Tanner-graph edges, which is linear in the blocklength for sparse codes such as LDPC codes [55]. BP is *exact* on tree-structured factor graphs (it produces the true marginals after a finite number of passes), while on graphs with cycles ("loopy BP") it becomes an approximation that may not converge but is empirically highly effective for many sparse-graph codes [56].

BPQM is a quantum adaptation in which the "messages" are *qubits* (quantum states) rather than classical likelihoods, enabling a structured *collective* decoding measurement for classical information sent over pure-state classical–quantum (CQ) channels [57]. Despite the name, BPQM is *not* "belief propagation" in the classical sense: it does not propagate scalar beliefs such as probabilities or log-likelihood ratios (LLRs). Instead, the information that would be summarized by a classical belief message is carried implicitly in a *quantum message state* (and, in some formulations, accompanied by a small amount of classical side information that controls the appropriate unitary/message-combination steps) [58]. This "quantum message passing" viewpoint is more than a formal analogy: on *tree* Tanner graphs over pure-state CQ channels, BPQM can implement the *optimal collective* (block) measurement for discriminating the transmitted *codeword* state, whereas classical BP is inherently a bitwise inference procedure built around marginal/bit-MAP beliefs from LLR messages. Another advantage of this approach: on tree Tanner graphs over pure-state CQ channels, BPQM is block-optimal (optimal joint codeword measurement), while classical BP is generally only bitwise-optimal.

A direct "quantization" of classical BP would require coherently representing and updating real-valued belief messages (typically LLRs) and repeatedly performing the corresponding nonlinear check-node/variable-node update rules across the Tanner graph. Even classically, LLR-BP decoders invest substantial effort in numerically stable representations of these updates and their approximations. In contrast, BPQM sidesteps explicit LLR arithmetic: it keeps the relevant information in quantum states and realizes the update/combine steps by composing unitaries dictated by the factor-graph structure. This makes BPQM a more natural candidate when the decoding subroutine itself must be embedded coherently inside a larger quantum algorithm such as DQI.

From Tanner graph to message-passing graph (MPG). BPQM decodes a chosen *target* bit by unrolling the factor graph / Tanner graph into a directed, tree-like *message-passing graph* (MPG) rooted at that target (Fig. 4.9). Unrolling duplicates nodes/edges as needed to break cycles, yielding an acyclic computation structure toward the root. Internal MPG nodes are of two types: (i) *check* nodes enforcing parity constraints ("+"), and (ii) *equality* nodes identifying duplicated variables ("="). On graphs with cycles, unrolling can create repeated leaves; practical BPQM variants address this by (approximate) cloning to supply multiple copies of the same leaf message [58].

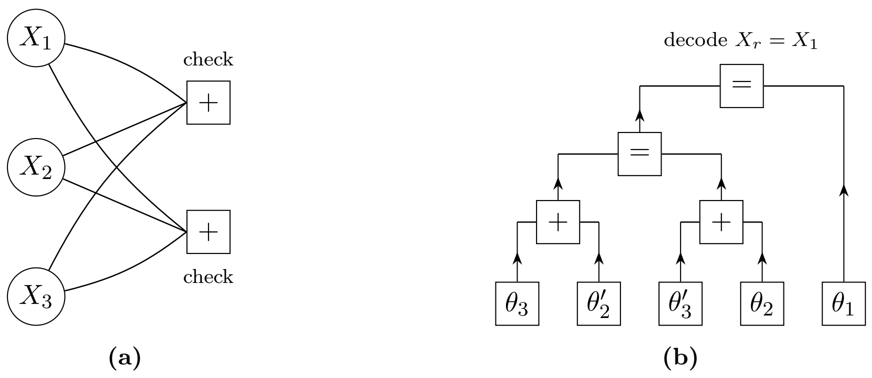


Figure 4.9: (a) Example factor graph/Tanner graph with variable nodes X_1, X_2, X_3 and check nodes "+". (b) Unrolled message-passing graph (MPG) rooted at $X_r = X_1$ with check nodes "+" and equality nodes "=". Primes denote duplicated (cloned) variables created by unrolling, and leaf rectangles show the BPQM leaf initialization labels θ_j associated with the channel outputs.

Leaf-state initialization. In the standard pure-state CQ-channel setting, each variable node corresponds to a transmitted bit $x_j \in \{0,1\}$ that induces one of two non-orthogonal output states. Following the BPQM parameterization, we represent the corresponding leaf message by a single-qubit state

$$|Q(x_j, \theta_j)\rangle \;=\; \cos\!\left(\tfrac{\theta_j}{2}\right)|0\rangle \;+\; (-1)^{x_j}\sin\!\left(\tfrac{\theta_j}{2}\right)|1\rangle\,, \tag{4.33}$$

where $\theta_j \in (0,\pi)$ determines the overlap of the two hypotheses. For *syndrome-only* decoding (our DQI use-case), we typically do not have per-bit CQ observations. Instead, we place a prior at each leaf by choosing a bias $p_j := \Pr(y_j = 1)$ and initializing

$$\left|Q(0,\tilde{\theta}_j)\right\rangle \quad \text{with} \quad \sin^2\!\left(\tfrac{\tilde{\theta}_j}{2}\right) = p_j, \tag{4.34}$$

so that the prior is biased toward low weight (e.g., small p_j) when decoding within a bounded-distance radius.

Upward quantum message passing (local unitaries). Messages are propagated from leaves toward the root. Once an internal node has received two incoming qubit messages, it applies a node-type-dependent unitary, producing one outgoing *data* qubit (forwarded toward the root) and retaining one *ancilla* qubit:

1. **Check node (+):** apply a CNOT with the first predecessor as control and the second as target; forward the control as the data message and keep the target as an ancilla.

2. **Equality node (=):** apply a two-qubit unitary $U_{\circledast}(\alpha,\beta)$ to the incoming data qubits (with controls determined by ancillas produced at preceding check nodes); forward the first output as data and keep the second as a new ancilla.

The equality-node unitary can be written as

$$U_{\circledast}(\alpha,\beta) = \begin{pmatrix} a_+ & 0 & 0 & a_- \\ -a_- & 0 & 0 & a_+ \\ 0 & b_- & b_+ & 0 \\ 0 & b_+ & -b_- & 0 \end{pmatrix}, \tag{4.35}$$

with coefficients

$$a_\pm = \frac{1}{2}\,\frac{\cos\!\left(\frac{\alpha-\beta}{2}\right) \pm \cos\!\left(\frac{\alpha+\beta}{2}\right)}{\left|\cos\!\left(\frac{\alpha\circledast\beta}{2}\right)\right|}, \tag{4.36}$$

$$b_\pm = \frac{1}{2}\,\frac{\sin\!\left(\frac{\alpha+\beta}{2}\right) \pm \sin\!\left(\frac{\alpha-\beta}{2}\right)}{\left|\sin\!\left(\frac{\alpha\circledast\beta}{2}\right)\right|}, \tag{4.37}$$

where the "$\circledast$" angle-combination is

$$\alpha \circledast \beta \;=\; \arccos\!\big(\cos\alpha \cdot \cos\beta\big). \tag{4.38}$$

Root decision and coherent readout. In the standard BPQM procedure, the root applies a final single-qubit basis change followed by a binary decision measurement. In our DQI integration we avoid any mid-circuit measurement: instead, we implement the same root basis change *coherently* and transfer the resulting decision into the designated error/output register so that the decoder can be run *reversibly* and subsequently uncomputed as part of the overall DQI pipeline.

Syndrome decoding via BPQM (DQI adaptation). In the standard BPQM setting, the objective is codeword recovery given channel outputs; in contrast, in DQI we are given a measured syndrome s and seek an error pattern y consistent with it. The modification is in the parity constraints: each check node enforces

$$\bigoplus_{i\in N(\ell)} y_i \;=\; s_\ell, \tag{4.39}$$

rather than 0. Operationally, this can be incorporated by conditioning the check-node action on s_ℓ (equivalently, inserting an X flip on one incoming message line when $s_\ell = 1$), while leaving the upward message-passing rules and root readout unchanged. Figure 4.10 shows the MPG structure we use in this syndrome-only setting.

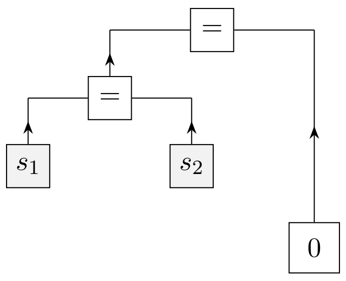


Figure 4.10: Unrolled MPG for syndrome decoding: check nodes are replaced by fixed syndrome boxes s_1, s_2; leaves are initialized with prior angles (example shown: 0).

BPQM as a quantum circuit. BPQM can be realized as an explicit quantum circuit by first unrolling the Tanner graph into an MPG rooted at a chosen target bit x_r, and then translating each MPG node into a local circuit gadget. Executing these gadgets in a fixed upward schedule (from leaves to root) defines a single global unitary, denoted V_r. Concretely, V_r is the ordered product of the node-update unitaries applied during message propagation,

$$V_r \;=\; U_T\, U_{T-1} \cdots U_2\, U_1, \tag{4.40}$$

where each U_t corresponds to one check-node or equality-node update acting on the currently available *data/message* qubits together with any ancilla qubits created/retained by earlier updates. After applying V_r, the information relevant for deciding x_r is concentrated onto a designated *root* qubit (up to a known single-qubit basis change); performing that basis change and measuring would yield the standard BPQM decision rule, while omitting any measurement leaves the coherent decoding unitary V_r itself. Because V_r is unitary (hence reversible), it can be embedded as a coherent subroutine and later uncomputed when composing BPQM inside a larger circuit.

Integration into DQI. In our DQI pipeline we avoid mid-circuit measurement because the decoder must remain reversible and be uncomputed later. Concretely, after applying the single-bit BPQM unitary V_r, we *coherently* transfer the root decision into the designated error/output register (e.g., by a CNOT from the root qubit to $\hat{y}_r$ in the appropriate basis), and then apply $V_r^\dagger$ to restore the BPQM working registers to their pre-decoding state. This decode–write–uncompute pattern (Fig. 4.11) allows BPQM to be composed as a coherent subroutine inside the larger DQI circuit. Moreover, it enables *successive* extraction of multiple bits: after writing $\hat{y}_r$, we proceed to the next target bit by running the corresponding decoding unitary V_{r+1} (and its inverse) and writing $\hat{y}_{r+1}$, and so on, analogous to the complete-codeword BPQM circuits built from alternating V_j and $V_j^\dagger$ blocks.

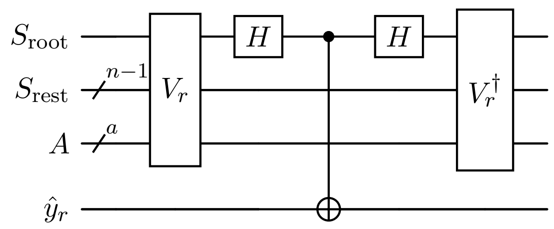


Figure 4.11: Coherent BPQM integration using register bundles. The BPQM unitary V_r acts on (S, A), we coherently apply the root decision into the error bit $\hat{y}_r$, then apply $V_r^\dagger$ to uncompute and restore (S, A).

4.5 Quantum Fourier Transform

The quantum Fourier transform (QFT) [59] is the unitary change-of-basis that implements the discrete Fourier transform on the amplitude vector of a quantum state. Concretely, for $N = 2^n$ it maps computational basis states as

$$\mathrm{QFT}_N : \ |z\rangle \ \longmapsto \ \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{2\pi i x z/N} \, |x\rangle \,, \tag{4.41}$$

and extends linearly to superpositions.

Its importance is that it serves as a *core primitive* in many quantum algorithms—most famously quantum phase estimation [60] and Shor's factoring/discrete-log algorithms [13] —and more generally in Fourier-sampling and hidden-structure routines [61]. In DQI, the Fourier-basis change is the key ingredient that turns the original optimization task into a *decoding* task.

After coherent bounded-distance decoding (Section 4.4) and uncomputation, the low-weight pattern register Y is returned to $|0^m\rangle$, leaving only the n-qubit syndrome register. In the binary Max-XORSAT specialization, this remaining register stores

$$s(y) = B^T y \in \mathbb{F}_2^n.$$

Following the original DQI pipeline, we then apply a Fourier transform to the syndrome register to obtain the final output sampling distribution.

QFT in the binary setting. Throughout this section we focus on the *binary* Max-XORSAT specialization, so the Fourier transform we need is the QFT over the Boolean group $(\mathbb{Z}/2\mathbb{Z})^n$. The characters of this group are $\chi_x(z) = (-1)^{x \cdot z}$ for $x, z \in \{0,1\}^n$. The quantum Fourier transform on $(\mathbb{Z}/2\mathbb{Z})^n$ is therefore exactly the *Walsh–Hadamard transform*, implemented by a Hadamard on each qubit:

$$\mathrm{QFT}_{(\mathbb{Z}/2\mathbb{Z})^n} \ = \ H^{\otimes n}. \tag{4.42}$$

Equivalently, for any computational basis state $|z\rangle$,

$$H^{\otimes n}|z\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot z} |x\rangle, \qquad x\cdot z := \bigoplus_{i=1}^{n} x_i z_i. \tag{4.43}$$

Thus, in our implementation the "QFT step" is a single depth-1 layer of Hadamard gates.

State evolution. Let X denote the n-qubit register holding the syndrome value $s(y) = B^T y$ after the decode–cancel–uncompute pattern. The pre-transform state on X has the generic form

$$\left|\widetilde{P}(f)\right\rangle_X = \sum_{\substack{y\in\{0,1\}^m\\ \mathrm{wt}(y)\le\ell}} \alpha_y (-1)^{v\cdot y} |s(y)\rangle_X, \qquad s(y) = B^T y, \tag{4.44}$$

where the coefficients α_y capture the prepared low-weight superposition (including the w_k weights from the Dicke-layer construction).

Applying the Boolean-group QFT, i.e. $H^{\otimes n}$, yields the post-transform (polynomial) state

$$\begin{aligned} |P(f)\rangle_X \;:=\; H^{\otimes n}\left|\widetilde{P}(f)\right\rangle_X &= \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} \left(\sum_{\substack{y\in\{0,1\}^m\\ \mathrm{wt}(y)\le\ell}} \alpha_y (-1)^{v\cdot y} (-1)^{x\cdot s(y)} \right) |x\rangle \\ &= \sum_{x\in\{0,1\}^n} P(f(x))\, |x\rangle, \end{aligned} \tag{4.45}$$

where we define the (generally unnormalized) amplitude

$$P(f(x)) := \frac{1}{\sqrt{2^n}} \sum_{\substack{y\in\{0,1\}^m\\ \mathrm{wt}(y)\le\ell}} \alpha_y (-1)^{v\cdot y + x\cdot(B^T y)}. \tag{4.46}$$

Thus the Hadamard layer induces interference: each output amplitude $P(f(x))$ is a coherent sum over low-weight patterns y with phases determined by the instance data and the Fourier kernel.

4.6 Measurement and postselection

At the end of the QFT step (Section 4.5), the remaining n-qubit register X is in the polynomial state

$$|P(f)\rangle_X = \sum_{x\in\{0,1\}^n} P(f(x))\, |x\rangle, \tag{4.47}$$

up to normalization, where P is the degree-ℓ polynomial defined by the chosen weight profile. Measuring X in the computational basis yields an output string x with probability proportional to $|P(f(x))|^2$, thereby biasing samples toward higher objective values relative to uniform sampling.

Postselection register and success event. In DQI, the decoder is used *coherently* to uncompute the low-weight pattern register. Ideally, for every syndrome value that arises from the intended low-weight superposition, bounded-distance decoding succeeds and the subsequent cancellation/uncomputation returns the m-qubit error register Y to $|0^m\rangle$, disentangling it from the rest of the computation. We therefore define the acceptance (success) event as

$$\mathsf{Acc} := (Y = 0^m) \tag{4.48}$$

Operationally, we measure Y (or F) and postselect on outcomes satisfying (4.48), using only accepted shots to form the output sample distribution on X.

In the theoretical reduction, postselection is benign because the decoder is assumed to work correctly on the relevant (thresholded/low-weight) portion of the superposition. In our circuit-level implementation, however, the decoder can be *imperfect* (e.g., due to heuristic pivoting conventions, approximate message passing, or other practical deviations from the ideal bounded-distance guarantee). In that case, some amplitude components fail to uncompute and leave $Y \neq 0^m$; those branches would otherwise contaminate the intended interference on X. Postselection suppresses this effect by projecting onto the "decode-succeeded" subspace, at the cost of a success probability $p_{\text{succ}} := \Pr[\mathsf{Acc}] < 1$ that determines the repetition overhead.

Chapter 5

Resource Estimation and Results

This chapter emphasizes the *resource cost* of each subroutine—qubits, gate counts, and circuit depth. For each DQI circuit component, we derive resource estimates (qubit counts, gate counts, and depth) under our chosen decompositions, and we summarize the overall scaling for the full pipeline. We then complement these estimates with small-instance simulations, verifying that after the decode–cancel–uncompute step and postselection, the measured distribution on X concentrates on high-value assignments, consistent with the DQI mechanism of decoding-assisted Fourier interference.

5.1 Metrics

We characterize each circuit block using three metrics:

1. **Gate counts:** the number of occurrences of each gate type in the working gate set

   $$\mathcal{G} = \{X,\ Z,\ H,\ R_y,\ \text{CNOT},\ C^nX,\ \text{SWAP}\}.$$

   We report counts at this level because these gates appear as the *natural primitives* in our designed circuit components. Moreover, some of these operations (notably C^nX) admit multiple decompositions with different ancilla requirements and depth, so leaving them explicit avoids committing to a particular synthesis strategy prematurely.

2. **Circuit depth:** the length of the critical path under an idealized parallel schedule, i.e., gates acting on disjoint qubits are assumed to run simultaneously.

3. **Qubit count:** the number of logical qubits required by the block, including any algorithmic ancillas and workspace registers (but excluding hardware-dependent routing overhead).

5.2 Asymptotic resource scaling by component

In this section, we examine how the resource cost of each DQI circuit component scales with the instance parameters. Concretely, we express gate counts (in the working gate set $\mathcal{G}$ from Section 5.1), circuit depth, and qubit requirements as asymptotic functions of (n, m, ℓ).

5.2.1 Unary Amplitude Encoding (UAE)

The UAE block prepares the weight superposition specified by $\{w_k\}_{k=0}^{\ell}$ using a unary representation of k. The weights w_k are computed offline (classically), and the circuit loads them by a fan-out / ladder construction implemented as a sequence of controlled-R_y rotations along the unary register (Fig. 4.2) [36]. Since our resource accounting uses the primitive gate set $\mathcal{G} = \{X, Z, H, R_y, \mathrm{CNOT}, C^nX, \mathrm{SWAP}\}$, we decompose each controlled-$R_y(\theta)$ gate into single-qubit R_y rotations and CNOTs using the standard identity shown in Fig. 5.1a. In the binary Max-XORSAT implementation, UAE is the first stage that fixes the desired amplitude profile over Hamming-weight layers up to ℓ.

Gate counts. UAE uses only R_y and CNOT gates. The ladder construction applies a constant number of R_y rotations and CNOTs per unary step, so the gate counts scale linearly with ℓ:

$$N_{R_y} = 2\ell + 1, \qquad N_{\mathrm{CNOT}} = 2\ell, \tag{5.1}$$

Depth. The UAE circuit forms a repeating ladder pattern which, after decomposing controlled-R_y gates, can be viewed as interleaved layers of R_y and CNOT gates. Naively, each step contributes a serial pattern

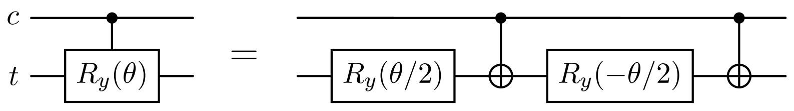
$$R_y \;\to\; \mathrm{CNOT} \;\to\; R_y \;\to\; \mathrm{CNOT},$$

suggesting depth 4ℓ; however, the first R_y rotations across different qubits can be grouped into a single parallel layer. After this compaction, each subsequent step contributes three sequential layers (CNOT, R_y, CNOT), giving

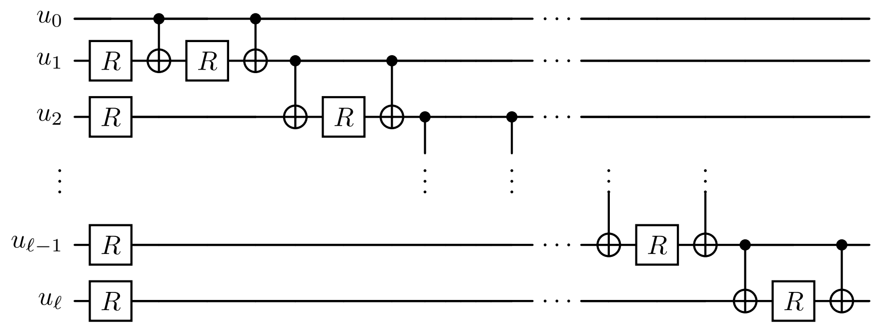
$$D_{\mathrm{UAE}} = 3\ell + 1, \tag{5.2}$$

under our idealized parallel scheduling model [36].

Qubits. UAE is performed in-place on the existing register used to hold the unary weight superposition (a subset of the m-qubit error register), and it requires no additional workspace ancillas. Thus, the UAE block adds no extra logical qubits beyond the registers already allocated for the full DQI circuit [36].



(a) A controlled-$R_y(\theta)$ gate and a decomposition into R_y + CNOT.



(b) UAE ladder compiled into R and CNOT gates. Here each R denotes an $R_y(\pm\theta_k/2)$ rotation: for ladder step k, the target line applies $R_y(\theta_k/2)$ before the first CNOT and $R_y(-\theta_k/2)$ before the second, implementing a controlled-$R_y(\theta_k)$.

Figure 5.1: Unary Amplitude Encoding (UAE) ladder and its decomposition into the primitive gate set.

5.2.2 Deterministic Dicke-state preparation

After UAE, we apply a fixed deterministic unitary $U_{m,\ell}$ that maps each unary basis state $|u_k\rangle = |1^k 0^{m-k}\rangle$ (already prepared on the m-qubit error register Y) to the corresponding Dicke state $|D_{m,k}\rangle$, thereby producing the low-weight Dicke-layer superposition $\sum_{k=0}^{\ell} w_k |D_{m,k}\rangle$ without any additional weight register (Section 4.2, Eq. (4.10)).

SCS-block structure. We adopt the deterministic construction of Bärtschi and Eidenbenz [43], in which $U_{m,\ell}$ is expressed as a sequence of Split-and-Cyclic-Shift (SCS) blocks (Fig. 4.3). The circuit consists of two stages: (i) $\mathrm{SCS}_{q,\ell}$ for $q = m, m-1, \ldots, \ell+1$ (count $m-\ell$), followed by (ii) $\mathrm{SCS}_{q,q-1}$ for $q = \ell, \ell-1, \ldots, 2$ (count $\ell-1$). Hence the total number of SCS blocks is

$$N_{\mathrm{SCS}} = (m-\ell) + (\ell-1) = m-1. \tag{5.3}$$

Each $\mathrm{SCS}_{n,k}$ decomposes into $(k-1)$ copies of a routing primitive SCS_3 and one splitting primitive SCS_2 (Section 4.2). Therefore, the total number of SCS_2 primitives is simply $N_{\mathrm{SCS}_2} = m - 1$, while the number of SCS_3 primitives is

$$\begin{aligned} N_{\mathrm{SCS}_3} &= \sum_{q=\ell+1}^{m} (\ell - 1) \; + \; \sum_{q=2}^{\ell} (q - 2) \\ &= (m - \ell)(\ell - 1) \; + \; \frac{(\ell - 1)(\ell - 2)}{2}. \end{aligned} \tag{5.4}$$

Compiled primitive costs (gate set $\mathcal{G}$). We count gates after decomposing each controlled-R_y in the SCS primitives into R_y and CNOT (Fig. 5.1a), consistent with our working gate set $\mathcal{G}$. With this compilation, the SCS primitives used in Fig. 4.4 contribute:

$$\mathrm{SCS}_2: \quad N_{R_y} = 2, \quad N_{\mathrm{CNOT}} = 4, \tag{5.5}$$

$$\mathrm{SCS}_3: \quad N_{R_y} = 6, \quad N_{\mathrm{CNOT}} = 10, \tag{5.6}$$

and no $X, Z, H, C^n X$, or SWAP gates. (The different costs reflect the additional routing logic required by SCS_3 compared to the basic two-qubit amplitude split in SCS_2.)

Gate counts. Summing Eqs. (5.5)–(5.6) over the total numbers of primitives gives

$$\begin{aligned} N_{R_y} &= 2\, N_{\mathrm{SCS}_2} + 6\, N_{\mathrm{SCS}_3} \\ &= 2(m - 1) + 6 \left[(m - \ell)(\ell - 1) + \frac{(\ell - 1)(\ell - 2)}{2} \right] \\ &= 6m\ell - 4m - 3\ell^2 - 3\ell + 4, \end{aligned} \tag{5.7}$$

$$\begin{aligned} N_{\mathrm{CNOT}} &= 4\, N_{\mathrm{SCS}_2} + 10\, N_{\mathrm{SCS}_3} \\ &= 4(m - 1) + 10 \left[(m - \ell)(\ell - 1) + \frac{(\ell - 1)(\ell - 2)}{2} \right] \\ &= 10m\ell - 6m - 5\ell^2 - 5\ell + 6. \end{aligned} \tag{5.8}$$

As expected, the leading scaling is $N_{R_y} = \Theta(m\ell)$ and $N_{\mathrm{CNOT}} = \Theta(m\ell)$ for fixed low-weight cutoff $\ell \ll m$, with negative ℓ^2 corrections due to the shorter refinement stage.

Depth. The SCS blocks in Fig. 4.3 act on overlapping subsets of Y, so we conservatively schedule them serially. Using a compiled depth bound of at most $16k - 10$ for one $\mathrm{SCS}_{n,k}$ block under our decomposition (dominated by the $(k-1)$ routing primitives), and upper-bounding $k \leq \ell$ for all blocks, we obtain

$$D_{\mathrm{Dicke}} \; \leq \; (m - 1)\,(16\ell - 10). \tag{5.9}$$

This is a worst-case bound; modest constant-factor improvements are possible with local commutation/parallelization, but do not change the $\Theta(m\ell)$ scaling in our setting.

Qubits. Deterministic Dicke preparation is performed entirely in-place on the m-qubit error register Y and requires no workspace ancillas (Section 4.2). Thus it introduces no additional logical qubits beyond the existing register allocation:

$$Q_{\text{Dicke}} = m. \tag{5.10}$$

5.2.3 Problem encoding: phase and constraint encoding

After preparing the low-weight Dicke-layer superposition on the m-qubit error register Y, the next stage encodes the Max-XORSAT instance information into the quantum state. In our implementation (Chapter 4), this *problem encoding* is applied in the order

$$\text{phase encoding} \;\rightarrow\; \text{constraint encoding (syndrome computation)}.$$

Phase encoding. The phase-encoding unitary applies a sign $(-1)^{v\cdot y}$ to each computational basis state $|y\rangle_Y$,

$$U_{\text{ph}}: \; |y\rangle_Y \longmapsto (-1)^{v\cdot y} |y\rangle_Y \,, \tag{5.11}$$

for a fixed classical binary vector $v \in \{0,1\}^m$ determined by the instance. Since v is fixed, U_{ph} is implemented by applying a single-qubit Z on Y_i whenever $v_i = 1$. Thus the phase-encoding gate count is at most linear in m:

$$N_Z^{(\text{ph})} \leq m, \qquad N_{\text{CNOT}}^{(\text{ph})} = 0, \qquad N_{R_y}^{(\text{ph})} = 0. \tag{5.12}$$

All Z gates commute and can be applied in parallel, so the depth is constant,

$$D_{\text{ph}} = 1, \tag{5.13}$$

and no additional qubits are required beyond the existing m qubits in Y.

Constraint encoding (reversible syndrome computation). Next, we compute the induced syndrome associated with y under the constraint matrix B using the reversible map

$$U_{\text{syn}}: \; |y\rangle_Y |s\rangle_S \longmapsto |y\rangle_Y \left|s \oplus (B^T y)\right\rangle_S \,, \tag{5.14}$$

where $B^T \in \mathbb{F}_2^{n\times m}$ and the syndrome register S consists of n qubits initialized to $|0^n\rangle$. Equation (5.14) can be realized as a CNOT network: for each nonzero entry $(B^T)_{j,i} = 1$, apply CNOT$(Y_i \to S_j)$.

The exact number of CNOTs equals the number of ones in B^T. Since any binary $n \times m$ matrix has at most nm ones, we obtain the parameter-only upper bound

$$N_{\text{CNOT}}^{(\text{syn})} \leq nm, \qquad N_Z^{(\text{syn})} = 0, \qquad N_{R_y}^{(\text{syn})} = 0. \tag{5.15}$$

The logical qubit count increases by the syndrome register:

$$Q_{\text{syn}} = m + n. \tag{5.16}$$

For depth, a conservative serial schedule yields

$$D_{\text{syn}} \le N_{\text{CNOT}}^{(\text{syn})} \le nm. \tag{5.17}$$

In practice, for sparse instances (e.g., constant row/column weights), the depth is typically much smaller than the worst-case nm bound; however, for the purpose of asymptotic resource reporting in terms of (m, n, ℓ) we will use the bound in Eq. (5.17).

Total cost of problem encoding. Combining phase and constraint encoding, the full problem-encoding stage contributes

$$N_Z^{(\text{enc})} \le m, \qquad N_{\text{CNOT}}^{(\text{enc})} \le nm, \qquad D_{\text{enc}} \le 1 + nm, \qquad Q_{\text{enc}} = m + n. \tag{5.18}$$

.

5.2.4 Coherent bounded-distance decoding

After problem encoding, the circuit holds an m-qubit error-pattern register Y and an n-qubit syndrome register S. The decoding stage applies a fully reversible, coherent bounded-distance decoder that uses S to compute a corresponding low-weight correction within the cutoff ℓ:

$$U_{\text{dec}} : \; |y\rangle_Y \, |s\rangle_S \, |0\rangle_A \longmapsto |y\rangle_Y \, |s\rangle_S \, |\text{Dec}(s)\rangle_A \,, \tag{5.19}$$

where A denotes decoder work/output qubits. In the full DQI circuit the decoder workspace is uncomputed later, so we report resources for a single application of U_{dec}.

(1) Lookup-table decoder

The baseline lookup-table (LUT) decoder follows the construction in the original DQI work: precompute all low-weight error patterns $e \in \{0,1\}^m$ with $\text{wt}(e) \le \ell$, compute their induced syndromes $s = B^T e$, and hard-code the resulting syndrome$\to$error mapping into a reversible circuit.

Number of entries. The number of distinct low-weight patterns up to radius ℓ is

$$N_{\text{pat}}(m, \ell) \; = \; \sum_{r=0}^{\ell} \binom{m}{r}. \tag{5.20}$$

Gate counts A direct reversible realization iterates over all $N_{\text{pat}}(m,\ell)$ precomputed entries $\{(s^{(t)},e^{(t)})\}_{t=1}^{N_{\text{pat}}}$. For each entry, syndrome matching on an n-bit register can be implemented by (i) applying X gates to turn controls on $|0\rangle$ into controls on $|1\rangle$, (ii) performing the conditional update, then (iii) undoing those X gates. In the worst case this uses $2n$ X gates per entry. The conditional update of an m-bit correction can be implemented by at most m n-controlled NOT gates (C^nX) per entry. Therefore,

$$N_X^{(\text{lookup})} = 2n\sum_{r=0}^{\ell}\binom{m}{r}, \tag{5.21}$$

$$N_{C^nX}^{(\text{lookup})} = m\sum_{r=0}^{\ell}\binom{m}{r}. \tag{5.22}$$

Depth. Using a conservative serial schedule over table entries, the depth per entry consists of two parallel X layers (to implement matching on controls in $|0\rangle$ and then undo), plus up to m layers of C^nX (one per target bit in the worst case). Hence,

$$D_{\text{lookup}} \;=\; (m+2)\sum_{r=0}^{\ell}\binom{m}{r}. \tag{5.23}$$

Qubits. If the LUT decoder writes the correction directly onto Y (as a conditional flip), it requires no additional asymptotic workspace beyond the existing (Y,S) registers:

$$Q_{\text{lookup}} = m+n. \tag{5.24}$$

(2) Reversible Gauss–Jordan elimination (GJE) decoder

The reversible GJE decoder implements Gauss–Jordan row operations coherently by replacing classical (i) row swaps and (ii) row XOR operations with quantum primitives. In the original (uncompiled) description, a row swap is implemented by a SWAP gate, and a row XOR is implemented by a CNOT gate. Here, since our final resource tables use only the base two-qubit gate (CNOT), we compile every SWAP into three CNOT gates:

$$\text{SWAP}(a,b) = \text{CNOT}(a\!\to\! b)\,\text{CNOT}(b\!\to\! a)\,\text{CNOT}(a\!\to\! b). \tag{5.25}$$

Gate counts. The GJE schedule proceeds over n pivot columns. For each pivot column:

1. **Pivot placement:** in the worst case, at most one row swap is required, hence at most n SWAPs total. After compiling Eq. (5.25), this contributes

$$N_{\text{CNOT,swap}}^{(\text{GJE})} \le 3n. \tag{5.26}$$

2. **Elimination:** once the pivot is placed, eliminate the remaining 1s in that pivot column by XOR-ing the pivot row into other rows. In the worst case, this is $(m-1)$ row-XORs per column, hence

$$N_{\text{CNOT,elim}}^{(\text{GJE})} \leq n(m-1). \tag{5.27}$$

3. **Final copy/uncompute step:** after elimination, one performs a final register update/uncomputation step costing at most n additional CNOTs,

$$N_{\text{CNOT,final}}^{(\text{GJE})} \leq n. \tag{5.28}$$

Summing Eqs. (5.26)–(5.28) gives the explicit CNOT-only bound

$$N_{\text{CNOT}}^{(\text{GJE})} = n(m-1) + n + 3n = n(m+3). \tag{5.29}$$

Depth. Using the same conservative serial scheduling assumption as before: per pivot column, elimination costs at most $(m-1)$ CNOT layers, and the compiled SWAP costs 3 CNOT layers. Thus each pivot column costs at most $(m+2)$ CNOT layers, and the final step costs at most n CNOT layers. Hence

$$D_{\text{GJE}} \leq n(m+2) + n = n(m+3). \tag{5.30}$$

Qubits. The reversible GJE decoder is executed in-place on the existing error and syndrome registers and does not require additional workspace qubits:

$$Q_{\text{GJE}} = m + n. \tag{5.31}$$

(3) Belief Propagation with Quantum Messages (BPQM) decoder

We now estimate the resources for a BPQM-based coherent decoder as implemented in this thesis, using the primitive gate set $\{\text{CNOT}, X, \text{CZ}, \text{UCR}_y\}$, where UCR_y denotes a uniformly-controlled R_y multiplexer (a multi-controlled R_y specified by 2^t rotation angles for t control qubits).

BPQM is executed on an *unrolled computation tree* obtained from the Tanner graph by unrolling to a fixed height h. Because the circuit size depends on the width of this computation tree, we introduce two additional parameters:

$$d := \max\{\text{node degree in the Tanner graph}\}, \qquad h := \text{chosen tree unrolling height.}$$

Check- and variable-node primitives. The BPQM computation proceeds by repeatedly combining two incoming subtrees at either a check node or a variable node.

1. **Check-node combine.** A check-node combine applies an optional syndrome-controlled phase (implemented by a CZ from the relevant syndrome qubit onto the propagated data qubit), followed by a single CNOT between the two incoming data qubits. Thus, per check-node combine we count

$$\text{per check combine:} \qquad N_{\text{CNOT}} = 1, \qquad N_{\text{CZ}} \le 1. \tag{5.32}$$

2. **Variable-node combine.** A variable-node combine applies a fixed two-qubit Clifford skeleton and two uniformly-controlled rotations:

$$4 \times \text{CNOT} \;+\; 2 \times X \;+\; 2 \times \text{UCR}_y.$$

   Hence, per variable-node combine we count

$$\text{per variable combine:} \qquad N_{\text{CNOT}} = 4, \qquad N_X = 2, \qquad N_{\text{UCR}_y} = 2. \tag{5.33}$$

Computation-tree size (bounds via d, h). Consider decoding a single root variable by unrolling the Tanner neighborhood to height h. In the worst case (all node degrees saturate the maximum d), the number of check-node occurrences in the unrolled tree is bounded by

$$N_{\text{chk}}(d, h) = d \sum_{r=0}^{h-1} (d-1)^{2r}, \tag{5.34}$$

and the number of variable-node occurrences (including the root) is bounded by

$$N_{\text{var}}(d, h) = 1 + d(d-1) \sum_{r=0}^{h-1} (d-1)^{2r}. \tag{5.35}$$

The number of leaf data qubits (leaf variable nodes at depth $2h$ in this worst-case tree) is

$$n_{\text{data}}(d, h) = d(d-1)^{2h-1}. \tag{5.36}$$

Number of combine operations. A node with k children can be reduced to one output by a binary tree of merges requiring $(k-1)$ combine operations. Using worst-case child counts in a degree-d unrolling:

1. each check node has at most $(d-1)$ children, hence requires at most $(d-2)$ check combines;

2. the root variable has at most d children (thus $d-1$ variable combines), and each internal non-root variable node has at most $(d-1)$ children (thus at most $d-2$ variable combines).

This yields explicit bounds

$$N_{\text{chk\_comb}}(d,h) = (d-2)\,N_{\text{chk}}(d,h) = d(d-2)\sum_{r=0}^{h-1}(d-1)^{2r}, \tag{5.37}$$

$$\begin{aligned} N_{\text{var\_comb}}(d,h) &= (d-1) \;+\; (d-2)\big(N_{\text{var}}(d,h)-1\big) \\ &= (d-1) \;+\; (d-2)\,d(d-1)\sum_{r=0}^{h-1}(d-1)^{2r}. \end{aligned} \tag{5.38}$$

Total primitive gate counts (one decoded root). Combining the per-combine costs in Eqs. (5.32)–(5.33) with Eqs. (5.37)–(5.38), the total primitive counts for one decoded root are

$$N_{\text{CNOT}}^{(\text{BPQM})} = N_{\text{chk\_comb}}(d,h) + 4\,N_{\text{var\_comb}}(d,h), \tag{5.39}$$

$$N_{X}^{(\text{BPQM})} = 2\,N_{\text{var\_comb}}(d,h), \tag{5.40}$$

$$N_{\text{UCR}_y}^{(\text{BPQM})} = 2\,N_{\text{var\_comb}}(d,h), \tag{5.41}$$

$$N_{\text{CZ}}^{(\text{BPQM})} \le N_{\text{chk\_comb}}(d,h) + N_{\text{chk}}(d,h). \tag{5.42}$$

Qubits. The BPQM computation tree uses $n_{\text{data}}(d,h)$ data qubits in Eq. (5.36). When embedded into DQI (which already uses $m+n$ qubits for (Y,S)), the additional qubits required by BPQM are

$$Q_{\text{BPQM,extra}} = n_{\text{data}}(d,h) = d(d-1)^{2h-1}, \qquad Q_{\text{BPQM,total}} = (m+n) + d(d-1)^{2h-1}. \tag{5.43}$$

Remark on compiling UCR$_y$**.** In our present implementation, each UCR$_y$ gate is specified by 2^t rotation angles, where t is the number of control qubits for that multiplexer. Consequently, if one compiles UCR$_y$ into a two-qubit gate set, the cost generally scales with the number of angles (and hence exponentially in t). A polynomial-time alternative proposed in the BPQM literature replaces such multiplexed rotations by a discretized message-passing implementation using a small angle register, which we leave as an important direction for future work.

5.2.5 Hadamard/QFT step

After decoding, DQI applies a Fourier-type transform on the register used for interference. In this thesis we implement this step as a layer of Hadamard gates (i.e., $H^{\otimes n}$) on an n-qubit register.

Gate count and depth. Since the Hadamard transform applies one single-qubit H gate per qubit, the total gate count is

$$N_H^{(\mathrm{QFT})} = n. \tag{5.44}$$

All Hadamard gates act on distinct qubits and can be applied in parallel, hence the depth is

$$D_{\mathrm{QFT}} = 1. \tag{5.45}$$

Qubits. The Hadamard layer does not introduce additional ancillas; it acts in-place on the existing n-qubit register:

$$Q_{\mathrm{QFT}} = n. \tag{5.46}$$

5.2.6 End-to-end scaling summary

This chapter decomposed the DQI circuit into modular components: (i) state preparation (Unary Amplitude Encoding and deterministic Dicke preparation), (ii) problem encoding (phase and constraint/syndrome encoding), (iii) a coherent bounded-distance decoding subroutine (Lookup-table, GJE, or BPQM), and (iv) the final interference transform implemented as $H^{\otimes n}$. Table 5.1 and 5.2 summarize the explicit gate counts, depth, and qubit requirements for each component in terms of the problem parameters (m, n, ℓ) and, for BPQM, the additional computation-tree parameters (d, h).

For end-to-end resource estimation, the dominant terms depend strongly on the chosen decoder: the lookup-table decoder scales with the number of low-weight patterns $\sum_{r=0}^{\ell} \binom{m}{r}$, the reversible GJE decoder scales quadratically in (m, n), and BPQM depends on the unrolled computation-tree size controlled by (d, h). The Hadamard step contributes only n single-qubit gates at depth 1 and is negligible compared to the other stages for nontrivial instances.

Table 5.1: Gate Count and Depth Complexity for DQI Circuit Components.

Gate	Unary Amp. Encoding	Dicke State Prep.	Phase Enc.	Constraint Enc.	Hadamard Trans.
X	0	0	0	0	0
Z	0	0	$\leq m$	0	0
Hadamard	0	0	0	0	n
$\mathbf{R_y}$	$2\ell+1$	$6m\ell-4m$ $-3\ell^2-3\ell+4$	0	0	0
CNOT	2ℓ	$10m\ell-6m$ $-5\ell^2-5\ell+6$	0	$\leq mn$	0
$\mathbf{C^n(X)}$	0	0	0	0	0
Depth	$3\ell+1$	$\leq (m-1)(16\ell-10)$	1	$\leq mn$	1
Qubits	$2^{\lceil \log_2(\ell+1)\rceil}$	m	m	$m+n$	n

Table 5.2: Gate Count and Depth Complexity for Coherent Decoders

Gate	Lookup Table	GJE Elim.	BPQM
X	$2n\sum_{r=0}^{\ell}\binom{m}{r}$	0	$2\,N_{\text{var\_comb}}(d,h)$
Z	0	0	0
Hadamard	0	0	0
$\mathbf{R_y}$	0	0	0
CNOT	0	$n(m+3)$	$N_{\text{chk\_comb}}(d,h)+4\,N_{\text{var\_comb}}(d,h)$
$\mathbf{C^n(X)}$	$m\sum_{r=0}^{\ell}\binom{m}{r}$	0	0
CZ	0	0	$\leq N_{\text{chk\_comb}}(d,h)+N_{\text{chk}}(d,h)$
$\mathbf{UCR_y}$	0	0	$2\,N_{\text{var\_comb}}(d,h)$
Depth	$\leq (m+2)\sum_{r=0}^{\ell}\binom{m}{r}$	$n(m+3)$	$\leq 2N_{\text{chk\_comb}}(d,h)+8N_{\text{var\_comb}}(d,h)+N_{\text{chk}}(d,h)$
Qubits	$m+n$	$m+n$	$(m+n)+d(d-1)^{2h-1}$

$$N_{\text{chk}}(d,h) = d\sum_{r=0}^{h-1}(d-1)^{2r}, \tag{5.47}$$

$$N_{\text{chk\_comb}}(d,h) = d(d-2)\sum_{r=0}^{h-1}(d-1)^{2r}, \tag{5.48}$$

$$N_{\text{var\_comb}}(d,h) = (d-1)+(d-2)\,d(d-1)\sum_{r=0}^{h-1}(d-1)^{2r}. \tag{5.49}$$

5.3 Simulation results

5.3.1 Experimental setup

We evaluate the Decoded Quantum Interferometry (DQI) algorithm on a family of binary Max-XORSAT instances, specifically MaxCut problems. Each experiment is defined by the parameters (m, n, ℓ) and a choice of coherent decoder, including reversible Gauss–Jordan elimination (GJE), a reversible lookup-table decoder, and a BPQM-based decoder. For each circuit run, we measure all registers and apply postselection, retaining only those outcomes where the error register is measured in the state $|0\rangle^{\otimes m}$. This postselection isolates the interference fringe corresponding to successful decoding, so that the remaining measurement statistics on the solution register reflect the decoded objective landscape.

5.3.2 Resource metrics for DQI circuit components

To quantify physical cost, we report gate counts (GC), circuit depth (D), and qubit count (Q) for each major subroutine after transpiling the circuit to a fixed universal gate set. Table 5.3 compares the dominant implementation blocks, separating the decoding stage into three alternatives (GJE, lookup-table, and BPQM).

Two trends are immediate. First, deterministic Dicke state preparation is the largest non-decoding cost for the smallest instances, but its growth with m remains moderate for fixed ℓ. Second, the decoder choice dominates the overall scaling: the reversible lookup-table decoder rapidly becomes the bottleneck, whereas the GJE decoder maintains substantially smaller gate counts and depth. The BPQM implementation reported here sits between these two extremes in the tested regime, but introduces additional message/data qubits and UCR$_y$ multiplexers.

Table 5.3: Resource metrics for DQI circuit components (after transpilation). GC = gate count, D = depth, Q = qubits.

Subroutine	**GC**			**D**			**Q**		
Instance size	m=5	m=10	m=15	m=5	m=10	m=15	m=5	m=10	m=15
Unary Amplitude Encoding	13	13	13	5	5	5	5	10	15
Dicke State Preparation	117	284	441	81	195	313	5	10	15
Phase Encoding	5	10	15	1	1	1	5	10	15
Constraint Encoding	10	20	30	6	5	8	10	20	30
Decoding (GJE)	7	12	41	5	7	24	5	10	15
Decoding (Lookup Table)	1885	17112	67027	1655	15778	62547	10	20	30
Decoding (BPQM)	763	2445	2775	475	1619	2040	17	31	45
Hadamard Transform	10	20	30	2	2	2	5	10	15

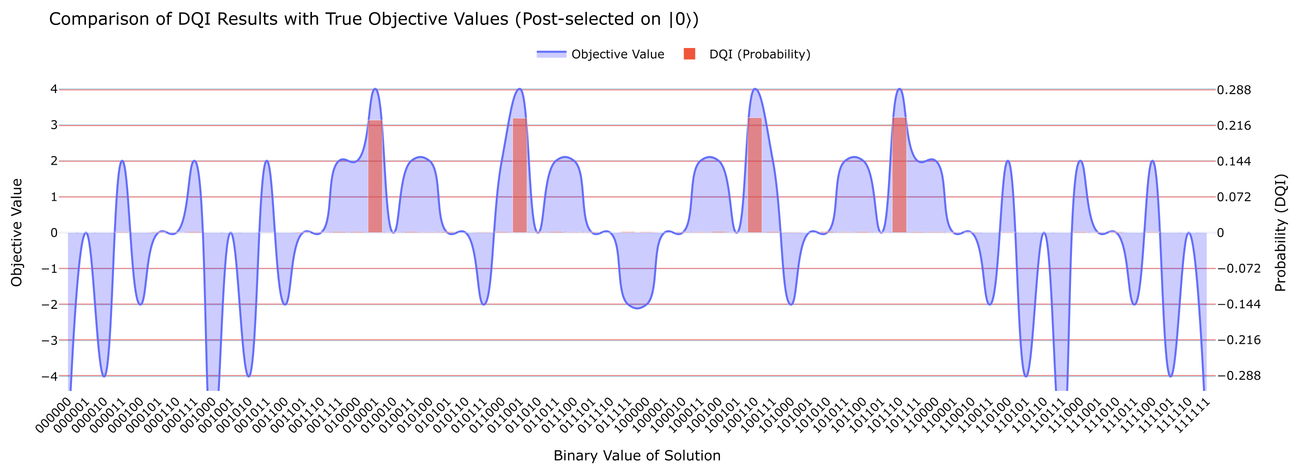


Figure 5.2: Result plot for the 6-bit instance. The x-axis lists all 64 possible solution states. The blue line (left y-axis) indicates the classical objective value for each state, and the red bars (right y-axis) show the measured probability after postselecting on $|0\rangle^{\otimes 6}$. The coincidence between objective peaks and measurement probabilities indicates that DQI biases outcomes toward optimal solutions.

5.3.3 Running example: 6-bit instance results

For a concrete illustration, we consider a 6-bit MaxCut instance defined by a specific constraint matrix B and vector $v = (1, 1, 1, 1, 1, 1)$. As shown in Fig. 5.2, the algorithm concentrates probability mass on states with high classical objective values, demonstrating the intended interference effect under postselection. While our current simulations are limited to systems of up to ~ 30 qubits, these results validate the end-to-end behavior of the implementation and highlight the practical advantage of efficient coherent decoding subroutines (in particular GJE) over lookup-based baselines.

Chapter 6

Discussion and Conclusion

This thesis advanced Decoded Quantum Interferometry (DQI) from a primarily algorithm-level proposal to an implementation-oriented, circuit-level study. DQI is conceptually distinct from dominant NISQ optimization approaches such as variational quantum algorithms: instead of repeatedly evaluating a cost function and tuning parameters in a quantum–classical loop, DQI uses an interference (Hadamard/QFT-type) transformation to convert an optimization objective into an induced bounded-distance decoding problem. In this view, the quality of the output distribution is determined by the capability of a decoder that must be embedded coherently and reversibly into the quantum circuit. This decoder-centric interpretation is present in the original DQI formulation, and the results of this thesis reinforce it in concrete implementation terms: implementable DQI is, to a large extent, coherent decoder engineering under strict reversibility and depth constraints.

The primary outcome is a complete and explicit circuit blueprint for DQI applied to Max-XORSAT (instantiated as MaxCut-style instances), decomposed into modular subroutines that can be independently implemented and analyzed. The pipeline is realized as: (i) classical preprocessing to compute the weight vector $\{w_k\}$; (ii) unary amplitude encoding (UAE) that loads $\{w_k\}$ into amplitudes; (iii) deterministic preparation of a coherent low-weight superposition over Dicke layers on the m-qubit error register; (iv) problem encoding, structured as phase encoding followed by constraint/syndrome encoding; (v) coherent bounded-distance decoding with subsequent uncomputation of the error register; (vi) a final $H^{\otimes n}$ interference step; and (vii) measurement with postselection on the successful decoding branch. Beyond describing these stages, the thesis specifies implementable circuit families for each stage and uses that structure to derive explicit, per-block resource formulas and end-to-end resource estimates. In particular, the thesis makes the decoder–uncompute interface a first-class design object: the decoder must be reversible, must act coherently on superpositions of induced syndromes, and must restore the error register to $|0\rangle^{\otimes m}$ to enable the intended interference.

A central conclusion is that the decoder is the dominant algorithmic lever inside DQI. Increasing the effective decoding capability directly improves the amplitude concentration toward high-quality assignments after interference, but it also increases circuit cost. This thesis therefore implemented and compared multiple decoder realizations under a common coherent interface. The decoder portfolio spans three regimes: a reversible lookup-table decoder that is valuable as a correctness baseline for very small instances but grows rapidly with $\sum_{r=0}^{\ell}\binom{m}{r}$; a reversible Gauss–Jordan elimination (GJE) decoder that is transparent and broadly applicable as a reversible linear-algebra primitive; and a BPQM-based decoder that follows a message-passing structure on an unrolled computation tree and introduces additional parameters (d, h) (maximum graph degree and unrolling height). By placing these alternatives side-by-side with the same upstream preparation and encoding stages, the thesis isolates the cost–performance tradeoffs attributable specifically to decoder choice.

The simulation-based resource breakdown strengthens this picture. For the tested sizes, deterministic Dicke state preparation is a substantial front-end cost and can dominate at small scales, which highlights that DQI's practical overhead is not solely "decoding plus interference," but includes a nontrivial state-preparation investment before any decoding advantage can be realized. At the same time, the decoder choice quickly becomes the overall bottleneck as instances grow. The lookup-table decoder exhibits the expected blow-up in gate count and depth, while the GJE decoder remains dramatically smaller in the same regime. The BPQM implementation lies between these extremes in measured resource usage, but it requires additional qubits and introduces multiplexed rotations (UCR_y) as a primitive that can be expensive to compile, making its effective cost strongly dependent on the control size of these multiplexers.

The inclusion of BPQM in this thesis is significant for two complementary reasons. Conceptually, BPQM offers a route to coherent decoding that is structurally aligned with the success of classical belief propagation: it leverages factor-graph locality through repeated check- and variable-node primitives, rather than relying on global elimination or exhaustive enumeration. Technically, BPQM has a growing theoretical foundation, including a message-passing approximation framework that can be made polynomial in relevant parameters when angles/messages are represented in a digitized form. In the present thesis, BPQM is implemented via uniformly-controlled R_y multiplexers (UCR_y), which provides a direct and implementable circuit realization but retains an exponential dependence on the number of control qubits when compiled into a two-qubit gate set. This leads to a clear interpretation: BPQM is a promising decoder family for structured instances, but realizing its scalability in practice likely requires moving from the multiplexer-based construction to a digitized-message implementation that keeps circuit growth polynomial. This gap between a straightforward coherent construction and a truly scalable one is an important outcome of the implementation-first perspective taken in this thesis.

From a systems viewpoint, the modular decomposition provides practical guidance about what is unavoidable and what is flexible in a Max-XORSAT DQI implementation. The front-end (weight loading, low-weight superposition, and reversible syndrome computation) is structurally necessary and establishes a fixed cost that competes directly with the depth budget available for decoding. In contrast, several elements offer genuine design freedom: the specific Dicke-layer preparation method; the reversible decoder embedding; and the tradeoff between exactness and resource reduction in decoding (e.g., limiting ℓ, using approximate message schedules, or restricting computation-tree height h). Because the tables report resources at the subroutine level, they can directly guide where optimization effort is most likely to pay off: reducing Dicke preparation overhead lowers the baseline cost paid by all decoders, while improving decoder circuits yields the most direct gains in DQI's effective amplification behavior.

Several concrete future directions follow from these findings. First, decoder development should be expanded beyond the baseline set studied here. For BPQM in particular, a natural next step is to implement a digitized-message variant that avoids large UCR_y multiplexers and to benchmark its success probability and resource footprint against GJE across controlled instance families and graph structures. Second, hardware-aware evaluation is necessary to assess practical feasibility: compiling the modular circuit families under realistic connectivity, incorporating noise models, and quantifying whether the postselection-and-interference mechanism remains robust at achievable depths. Third, the state-preparation front-end deserves targeted optimization. Since UAE and deterministic Dicke preparation are paid before decoding begins, improvements here directly benefit all decoder choices and can shift the feasible regime of (m, n, ℓ). Finally, scaling studies should move from isolated demonstrations toward systematic laws: how success probability, postselection rate, and required resources scale with (m, n, ℓ) and with instance structure, and how DQI compares to classical baselines and other quantum optimization methods under comparable resource constraints.

In summary, this thesis provides a concrete foundation for studying DQI as an implementable quantum algorithm. It delivers an end-to-end circuit realization for Max-XORSAT, specifies modular implementations of all major blocks, derives explicit resource estimates that expose the dominant tradeoffs, and validates the pipeline in simulation. Most importantly, it makes decoder choice — the central determinant of practical performance and scalability. This closes a key gap between DQI's high-level promise and the requirements of circuit-level execution, and it outlines a clear roadmap for future work: improved coherent decoders (especially scalable BPQM variants), hardware-aware compilation and noise studies, optimized state preparation, and broader benchmarking on structured instance families.

Acknowledgment

This work was supported by MEXT-Quantum Leap Flagship Program and JST Moonshot RD.

January 26, 2026

Bibliography

[1] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. 2014. doi:10.48550/arXiv.1411.4028. arXiv preprint.

[2] M. B. Hastings. A Short Path Quantum Algorithm for Exact Optimization. *Quantum*, 2:78, July 2018. ISSN 2521-327X. doi:10.22331/q-2018-07-26-78.

[3] Stephen P. Jordan, Noah Shutty, Mary Wootters, Adam Zalcman, Alexander Schmidhuber, Robbie King, Sergei V. Isakov, Tanuj Khattar, and Ryan Babbush. Optimization by decoded quantum interferometry. *Nature*, 646(8086):831–836, October 2025. ISSN 1476-4687. doi:10.1038/s41586-025-09527-5.

[4] André Chailloux and Jean-Pierre Tillich. Quantum advantage from soft decoders. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, STOC '25, page 738–749, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715105. doi:10.1145/3717823.3718319.

[5] Bahaa Saleh. *The Laser*, pages 71–85. Springer International Publishing, Cham, 2016. ISBN 978-3-319-31903-2. doi:10.1007/978-3-319-31903-2_4.

[6] Neil Ashby. Relativity in the global positioning system. *Living Reviews in Relativity*, 6(1):1, Jan 2003. ISSN 1433-8351. doi:10.12942/lrr-2003-1.

[7] Simon M. Sze and Kwok K. Ng. *Physics of Semiconductor Devices*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 3 edition, 2006. ISBN 9780470068328. doi:10.1002/9780470068328.ch0.

[8] Jonathan P. Dowling and Gerard J. Milburn. Quantum technology: The second quantum revolution. 2002. doi:10.48550/arXiv.quant-ph/0206091. arXiv preprint.

[9] C. L. Degen, F. Reinhard, and P. Cappellaro. Quantum sensing. *Rev. Mod. Phys.*, 89:035002, Jul 2017. doi:10.1103/RevModPhys.89.035002.

[10] Rodney Van Meter. *Quantum Networking*. John Wiley Sons, Ltd, 2014. ISBN 9781118648919. doi:10.1002/9781118648919.ch1.

[11] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, 2006. doi:10.1109/N-SSC.2006.4785860.

[12] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974. doi:10.1109/JSSC.1974.1050511.

[13] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. doi:10.1137/S0097539795293172.

[14] Lov K. Grover. A fast quantum mechanical algorithm for database search. 1996. doi:10.48550/arXiv.quant-ph/9605043. arXiv preprint.

[15] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[16] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, Jan 2009. ISSN 1745-2481. doi:10.1038/nphys1157.

[17] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Rev. Mod. Phys.*, 90:015002, Jan 2018. doi:10.1103/RevModPhys.90.015002.

[18] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950. doi:10.1002/j.1538-7305.1950.tb00463.x.

[19] R. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28, 1962. doi:10.1109/TIT.1962.1057683.

[20] Neal Glover and Trent Dudley. *Practical Error Correction Design for Engineers*. Cirrus Logic, Broomfield, CO, 2 edition, 1991. ISBN 0-927239-00-0. Rev. 1.1.

[21] T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, 2001. doi:10.1109/18.910578.

[22] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981. doi:10.1109/TIT.1981.1056404.

[23] Judea Pearl. Reverend bayes on inference engines: a distributed hierarchical approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI'82, page 133–136. AAAI Press, 1982.

[24] Ye-Hua Liu and David Poulin. Neural Belief-Propagation Decoders for Quantum Error-Correcting Codes. *Phys. Rev. Lett.*, 122(20):200501, 2019. doi:10.1103/PhysRevLett.122.200501.

[25] Fotios Petropoulos, Gilbert Laporte, Emel Aktas, Sibel A. Alumur, Claudia Archetti, Hayriye Ayhan, Maria Battarra, Julia A. Bennell, Jean-Marie Bourjolly, John E. Boylan, Michèle Breton, David Canca, Laurent Charlin, Bo Chen, Cihan Tugrul Cicek, Louis Anthony Cox, Christine S.M. Currie, Erik Demeulemeester, Li Ding, Stephen M. Disney, Matthias Ehrgott, Martin J. Eppler, Güneş Erdoğan, Bernard Fortz, L. Alberto Franco, Jens Frische, Salvatore Greco, Amanda J. Gregory, Raimo P. Hämäläinen, Willy Herroelen, Mike Hewitt, Jan Holmström, John N. Hooker, Tuğçe Işık, Jill Johnes, Bahar Y. Kara, Özlem Karsu, Katherine Kent, Charlotte Köhler, Martin Kunc, Yong-Hong Kuo, Adam N. Letchford, Janny Leung, Dong Li, Haitao Li, Judit Lienert, Ivana Ljubić, Andrea Lodi, Sebastián Lozano, Virginie Lurkin, Silvano Martello, Ian G. McHale, Gerald Midgley, John D.W. Morecroft, Akshay Mutha, Ceyda Oğuz, Sanja Petrovic, Ulrich Pferschy, Harilaos N. Psaraftis, Sam Rose, Lauri Saarinen, Said Salhi, Jing-Sheng Song, Dimitrios Sotiros, Kathryn E. Stecke, Arne K. Strauss, İstenç Tarhan, Clemens Thielen, Paolo Toth, Tom Van Woensel, Greet Vanden Berghe, Christos Vasilakis, Vikrant Vaze, Daniele Vigo, Kai Virtanen, Xun Wang, Rafał Weron, Leroy White, Mike Yearworth, E. Alper Yıldırım, Georges Zaccour, and Xuying Zhao. Operational research: methods and applications. *Journal of the Operational Research Society*, 75 (3):423–617, December 2023. ISSN 1476-9360. doi:10.1080/01605682.2023.2253852.

[26] Jorge Nocedal and Stephen J. Wright. Numerical optimization. In *Fundamental Statistical Inference*, 2018. URL `https://api.semanticscholar.org/CorpusID:189864167`.

[27] Amira Abbas, Andris Ambainis, Brandon Augustino, Andreas Bärtschi, Harry Buhrman, Carleton Coffrin, Giorgio Cortiana, Vedran Dunjko, Daniel J. Egger, Bruce G. Elmegreen, Nicola Franco, Filippo Fratini, Bryce Fuller, Julien Gacon, Constantin Gonciulea, Sander Gribling, Swati Gupta, Stuart Hadfield, Raoul Heese, Gerhard Kircher, Thomas Kleinert, Thorsten Koch, Georgios Korpas, Steve Lenk, Jakub Marecek, Vanio Markov, Guglielmo Mazzola, Stefano Mensa, Naeimeh Mohseni, Giacomo Nannicini, Corey O'Meara, Elena Peña Tapia, Sebastian Pokutta, Manuel Proissl, Patrick Rebentrost, Emre Sahin, Benjamin C. B. Symons, Sabine Tornow, Víctor Valls, Stefan Woerner, Mira L. Wolf-Bauwens, Jon Yard, Sheir Yarkoni, Dirk Zechiel, Sergiy Zhuk, and Christa Zoufal. Challenges and opportunities in quantum optimization. *Nature Reviews Physics*, 6(12):718–735, October 2024. ISSN 2522-5820. doi:10.1038/s42254-024-00770-9.

[28] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2, 2014. ISSN 2296-424X. doi:10.3389/fphy.2014.00005.

[29] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models. 2019. doi:10.48550/arXiv.1811.11538. arXiv preprint.

[30] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), July 2014. ISSN 2041-1723. doi:10.1038/ncomms5213.

[31] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. ISSN 2521-327X. doi:10.22331/q-2018-08-06-79.

[32] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *Nature Physics*, 16(10):1050–1057, June 2020. ISSN 1745-2481. doi:10.1038/s41567-020-0932-7.

[33] Kristan Temme, Sergey Bravyi, and Jay M. Gambetta. Error mitigation for short-depth quantum circuits. *Physical Review Letters*, 119(18), November 2017. ISSN 1079-7114. doi:10.1103/physrevlett.119.180509.

[34] Zhenyu Cai, Ryan Babbush, Simon C. Benjamin, Suguru Endo, William J. Huggins, Ying Li, Jarrod R. McClean, and Thomas E. O'Brien. Quantum error mitigation. *Reviews of Modern Physics*, 95(4), December 2023. ISSN 1539-0756. doi:10.1103/revmodphys.95.045005.

[35] Ryuji Takagi, Suguru Endo, Shintaro Minagawa, and Mile Gu. Fundamental limits of quantum error mitigation. *npj Quantum Information*, 8(1), September 2022. ISSN 2056-6387. doi:10.1038/s41534-022-00618-z.

[36] Natchapol Patamawisut, Naphan Benchasattabuse, Michal Hajdušek, and Rodney Van Meter. Quantum circuit design for decoded quantum interferometry. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 01, pages 291–301, 2025. doi:10.1109/QCE65121.2025.00041.

[37] Eric R. Anschuetz, David Gamarnik, and Jonathan Z. Lu. Decoded quantum interferometry requires structure. 2025. doi:10.48550/arXiv.2509.14509. arXiv preprint.

[38] Hsin-Yuan Huang, Soonwon Choi, Jarrod R. McClean, and John Preskill. The vast world of quantum advantage. 2025. doi:10.48550/arXiv.2508.05720. arXiv preprint.

[39] Takashi Yamakawa and Mark Zhandry. Verifiable quantum advantage without structure. *Journal of the ACM*, 71(3):1–50, June 2024. ISSN 1557-735X. doi:10.1145/3658665.

[40] Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. Transformation of quantum states using uniformly controlled rotations. July 2004. doi:10.48550/arXiv.quant-ph/0407010. arXiv preprint.

[41] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. Quantum circuits for isometries. *Phys. Rev. A*, 93:032318, Mar 2016. doi:10.1103/PhysRevA.93.032318.

[42] Guang Hao Low, Vadym Kliuchnikov, and Luke Schaeffer. Trading t gates for dirty qubits in state preparation and unitary synthesis. *Quantum*, 8:1375, June 2024. ISSN 2521-327X. doi:10.22331/q-2024-06-17-1375.

[43] Andreas Bärtschi and Stephan Eidenbenz. *Deterministic Preparation of Dicke States*, page 126–139. Springer International Publishing, 2019. ISBN 9783030250270. doi:10.1007/978-3-030-25027-0_9.

[44] Peter Høyer and Robert Špalek. Quantum fan-out is powerful. *Theory of Computing*, 1(5):81–103, 2005. doi:10.4086/toc.2005.v001a005.

[45] R. H. Dicke. Coherence in spontaneous radiation processes. *Phys. Rev.*, 93:99–110, Jan 1954. doi:10.1103/PhysRev.93.99.

[46] R. Prevedel, G. Cronenberg, M. S. Tame, M. Paternostro, P. Walther, M. S. Kim, and A. Zeilinger. Experimental realization of dicke states of up to six qubits for multiparty quantum networking. *Physical Review Letters*, 103(2), July 2009. ISSN 1079-7114. doi:10.1103/physrevlett.103.020503.

[47] Yun-Feng Xiao, Xu-Bo Zou, and Guang-Can Guo. Generation of atomic entangled states with selective resonant interaction in cavity quantum electrodynamics. *Phys. Rev. A*, 75:012310, Jan 2007. doi:10.1103/PhysRevA.75.012310.

[48] Pradeep Niroula, Ruslan Shaydulin, Romina Yalovetzky, Pierre Minssen, Dylan Herman, Shaohan Hu, and Marco Pistoia. Constrained quantum optimization for extractive summarization on a trapped-ion quantum computer. *Scientific Reports*, 12 (1):17171, Oct 2022. ISSN 2045-2322. doi:10.1038/s41598-022-20853-w.

[49] Classiq. Decoded quantum interferometry algorithm for max-xorsat (documentation), 2025. Available at: `https://docs.classiq.io/latest/explore/algorithms/dqi/dqi_max_xorsat/`.

[50] Steven C. Althoen and Renate Mclaughlin. Gauss-jordan reduction: A brief history. *The American Mathematical Monthly*, 94(2):130–142, 1987. doi:10.1080/00029890.1987.12000605.

[51] C.D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Other titles in applied mathematics. Society for Industrial and Applied Mathematics, 2023. ISBN 9781611977431. URL `https://books.google.co.jp/books?id=z7GczwEACAAJ`.

[52] Simone Perriello, Alessandro Barenghi, and Gerardo Pelosi. A complete quantum circuit to solve the information set decoding problem. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 366–377, 2021. doi:10.1109/QCE52317.2021.00056.

[53] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001. doi:10.1109/18.910572.

[54] H.-A. Loeliger. An introduction to factor graphs. *IEEE Signal Processing Magazine*, 21(1):28–41, 2004. doi:10.1109/MSP.2004.1267047.

[55] David Mackay and Radford Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 08 2002.

[56] Kevin Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. 2013. URL `10.48550/arXiv.1301.6725`. arXiv preprint.

[57] Joseph M Renes. Belief propagation decoding of quantum channels by passing quantum messages. *New Journal of Physics*, 19(7):072001, July 2017. ISSN 1367-2630. doi:10.1088/1367-2630/aa7c78.

[58] Christophe Piveteau and Joseph M. Renes. Quantum message-passing algorithm for optimal and efficient decoding. *Quantum*, 6:784, August 2022. ISSN 2521-327X. doi:10.22331/q-2022-08-23-784.

[59] D. Coppersmith. An approximate fourier transform useful in quantum factoring. 2002. doi:10.48550/arXiv.quant-ph/0201067. arXiv preprint.

[60] A. Yu. Kitaev. Quantum measurements and the abelian stabilizer problem. 1995. doi:10.48550/arXiv.quant-ph/9511026. arXiv preprint.

[61] Cristopher Moore, Daniel Rockmore, Alexander Russell, and Leonard J. Schulman. The power of strong fourier sampling: Quantum algorithms for affine groups and hidden shifts. 2005. doi:10.48550/arXiv.quant-ph/0503095. arXiv preprint.