



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

**HYBRID QUANTUM COMPUTING:
PRACTICAL APPLICATIONS AND IMPLEMENTATION**

HYBRIDNÍ KVANTOVÉ POČÍTÁNÍ: PRAKTICKÉ VYUŽITÍ A IMPLEMENTACE

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. JAN SWIATKOWSKI

SUPERVISOR

VEDOUCÍ PRÁCE

prof. Ing. JIŘÍ JAROŠ, Ph.D.

BRNO 2026

Master's Thesis Assignment



172425

Institut: Department of Computer Systems (DCSY)
Student: **Swiatkowski Jan, Bc.**
Programme: Information Technology and Artificial Intelligence
Specialization: High Performance Computing
Title: **Hybrid Quantum Computing: Practical Applications and Implementation**
Category: Modelling and Simulation
Academic year: 2025/26

Assignment:

1. Study the concept of hybrid quantum computing and its place in the current development of information technologies.
2. Explore existing platforms and libraries for hybrid quantum programming (e.g., Qiskit, PennyLane, Cirq, D-Wave Ocean).
3. Choose a suitable existing practical task (e.g., scheduling optimization, simple machine learning models, pathfinding in graphs).
4. Prepare a solution to this task using a hybrid approach, leveraging both the classical and the quantum parts.
5. Implement the solution in the chosen framework and verify its functionality on a simulator and, if possible, also on an available quantum computer.
6. Analyze the results of the implemented tasks in terms of speed/performance, implementation complexity, and practical applicability.
7. Discuss the current state of quantum computing and its readiness for practical use.

Literature:

- According to the supervisor's advice.

Requirements for the semestral defence:

- Items 1 to 4 of the assignment.

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Jaroš Jiří, prof. Ing., Ph.D.**
Head of Department: Sekanina Lukáš, prof. Ing., Ph.D.
Beginning of work: 1.11.2025
Submission deadline: 20.5.2026
Approval date: 31.10.2025

Abstract

This thesis focuses on the methodology for mapping and executing practically applicable optimization problems, such as Exact Set Cover, Knapsack, and SAT, on quantum computers. Unlike classical computers that operate with bits, quantum computers utilize quantum qubits and phenomena such as superposition and entanglement, allowing them to search vast state spaces in parallel. They are therefore particularly suitable for solving complex combinatorial problems where classical algorithms encounter computational limits. The core of the work is a hybrid solver for the Exact Set Cover problem employing a Branch-and-Bound strategy. Within this framework, quantum circuits based on Grover's algorithm are selectively generated for relevant subproblems. Grover's algorithm provides a quadratic speedup for unstructured search, which is crucial for finding configurations. Accounting for the constraints of the current NISQ hardware generation, the thesis includes a comparative analysis of results from simulators and real systems—IQM Star24 and IBM Marrakesh (Heron R2)—including a comparison of the Qrisp and Qiskit frameworks. The results of these optimization problem implementations were evaluated using Z-Score, and the requirements of the given quantum circuits were compared for the experimental problem instances. Furthermore, experiments were conducted to assess the threshold depth of a quantum circuit at which the expected output becomes indistinguishable. Matrix-free Measurement Mitigation (M3) was utilized for noise suppression. The thesis concludes with a practical methodological guide for the development of hybrid quantum-classical applications. The work also discusses the advantages of integrating quantum computers with supercomputers at a single physical location and the coordination of their resource allocation.

Abstrakt

Tato práce se zabývá metodikou mapování a spuštění prakticky uplatnitelných optimalizačních úloh jako Exact Set Cover, Knapsack, SAT na kvantových počítačích. Na rozdíl od klasických počítačů, které pracují s bity, využívají kvantové počítače kvantové qubity a jevy jako superpozice a provázanost, což jim umožňuje paralelně prohledávat rozsáhlé stavové prostory. Jsou proto vhodné zejména pro řešení složitých kombinatorických problémů, kde klasické algoritmy narážejí na výpočetní limity. Jádrem práce je hybridní solver pro problém Exact Set Cover využívající strategii Branch-and-Bound. V jejím rámci jsou selektivně generovány kvantové obvody založené na Groverově algoritmu pro relevantní podproblémy. Groverův algoritmus poskytuje kvadratické zrychlení pro nestrukturované prohledávání, což je důležité pro nalezení konfigurací. S ohledem na omezení současné generace hardwaru NISQ práce obsahuje komparativní analýzu výsledků ze simulačních a reálných systémů IQM Star24 a IBM Marrakesh (Heron R2), včetně porovnání frameworků Qrisp a Qiskit. Výsledky těchto implementací optimalizačních úloh byly zhodnoceny pomocí Z-Score a byly porovnány požadavky daných kvantových obvodů pro experimentální instance problémů daných úloh. Dále byly provedeny experimenty zhodnocující prahovou hloubku kvantového obvodu, kdy se stane očekávaný výsledek obvodu nerozlišitelným. K potlačení šumu byla využita technika Matrix-free Measurement Mitigation (M3). Práce je zakončena praktickou metodickou příručkou pro vývoj hybridních kvantově-klasických aplikací. Také diskutuje výhody integrace kvantového počítače se superpočítačem v jedné fyzické lokaci a koordinaci alokace jejich zdrojů.

Keywords

hybrid computing, quantum computing, practical application, comparison of approaches, Branch-and-Bound, Exact Set Color, SAT, Knapsack, Qiskit, Qrisp, VLQ, IQM, IBM, Error Correction, Error Mitigation, NISQ

Klíčová slova

hybridní výpočty, kvantové výpočty, praktické využití, porovnání přístupů, Groverův algoritmus, Branch-and-Bound, Exact Set Color, SAT, Knapsack, Qiskit, Qrisp, VLQ, IQM, IBM, Error Correction, Error Mitigation, NISQ

Reference

SWIATKOWSKI, Jan. *Hybrid Quantum Computing: Practical Applications and Implementation*. Brno, 2026. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Jiří Jaroš, Ph.D.

Rozšířený abstrakt

Tato diplomová práce dokumentuje komplexní vývoj a implementaci hybridních kvantově-klasických aplikací zaměřených na řešení NP-úplných kombinatorických optimalizačních problémů. Hlavní přínos spočívá v překlenutí mezery mezi teoretickým návrhem algoritmů a praktickým provedením na současném hardwaru typu NISQ (Noisy Intermediate-Scale Quantum), se specifickým zaměřením na problémy přesného pokrytí množiny (ECP), batohu (Knapsack) a splnitelnosti booleovských formulí (SAT).

Ústřední případová studie implementuje hybridní solver pro problém plánování směn na fyzioterapeutické klinice, mapovaný na formulaci ECP. V tomto scénáři musí být pacienti přiřazeni fyzioterapeutům se specifickou kvalifikací v rámci více časových slotů tak, aby byl každý pacient obsloužen právě jednou. Architektura využívá klasickou strategii větvení a mezí (Branch-and-Bound) k prořezávání vyhledávacího prostoru a rozkladu velkých instancí problémů na zvládnutelné podproblémy. Ty jsou následně odesílány kvantovému solveru založenému na Groverově vyhledávání. Kvantové orákulum bylo sestaveno ze tří fází: inkrementace čítačů, kontrolní podmínky a reverze akce (uncompute). Pro čítač byla použita implementace aritmetiky s přenosem (ripple-carry) za využití hradel MCX podle článku Jiang a Wang (2023), který sloužil jako zdroj pro implementaci kvantové části ECP (Exact Set Cover) řešiče za použití Groverova algoritmu. V článku Jiang a Wang (2023) byla tato kvantová implementace ověřena na simulátoru, nikoliv však na reálném hardwaru. Implementace byla porovnána ve frameworkcích Qiskit a Qrisp. Zatímco Qiskit poskytuje nízkouúrovňovou kontrolu nad kvantovými branami, Qrisp výrazně snižuje složitost implementace díky dekorátoru `@auto_uncompute` a nativní podpoře aritmetiky. ECP orákulum lze v Qrispu vyjádřit přibližně v 15 řádcích kódu, oproti více než stovce řádků potřebných při ruční správě přenosů a ancilla qubitů při syntéze na úrovni kvantových bran. Experimentální vyhodnocení a výsledky na hardwaru: Implementace byla testována na lokálních simulátorech, modelech šumu a fyzických kvantových procesorech: IQM Star24 v IT4Innovations a IBM Marrakesh Heron R2. Pro potlačení chyb byla experimentálně použita M3 korekce chyb jako dostupná reprezentativní technika oproti složitým alternativám jako např. planární kódy. Pro zhodnocení statistické rozlišitelnosti správných výsledků byla použita metrika Z-skóre. Následné spuštění implementace v Qrisp na hardwaru odhalilo klíčové výkonnostní charakteristiky. Jako limit se projevila hloubka transpilovaného obvodu přibližně při hodnotě 250; za tímto prahem způsobil šum hardwaru kolaps jistoty stavu, čímž se řešení stala nerozlišitelná. SAT solver prokázal vyšší odolnost na hardwaru a dosáhl 99% spolehlivosti při hloubce obvodu 233 díky přímému mapování logiky na kvantové brány. ECP Groverův solver dosáhl pouze 95% spolehlivosti i u nejjednodušších případů při hloubce 340, než chyby přesáhly prahovou úroveň rozlišitelnosti. Zatímco idealistické simulátory identifikovaly řešení s 99% spolehlivostí, fyzický hardware Heron R2 byl zásadně omezen hloubkami obvodu. V důsledku lze konstatovat, že fidelita bran, nikoli počet qubitů, zůstává primárním úzkým hrdlem pro praktické nasazení. Integrace HPC-QC a metodologie: Práce dále analyzuje výhody integrace HPC a kvantových výpočtů a zabývá se synchronizací plánování úloh (např. SLURM) pro kooperativní využití zdrojů. Práce je zakončena praktickým metodologickým průvodcem shrnujícím celý životní cyklus vývoje hybridních aplikací — od mapování problémů a konstrukce orákula až po zpracování šumu — a slouží jako základ pro práci s kvantovými gate-based algoritmy pro aplikaci na konkrétní problémy.

Hybrid Quantum Computing: Practical Applications and Implementation

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of prof. Ing. Jiří Jaroš, Ph.D. and the additional supervision of Ing. Jan Martinovič, Ph.D. from the ADAS Laboratory at IT4Innovations.

.....
Jan Swiatkowski
May 15, 2026

Acknowledgements

I would like to express my sincere gratitude to prof. Ing. Jiří Jaroš, Ph.D., for his invaluable guidance on the structure and writing of this work, and to Ing. Jan Martinovič, Ph.D., for his high-level supervision and insightful technical perspective. I am also deeply grateful to Mgr. Branislav Jansík, Ph.D., Ing. Petr Ptáček and Bc. Andrei Nechiporuk for providing their technical expertise and practical advice within the field of quantum computing. I would also like to extend my sincere thanks to my family, especially my parents and my wife Karolina, for their heartfelt support during the challenging times of working on this thesis.

Contents

1	Introduction	9
1.1	The Qubit	9
1.2	Thesis Objectives and Scope	10
1.3	Motivation	11
2	Overview of Current Quantum Hardware	12
2.1	Available Hardware and Purpose Comparison	13
2.2	Comparison of Selected Quantum Processing Unit Architectures	17
3	Exploring Quantum Computing Platforms and Languages	24
3.1	Q#	25
3.2	OpenQASM	26
3.3	Qiskit	27
3.4	D-Wave Ocean SDK	28
3.5	PennyLane	29
3.6	Qrisp Framework	30
3.7	Comparison of Quantum Programming Environments	33
3.8	Comparison of Simulation Paradigms of Qrisp and AerSimulator	35
4	Identifying Use Cases for Hybrid Quantum Computing	41
4.1	Exact Cover Problem Definition	42
4.2	Selected Use Case Problem Description	42
4.3	Example Instance of Selected ECP	46
5	Hybrid Quantum Computing ECP Implementation	47
5.1	Implementation Architecture Overview	47
5.2	Problem Mapping: From Scheduling to Exact Cover	47
5.3	Quantum Exact Cover Solver	49
5.4	Hybrid Branch-and-Bound Algorithm	52
5.5	Implementation Details	56
5.6	Complexity Analysis	57
5.7	Alignment with Initial Paper Design of ECP Solver	58
6	Error Mitigation and Correction	59
6.1	Quantum Error Correction and the Rotated Planar Code	59
6.2	Measurement Mitigation: The M3 Implementation	61
6.3	Implementation: Applying M3 to Grover's ECP	62

6.4	Empirical Depth Benchmarking: IT4I VLQ Star24 vs. IBM Marrakesh . . .	64
7	Qrisp Framework: ECP - Knapsack - SAT Implementation	68
7.1	ECP QAOA Implementation	68
7.2	ECP: Grover's Algorithm Using Quantum Gates Operation	69
7.3	ECP: Grover's Algorithm Fully Utilizing Qrisp Abstraction	70
7.4	Benchmark Instances and Expected Results for Exact Cover Problem . . .	70
7.5	Knapsack: Grover's Algorithm Fully Utilizing Qrisp Abstraction	72
7.6	SAT Grover's Algorithm Fully Utilizing Qrisp	74
7.7	Qrisp Framework: Integration with VLQ and IBM HW Backends	75
7.8	Qrisp: Summary and Outcomes of Results	76
8	Outcomes and Summary	85
8.1	Discussion about HPC-QC Integration	85
8.2	Results of Evaluation on real HW and Simulators	87
8.3	Key Findings and Lessons Learned	88
	Bibliography	92
.1	Preparation of Environemnt for Experiments and Usage Guide	101

List of Figures

2.1	Quantum chip of the IQM Star24 (VLQ) processor	12
2.2	Cryogenic refrigeration system for the IQM Star24 (VLQ) quantum processor	13
2.3	The Lucy quantum computer, a photonic system developed by Quandela and hosted at the TGCC (CEA), France	15
2.4	The PIAST-Q quantum computer, a trapped-ion system developed by AQT and hosted at the Poznań Supercomputing and Networking Center (PSNC), Poland	16
2.5	IQM Crystal 20-qubit chip design	23
2.6	IQM Radiance Star 24-qubit chip design	23
2.7	Comparison of IQM superconducting quantum processor chip designs: Crystal 20 and Radiance Star 24	23
3.1	A $Q\#$ operation that prepares a qubit in superposition and measures it. . .	26
3.2	A single-qubit superposition circuit written in OpenQASM 3.0.	27
3.3	Python code using the Qiskit SDK to construct a quantum circuit with one qubit and measure it.	28
3.4	Python code using D-Wave Ocean SDK to define a simple QUBO problem and sample solutions.	29
3.5	Python code using PennyLane to define a simple quantum circuit and compute its expectation value.	30
3.6	Qrisp code demonstrating high-level quantum arithmetic and variable measurement using a QuantumFloat.	32
5.1	Hybrid branch-and-bound algorithm with Grover-based quantum search for the exact cover problem	48
5.2	The Grover Diffusion Operator implemented in Qiskit.	52
6.1	Planary Code to Rotated Planary Code Transformation	60
6.2	Rotated Planary Code Example	61
6.3	Comparison of a three-qubit circular entanglement circuit - Logical	65
6.4	Comparison of a three-qubit circular entanglement circuit - IBM Marrakesh	65
6.5	Comparison of a three-qubit circular entanglement circuit - IQM Star24 (VLQ)	65
6.6	Comparison of a three-qubit circular entanglement circuit across different backends.	65
6.7	Quantum State Survival: Certainness vs. Circuit Depth across simulators and real hardware.	66
8.1	Memory Requirements for Sampled Quantum Results	86
8.2	Impact of number of shots on hardware execution latency	88

8.3	Impact of logical circuit depth on hardware execution latency	89
8.4	IBM Marrakesh: Execution scaling analysis.	90
8.5	IQM Star24 (VLQ): Execution scaling analysis.	90
8.6	Execution latency heatmaps for IBM Marrakesh and IQM Star24 (VLQ) . .	90
8.7	IQM Quantum Computers technology roadmap (2023–2030+) showing NISQ optimization, QEC demonstrators, and fault-tolerant scaling phases.	91

List of Tables

2.1	Comparative technical specifications for leading quantum computing platforms for year 2025.	18
2.2	Accessibility and pricing models for commercial and academic/research quantum computing platforms	19
2.3	Comparative specifications of QPU architectures for European HPC integration. Values from primary manufacturer sources; empty cells indicate data not publicly available.	21
3.1	Comparison of quantum computing platforms and languages.	33
3.2	Feature comparison of AerSimulator and Qrisp Simulator.	40
4.1	Maximum ECP problem sizes for available quantum hardware.	44
6.1	Original results from IBM Marrakesh. The target solution (B, C) is ranked 12th. The corresponding circuit features a depth of 449 and was evaluated using 8,000 shots and one iteration.	63
6.2	Results from IBM Marrakesh after M3 Mitigation. The target solution (B, C) has ascended to 3rd place. The corresponding circuit features a depth of 449 and was evaluated using 8,000 shots and one iteration.	63
7.1	Results for the Exact Cover Problem (ECP) Experiment 0, utilizing the QAOA solver on the <i>IBM FakeMarrakesh</i> backend	78
7.2	Results for the Exact Cover Problem (ECP) Experiment 1, utilizing the QAOA solver on the <i>IBM FakeMarrakesh</i> backend	79
7.3	Results for the Exact Cover Problem (ECP) Paper Example, utilizing the QAOA solver on the <i>IBM FakeMarrakesh</i> backend	80
7.4	Performance evaluation of quantum algorithms on various problem instances and backends	81
7.5	Performance evaluation of quantum algorithms on various problem instances and backends	82
1	QRISP framework — evaluation of ECP solver performance, implementation approaches, and quantified result certainty. Experiments and implementation details are in Chapter 7; z-score methodology is in Section 7.8.	97
2	QRISP framework — evaluation of SAT and Knapsack solver performance, implementation approaches, and quantified result certainty.	99

List of abbreviations

BSC Barcelona Supercomputing Center. 19

CEA The French Alternative Energies and Atomic Energy Commission. 19

CLOPS Circuit Layer Operations Per Second: A performance metric that measures the number of quantum circuit layers executed per second on a quantum processor, used to benchmark the speed and throughput of quantum hardware independent of qubit count. 20–22

CNF Conjunction Normal Form. 74, 83, 84

CX gate Controlled-X Gate is a two-qubit unitary gate represented by the matrix with diagonal elements $[0, 0, 0, 1]$ and off-diagonal elements that perform a bit flip, which applies an X gate to the target qubit conditioned on the control qubit being in the $|1\rangle$ state, essential for implementing multi-qubit quantum algorithms and creating entanglement. 64, 88, 89

CZ gate Controlled-Z gate is two-qubit unitary gate represented by the matrix with diagonal elements $[1, 1, 1, -1]$, which applies a Z gate, phase flip, to the target qubit conditioned on the control qubit being in the $|1\rangle$ state, essential for implementing multi-qubit quantum algorithms. 20–22, 66, 67

ECP The Exact Set Cover Problem is a classical computational NP-complete problem where you must find a collection of subsets that together cover all elements in a universal set, with each element appearing in exactly one subset. 1, 2, 5, 11, 42, 44, 46, 47, 49, 50, 55, 57, 58, 60–62, 64, 66–71, 76, 77, 81, 82, 87–89

FPGA Field-Programmable Gate Array: An integrated circuit designed to be configured by a customer or a designer after manufacturing, allowing for hardware-level optimization of specific algorithms. 39

GHZ A quantum circuit that creates a Greenberger–Horne–Zeilingner (GHZ) state, an entangled multi-qubit state where all qubits are equally superposed and correlated, commonly used to test quantum entanglement and demonstrate quantum advantage over classical systems. 20, 62

GPU Graphics Processing Unit. 38–40

HPC High-Performance Computing leverages massive parallel processing for complex simulations, predictive modeling, and scientific research. Systems like Karolina integrate

thousands of CPU cores and hundreds of GPU accelerators, utilizing high-speed interconnects for low-latency data transfer across nodes and providing petascale storage for massive datasets. 2, 5, 17, 19, 21, 22, 78, 85, 87

LRZ Leibniz Supercomputing Centre. 18, 19

MCX gate Multi-Controlled-X gate is a multi-qubit unitary gate that applies an X gate, bit flip, to the target qubit only when all control qubits are in the $|1\rangle$ state. 50, 74

NISQ NISQ stands for Noisy Intermediate-Scale Quantum, a term that describes the current era of quantum computing development. It refers to quantum computers that have somewhere up to few hundred qubits (quantum bits), but are still prone to errors and imperfections. 4, 11, 17, 34–36, 55, 59, 66, 78, 84, 85, 87, 91

PCSS Poznan Supercomputing and Networking Center. 16, 19

Q-score An application-based benchmark that measures the maximum number of variables a quantum system can effectively process to solve a Max-Cut optimization problem better than a random classical heuristic; it provides a holistic measure of the entire quantum stack, from hardware to compiler efficiency.. 20

QAOA Quantum Approximate Optimization Algorithm (QAOA): A hybrid variational algorithm that uses a parameterized ansatz of alternating problem and mixer Hamiltonian layers. Classical optimization routines iteratively tune circuit parameters, with each iteration evaluated on a quantum processor, until the parameter configuration that maximizes the objective function and thus produces the best approximate solution to a combinatorial optimization problem is reached. Well-suited for implementation on current noisy quantum hardware. 2, 5, 24, 33, 68–71, 77–82, 85

QC Quantum Computing utilizes quantum mechanical phenomena, such as superposition and entanglement, to perform computations fundamentally different from classical binary logic. By employing qubits instead of bits, QC can explore vast solution spaces simultaneously, offering exponential speedups for specific tasks like cryptography, material science, and complex optimization. 2, 17, 22, 85

QEC Quantum Error Correction: A set of techniques that detect and correct errors in quantum computations caused by decoherence and noise, essential for building fault-tolerant quantum computers by encoding logical qubits across multiple physical qubits. 4, 59, 91

QPU Quantum Processing Unit. 5, 17, 18, 20, 21, 23, 66, 67, 85

QUBO Quadratic Unconstrained Binary Optimization is an optimization problem of the form $E(x) = \sum_{i,j} Q_{ij}x_i x_j$ where $x \in \{0,1\}^n$ and Q is a real-valued matrix, commonly used to translate combinatorial optimization problems into formats compatible with quantum annealers and quantum algorithms. 3, 15, 25, 28, 29, 68, 69

SAT The Boolean Satisfiability Problem is a fundamental computational problem in logic and computer science that determines if there exists an assignment of truth values to variables that makes a given Boolean formula true. 2, 11, 41, 68, 74, 75, 77, 81–83, 87

SC Superconducting. 19

SDK Software Development Kit. 3, 25, 29

SWAP gate A two-qubit unitary gate represented by a permutation matrix that interchanges the quantum states of two qubits by mapping the basis states $|01\rangle \leftrightarrow |10\rangle$ while leaving $|00\rangle$ and $|11\rangle$ invariant, essential for qubit routing in hardware architectures with restricted connectivity and typically implemented via three CX gates. 14

VQA Variational Quantum Algorithm: A quantum algorithm that combines a parameterized quantum circuit (ansatz) with classical optimization to solve problems by iteratively refining circuit parameters based on measured expectation values, reducing coherence time requirements compared to fault-tolerant quantum algorithms . 15, 29

VQE A variational quantum algorithm that employs an ansatz quantum circuit with adjustable parameters to find the minimum eigenvalue of a Hamiltonian by repeatedly measuring expectation values and optimizing parameters classically, particularly suited for simulating molecular systems and quantum chemistry problems. 22

Chapter 1

Introduction

This thesis documents the development and implementation of hybrid quantum-classical applications designed to solve complex combinatorial optimization problems. The work encompasses the complete development lifecycle, beginning with the selection of suitable quantum algorithms and their mapping to practical problem domains, continuing through execution on contemporary available quantum hardware for science and research, and concluding with a rigorous analysis and synthesis of the results.

Nowadays, quantum computing has become a prominent field of research, particularly as hardware developers have successfully deployed several physical quantum computers. These machines are essential for validating the theories that, until recently, existed only on „white paper“ or within classical simulators. While the roots of quantum mechanics stretch back to the early 20th century, becoming a cornerstone of nuclear physics during the conflicts of the 1900s, the specific vision for quantum computing began to crystallize in 1981. During the First Conference on the Physics of Computation, Paul Benioff and Richard Feynman introduced the concept of a quantum Turing machine [5], sparking a revolution in how we think about information.

Progress was initially modest; the first quantum computer, introduced in 1998, boasted a mere 2 qubits. By 2005, we reached 12 qubits. Today, we handle hundreds of qubits and are actively pursuing the thousand-qubit milestone. However, we may ask ourselves: is that enough? We are currently in the era of Noisy Intermediate-Scale Quantum (NISQ) devices, where a significant portion of our qubit count is dedicated not to computation itself, but to error compensation and noise reduction. Thus, one logical qubit can be composed of multiple physical qubits, as detailed in Chapter 6. Furthermore, even the count of logical qubits is insufficient to encode practical problems, as demonstrated in the real problem qubit requirements calculation for Grover’s algorithm oracle in Section 4.2.7.

But to understand why these noisy machines are still so revolutionary, we must look at the root of the technology. Quantum computing differs from classical computing in almost every way, except for one: the final representation of the result. To understand the rest, we have to start with the qubit.

1.1 The Qubit

In the classical world, information is binary. A bit is like a light switch: it is either 0 or 1. If you want to describe a bit, you just need a single value. A qubit, however, is the basic unit of quantum information, and it behaves according to the laws of quantum mechanics [35].

Instead of being a switch, think of a qubit as a point on a sphere. We use Dirac notation, those „bra-ket“ symbols, to name these states. A classical 0 becomes the quantum state $|0\rangle$ (pronounced „ket zero“), and a 1 becomes $|1\rangle$.

Superposition

Where a classical bit is stuck at the poles (0 or 1), a qubit can exist in a superposition. This isn't just „being in two places at once“; mathematically, it is a linear combination of states. For example, the state $|+\rangle$ is an equal mix of $|0\rangle$ and $|1\rangle$:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

When we measure this qubit, it collapses into a classical 0 or 1. According to Born's Rule, the probability of the outcome depends on the „shadow“ the state casts on the axis we are measuring. If we measure the $|+\rangle$ state along the Z-axis, it casts an equal shadow on both poles, giving us a 50% chance of seeing a 0 and a 50% chance of seeing a 1.

Phase: The Quantum Sign

If we rotate our qubit further around the circle, we encounter phase. While a global phase, multiplying the whole state by -1, doesn't change what we see when we measure, a relative phase changes how the $|0\rangle$ and $|1\rangle$ parts of the qubit interact. For instance, the $|-\rangle$ state looks almost identical to the $|+\rangle$ state, but with a minus sign:

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

This tiny difference in phase is what allows quantum algorithms to use interference, in the other words, making unwanted answers cancel each other out while wanted answers reinforce one another.

Tensor Products and Entanglement

While a single qubit is interesting, the real magic happens when we bring them together. In classical computing, if you add a bit, you just add one more slot of memory. In quantum computing, adding a qubit doubles the size of the computational space. On the background level, we combine qubits using a mathematical operation called a tensor product ($\otimes$). If you have two qubits, their combined state is a vector in a four-dimensional space (2^2). If you have 300 qubits, the number of directions the system can point in is greater than the number of atoms in the observable universe. This leads us to entanglement, when qubits become so interconnected that they can no longer be described separately from one another. In this entangled state, a measurement on one qubit can instantaneously affect the state of its partner, regardless of the distance between them. This is the engine that drives quantum advantage, allowing us to process information in ways that would take a classical supercomputer billions of years to replicate.

1.2 Thesis Objectives and Scope

While the theoretical potential of entanglement is vast, the path from „white paper“ theory to a functioning implementation on modern hardware is paved with the practical hurdles

of the NISQ era. This thesis bridges that gap by applying these foundational principles to tangible combinatorial challenges.

The primary use case demonstrates how classical optimization techniques can be integrated with quantum search algorithms to address the Exact Cover Problem (ECP) within the context of resource scheduling. Furthermore, this research investigates the current boundaries of quantum hardware by exploring error mitigation and correction strategies. These investigations aim to identify the practical execution limits of modern NISQ devices and determine how decoherence affects the success of specific algorithmic approaches.

For experimental evaluation, this study utilized the IQM Star24 architecture (via the VLQ at IT4Innovations) alongside the IBM Heron r2 processor. To provide a comprehensive benchmarking suite, other NP-complete challenges, including the Knapsack Problem and the Boolean Satisfiability Problem (SAT), were also implemented and evaluated. The experimental framework leverages both the Qiskit [20] and Qrisp [44] frameworks, allowing for a comparative analysis of high-level algorithmic abstractions versus gate-level control. Additionally, the AerSimulator was employed with custom noise models to validate hardware performance and assess the impact of environmental noise on algorithmic success.

1.3 Motivation

The primary motivation for this work is to provide a practical reference for developing hybrid quantum applications based on existing quantum algorithms. While theoretical descriptions of quantum algorithms are available in the literature, the process of translating these algorithms into working implementations on real or simulated quantum hardware presents challenges that are not always explicitly documented. These include problem formulation, qubit mapping, oracle construction, error handling, and integration with classical computational components.

This thesis addresses these practical aspects by presenting a complete implementation workflow. The final chapter provides a concise guide summarizing the key steps, intended to help future researchers and practitioners acquire the necessary knowledge more efficiently and contribute to the developing field of quantum computing.

Chapter 2

Overview of Current Quantum Hardware

The current quantum computing market offers several competing approaches to qubit realization, each with distinct physical principles and application domains. Table 2.1 provides comprehensive technical specifications for these platforms. This section provides a concise overview of the fundamental physics, architectural differences, and primary use cases associated with each type of currently available quantum hardware. Given the experimental nature of this cutting-edge technology and the strong emphasis on research, smaller 5-qubit systems are also made available to academic institutions and small research groups as a cost-effective entry point to quantum computing hardware.

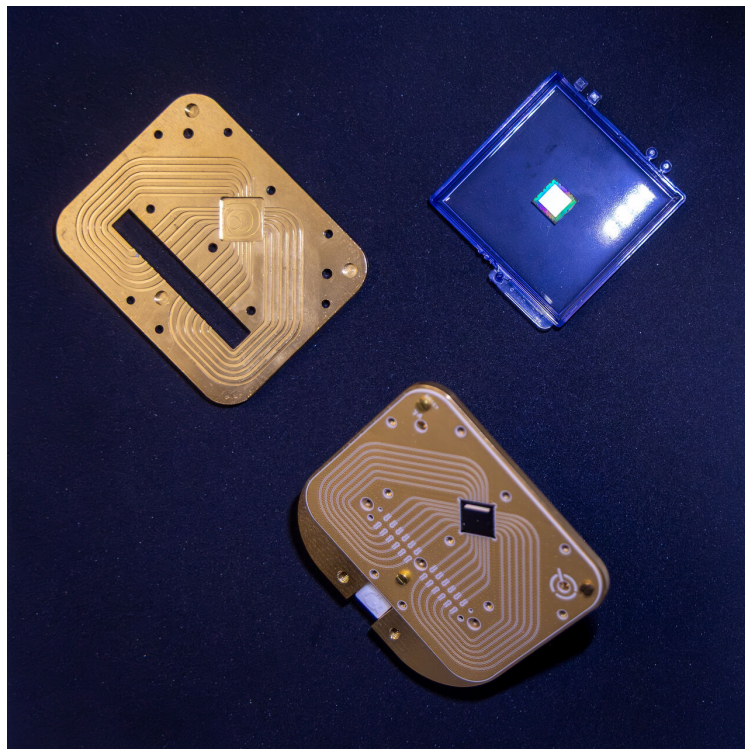


Figure 2.1: Quantum chip of the IQM Star24 (VLQ) processor. *Source:* <https://www.it4i.cz/en/infrastructure/vlq-quantum-computer>

2.1 Available Hardware and Purpose Comparison

Superconducting Qubits: General-Purpose Quantum Computing

Superconducting qubits represent the most mature and commercially dominant platform. Both D-Wave's flux qubits and gate-based systems from IBM and Google utilize Josephson junction physics [39] to create controllable quantum two-level systems. The key advantage of gate-based superconducting systems (IBM Osprey: 433 qubits [19], IBM Heron R2: 156 qubits [19], Google Willow: 105 qubits [16], IQM Crystal150: 150 qubits [28], IQM Star24: 24 qubits [28, 48]) is their flexibility for implementing arbitrary quantum algorithms through sequences of quantum gates.

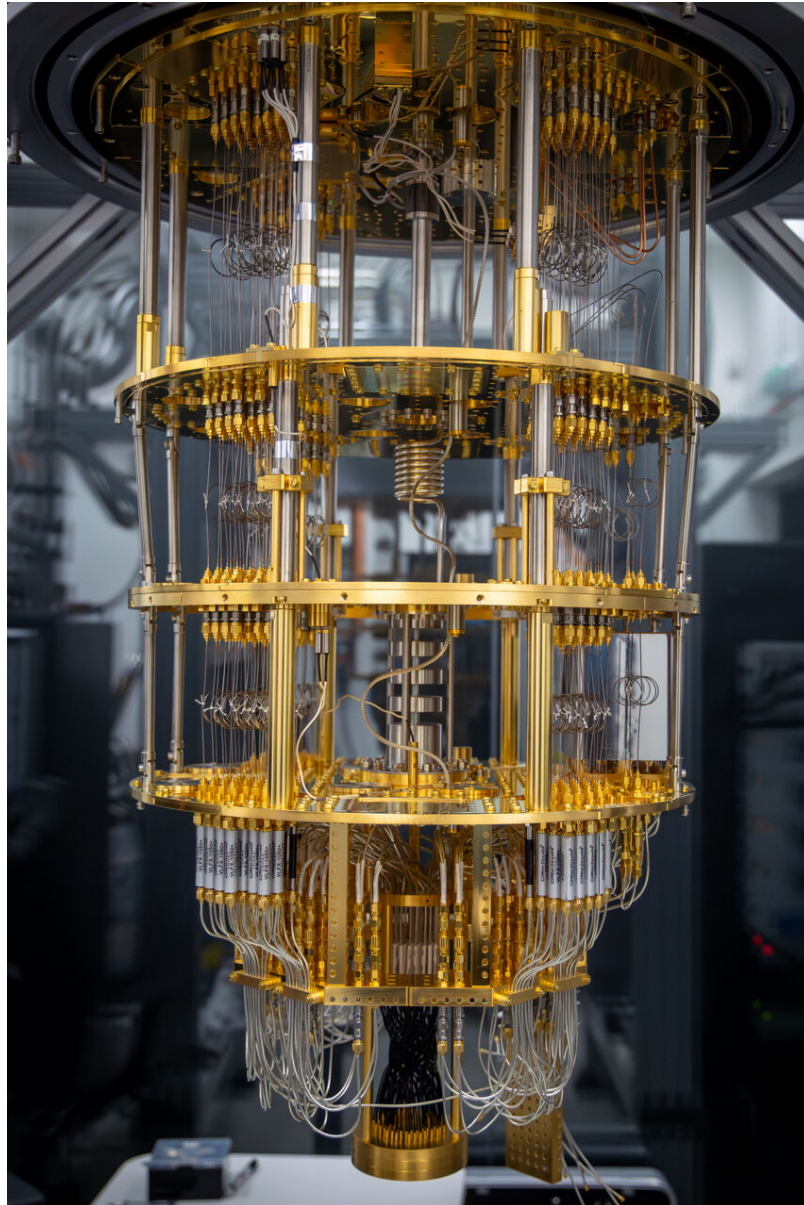


Figure 2.2: Cryogenic refrigeration system for the IQM Star24 (VLQ) quantum processor.
Source: <https://www.it4i.cz/en/infrastructure/vlq-quantum-computer>

The physical mechanism relies on nonlinear inductance provided by Josephson junctions, which enables qubit manipulation via microwave pulse sequences. A significant recent breakthrough is Google’s demonstration of exponential error suppression in surface codes, achieving below-threshold operation where logical error rates decrease with increasing code distance [15]. The transition from theoretical error correction to physical implementation represented by this milestone is a critical step in the development of practical quantum algorithms, as it addresses the primary barrier to fault-tolerant computation.

Notably, the surface code is specifically optimized for 2D lattice topologies, as its stabilizer measurements rely exclusively on local, nearest-neighbor interactions. While this connectivity facilitates scalable chip layouts for high qubit counts, it imposes significant routing constraints for more complex gate sequences. In contrast, architectures like the IQM Star utilize a central resonator to facilitate mediated all-to-all connectivity with a constant overhead of one state move from qubit to resonator, compared with lattice topology, where multiple state swaps are required for entanglement of two randomly selected qubits. However, the net operational advantage of star topology regarding circuit depth and noise scaling remains a subject of ongoing investigation.

Trapped Ions: High-Fidelity Simulation and Precision Computing

Trapped-ion systems, such as the IonQ Forte (30 qubits) [22] and Quantinuum H2-1 (56 qubits) [47], employ a fundamentally different physical principle: individual ions confined in electromagnetic traps and manipulated by laser pulses. A critical advantage of this platform is native all-to-all connectivity, where any ion pair can interact directly¹. This eliminates the need for intermediate routing or SWAP gate overhead. This architectural flexibility, combined with high gate fidelities (typically exceeding 99.9%), makes trapped ions particularly suited for deep circuits and complex entangling patterns where error accumulation from routing overhead would otherwise be prohibitive.

The EuroHPC JU ecosystem also features the Lucy system (Figure 2.3), a photonic quantum computer developed by Quandela and hosted at the TGCC (CEA) in France. Based on the MOSAIQ-12 architecture, Lucy utilizes single photons as qubits within reconfigurable optical circuits to perform quantum computations. Unlike matter-based qubits, these photonic flying qubits are largely immune to thermal decoherence, allowing the system to operate with high connectivity and stability within the European quantum infrastructure.

¹In a trapped-ion chain of L ions, mutual Coulomb repulsion couples all ions together, creating a system with $3L$ normal modes of oscillation—one for each spatial degree of freedom of each ion. When a laser pulse addresses a single ion, it does not isolate that ion; instead, the pulse can excite any of these shared collective motional modes, in which all ions participate simultaneously but with different phase relationships. For instance, the center-of-mass mode causes all ions to oscillate in phase like a single effective particle, while other modes involve out-of-phase oscillations. Because every ion couples to every mode, addressing any single ion can coherently affect the quantum state of the entire chain, enabling direct interactions between arbitrary ion pairs. This is the physical mechanism underlying the all-to-all connectivity [51]



Figure 2.3: The Lucy quantum computer, a photonic system developed by Quandela and hosted at the TGCC (CEA), France. Source: https://www.eurohpc-ju.europa.eu/eurohpc-quantum-computers/our-quantum-computers_en. Author: Agence Oblique/-Cyril Marcilhacy

The physical basis relies on laser-cooled ions occupying hyperfine or optical electronic states. Quantum gates are implemented through precisely timed laser pulses that create controlled interactions via resonant forces. Operation at room temperature eliminates the need for complex cryogenic infrastructure, reducing operational overhead. The primary application domain includes quantum simulation of molecular and materials systems, VQA for chemistry, and optimization problems where gate fidelity is more critical than qubit count.

Quantum Annealers: Combinatorial Optimization

D-Wave’s quantum annealers represent a specialized approach fundamentally different from gate-based systems. Rather than implementing arbitrary quantum circuits, annealers solve optimization problems through adiabatic evolution: the system begins in a uniform superposition and gradually transforms toward a Hamiltonian encoding the optimization problem. Quantum tunneling through energy barriers allows the system to escape local minima more effectively than classical simulated annealing.

The physical realization uses flux qubits (superconducting loops) [39] with tunable couplings and magnetic fields. The system is designed specifically for problems expressible as Ising models or quadratic unconstrained binary optimization (QUBO) [7]. With 4,400+ qubits [11], D-Wave systems can address large combinatorial problems in logistics, finance, and materials discovery. However, the lack of error correction and the problem-specific nature mean annealers complement rather than replace gate-based systems for universal quantum computation.

Photonic Quantum Computing: Modular Scalability

Xanadu's Aurora [3] represents an emerging approach using squeezed photon states as quantum resources. Rather than storing quantum information in a few highly-controlled qubits, photonic systems encode information in continuous variables using quadrature measurements. The 12 physical qubits (see Section 3.8.3) in Aurora are distributed across 35 photonic chips connected via 13 km of fiber optics, a modular architecture enabling future scaling to thousands of processing units.

While photonic systems excel in modularity, the EuroHPC JU ecosystem also utilizes stationary matter-based architectures such as PIAST-Q (Figure 2.4). This trapped-ion system, hosted at the Poznań Supercomputing and Networking Center (PCSS) in Poland, provides a high-fidelity alternative where qubits are suspended in electromagnetic fields and manipulated via precision lasers, offering a different set of advantages regarding coherence and gate connectivity.



Figure 2.4: The PIAST-Q quantum computer, a trapped-ion system developed by AQT and hosted at the Poznań Supercomputing and Networking Center (PSNC), Poland. Source: https://www.eurohpc-ju.europa.eu/eurohpc-quantum-computers/our-quantum-computers_en. Author: PSCN

The physical mechanism exploits nonlinear optical effects (squeezed light generation via four-wave mixing) and linear optical interference. A unique advantage is room temperature operation of most subsystems (only photon detectors require cryogenic cooling at 100 mK), dramatically reducing infrastructure costs. Xanadu recently demonstrated key components for fault tolerance: real-time adaptive measurements with single-cycle latency and cluster-state entanglement across separate chips [3]. The modular design with fiber-

optic interconnects positions photonic systems for distributed quantum networking, not just computation.

2.1.1 Application-Driven Platform Selection

Table 2.1 shows that platform selection depends critically on problem requirements:

- **Superconducting:** Optimal for quantum supremacy demonstrations, quantum simulation of few-qubit systems, and NISQ-era algorithms. Recent progress toward error-corrected logical qubits makes this the lead platform for eventual practical quantum computing.
- **Trapped ions:** Superior choice for quantum chemistry, materials simulation, and variational algorithms where gate fidelity and all-to-all connectivity outweigh qubit count limitations. The long coherence times enable deep circuit depths impossible on superconducting systems.
- **Quantum annealers:** Quantum annealers are specialized for combinatorial optimization and Ising problems but are not universal quantum computers. Practical advantages over classical approaches remain unproven.
- **Photonic:** Promising long-term platform emphasizing modularity and room temperature operation. Still experimental but addressing fundamental scalability challenges through distributed architecture and fiber-optic networking.

The diversity of approaches reflects the quantum computing field’s current stage: no single platform has demonstrated decisive superiority across all metrics. Investment across multiple physical platforms reduces technological risk and likely ensures that progress in one approach (e.g., error correction on superconducting systems) benefits others through shared algorithmic and theoretical insights.

As summarized in Table 2.1, the global quantum computing landscape comprises diverse physical modalities, many of which are being integrated into the European high-performance computing infrastructure. All the aforementioned types of quantum architectures are either currently available or will eventually be deployed under the EuroHPC Joint Undertaking (EuroHPC JU). However, because the physical deployment, calibration, and integration of these state-of-the-art machines into existing supercomputing centers is a phased and ongoing procedure, immediate research access to the complete EuroHPC portfolio remains highly restricted. Consequently, for the practical purposes of this thesis, experimental validation was conducted utilizing systems that are currently accessible to the author. Specifically, the experiments leverage IQM Star24 (VLQ) at IT4Innovations and IBM Marrakesh, a superconducting processor publicly accessible via cloud infrastructure but subject to strict computational time limits. The specific accessibility status of these platforms is detailed in Table 2.2.

2.2 Comparison of Selected Quantum Processing Unit Architectures

This section provides a comparative analysis of three superconductive quantum processing unit (QPU) architectures relevant to European HPC-QC integration: the IQM Crystal

Table 2.1: Comparative technical specifications for leading quantum computing platforms for year 2025.

Aspect	Quantum Annealing (D-Wave)	Gate-Based (Superconducting)	Gate-Based (Trapped Ion)	Photonic QC (Xanadu / Others)
Companies	D-Wave ^[11]	IBM, IQM, Google, Rigetti ^[19, 29, 15]	IonQ, Quantinuum ^[22, 47]	Xanadu, Quandela, PsiQuantum ^[3]
Physical Platform	Superconducting flux qubits ^[11]	Transmon superconducting qubits ^[19]	Trapped ions in EM fields ^[22]	Photons (squeezed light) ^[3]
Qubit Type	Josephson junction SQUIDs ^[11]	Superconducting LC circuits ^[19]	Hyperfine/optical states ^[22]	Continuous-variable photon states ^[3]
Current Qubit Count	4,400+ ^[11]	5–433 ^[15, 19, 21]	30–56 ^[22, 47]	12 physical qubits ^[3]
Operating Temperature	$\approx 15\text{--}20\text{ mK}$ ^[19]	$\approx 15\text{--}20\text{ mK}$ ^[15]	Room temperature ^[47]	Room temperature ^[3]
Coherence Time	Microseconds (limited) ^[11]	30–100 μs ^[15, 19]	0.2 s (optical) to 600+ s (hyperfine) ^[22]	Room temp, no decoherence ^[3]
Gate Operation Time	200–1000 μs (anneal) ^[11]	20–200 ns ^[19]	1–100 μs ^[22]	Picoseconds–nanoseconds ^[3]
Gate Fidelity	$\approx 99\%$ ^[11]	99.5–99.9% (1Q), 98–99% (2Q) ^[19]	$> 99.9\%$ (best in industry) ^[47]	90–95% (improving) ^[3]
Measurement Fidelity	$\approx 98\text{--}99\%$ ^[11]	$\approx 95\text{--}98\%$ ^[19]	$> 99.9\%$ ^[22]	$\approx 95\text{--}98\%$ ^[3]

20-qubit system deployed at the Leibniz Supercomputing Centre (LRZ) as part of the Q-Exaproject [25], the IQM Radiance Star 24-qubit system contracted for IT4Innovations within the LUMI-Q consortium [27], and the IBM Heron processor family [18] representing the current state-of-the-art in gate-based superconducting quantum computing. The QPUs were selected based on their accessibility for academic research purposes within the European Union. Under LUMI-Q consortium are all mentioned types accessible for science and research.

2.2.1 System Overview

LRZ Q-Exa (IQM Crystal 20)

The Q-Exa system, inaugurated in June 2024, represents Germany’s first hybrid quantum-classical computer, integrating a 20-qubit IQM processor with the SuperMUC-NG supercomputer at LRZ [25]. The system achieved the following benchmarks upon deployment:

Table 2.2: Accessibility and pricing models for commercial and academic/research quantum computing platforms. Abbreviations: SC = Superconducting

Quantum System	Modality	Accessibility	Pricing Model
<i>Commercial Platforms</i>			
D-Wave Advantage2	Quantum annealing	Public cloud (Leap service)	Monthly subscription + usage credits
IBM Quantum	SC	Public cloud (open access)	Free tier (free 10 minutes); pay-per-use for priority access
Google Sycamore/Willow	SC	Restricted (invitation-only)	Free for approved research partners; no commercial pricing
IonQ Aria/Forte	Trapped-ion	Public cloud (Azure Quantum, Google Cloud)	On-demand or reserved time; free simulator; research credits on request up to \$10k
Quantinuum H2/Helios	Trapped-ion	Public cloud (Nexus, Azure Quantum)	Hardware Quantum Credits (HQCs); commercial/development cycle pricing
Xanadu (Photonic)	Photonic	Limited (curated onboarding)	Free simulator; hardware via API key (request-based); AWS Braket (interface) integration
Rigetti Aspen	SC	Public cloud (Azure Quantum)	Pay-per-use via cloud partners
<i>Academic and Research Platforms (EuroHPC)</i>			
Lucy (Quandela)	Photonic	CEA access	HPC allocation-based; project grant priority queuing
Piast-Q (AQT)	Trapped-ion	PCSS access	HPC allocation-based; project grant priority queuing
VLQ (IQM Star24)	SC	IT4Innovations access	HPC allocation-based; project grant priority queuing
Euro-Q-Exa (IQM)	SC	LRZ access	HPC allocation-based; project grant priority queuing
EuroQCS-Spain (Qilimanjaro)	SC annealing	BSC access	HPC allocation-based; project grant priority queuing

- Median two-qubit (CZ gate) gate fidelity of 99.46% across 30 qubit pairs, with maximum fidelity reaching 99.74% on a single pair
- Median single-qubit gate fidelity of 99.94% across 20 qubits
- Quantum Volume of 32^2
- Entangled 19-qubit GHZ state without readout error mitigation; fully entangled 20-qubit GHZ state with readout error mitigation

The IQM Crystal 20 QPU utilises a square lattice topology with tunable couplers, containing 20 computational qubits and 30 tunable coupler³ qubits (50 physical qubits total) [28]. Reference benchmarks on the IQM 20-qubit platform (February 2024) demonstrated median CZ gate fidelity of 99.51%, CLOPS of 2600, and Q-score of 11 [26].

IT4I LUMI-Q (IQM Radiance Star 24)

The LUMI-Q consortium’s quantum computer, contracted in September 2024 for installation at IT4Innovations National Supercomputing Centre in Ostrava, Czech Republic, represents the first deployment of IQM’s proprietary star topology architecture [27, 13, 48]. Key characteristics include:

- 24 physical qubits connected to a central resonator in star-shaped topology [13, 48]
- Integration with the EuroHPC petascale supercomputer Karolina [27]

The star topology minimises the number of swap operations required between qubits, enabling execution of complex quantum algorithms more efficiently [13, 8]. According to IT4Innovations, the architecture offers optimal connections between qubits, with the aim of minimising error rates and increasing reliability of quantum operations [31].

Note: As of the contract announcement (September 2024), specific performance benchmarks (gate fidelities, Q-score) for the Star 24 topology have not been publicly disclosed. The system became operational in 2025 under the name VLQ [32].

IBM Heron Processor Family

The IBM Heron processor, introduced at the IBM Quantum Summit 2023 and upgraded to revision R2 at the Quantum Developer Conference (QDC) 2024, represents IBM’s current flagship architecture [17, 18]:

Heron R1 (December 2023): 133 fixed-frequency qubits with tunable couplers, delivering 3–5× performance improvement compared to the 127-qubit Eagle processors while virtually eliminating crosstalk [18].

²Quantum Volume (QV) is a holistic, hardware-agnostic metric used to quantify the computational capability of a quantum processor. A value of $QV = 32$ specifically indicates that the device can successfully execute a square circuit, where the number of qubits (n) equals the gate depth (d): $n = d = 5$, such that $QV = 2^n = 32$. This metric accounts for gate fidelities, qubit connectivity, and crosstalk, providing a more comprehensive performance measure than raw qubit count alone.

³A tunable coupler is a programmable circuit component that dynamically controls the coupling strength between adjacent qubits, serving as a hardware-level switch to activate high-fidelity two-qubit operations while isolating qubits to suppress parasitic ZZ -crosstalk and signal interference during idle periods.

Heron R2 (July 2024): 156 qubits in heavy-hexagonal lattice topology with the following demonstrated capabilities [17]:

- Execution of quantum circuits with up to 5000 two-qubit gate operations, nearly doubling the 2880 two-qubit gate depth achieved in the 2023 utility experiment [17]
- Circuit Layer Operations Per Second (CLOPS) exceeding 150,000, compared to 37,000 CLOPS earlier in 2024 and 950 CLOPS in 2022 [17]
- Two-level system (TLS) mitigation feature for improved coherence and stability [18]
- 50× speedup demonstrated: workloads requiring 122 hours on previous systems completed in approximately 2.4 hours [17]

Heron R3 (July 2025): 156 qubits in heavy-hexagonal lattice topology with manufacturing improvements that directly impact coherence, gate fidelity, and readout performance [18].

2.2.2 Comparative Specifications

Table 2.3 summarises the key specifications of the three architectures. Values marked with asterisks indicate estimates or values not directly comparable across platforms.

Table 2.3: Comparative specifications of QPU architectures for European HPC integration. Values from primary manufacturer sources; empty cells indicate data not publicly available.

Parameter	LRZ Q-Exa (Crystal 20)	IT4I LUMI-Q (Star 24)	IBM Heron R2
Qubits (computational)	20	24	156
Physical qubits (total)	50	—	156
Topology	Square lattice	Star (central resonator)	Heavy-hex
Native 2Q gate	CZ gate	CZ gate	CZ gate
Median 2Q fidelity	99.46%	—	~99.5%*
Median 1Q fidelity	99.94%	—	>99.9%*
Quantum Volume	32	—	512+
CLOPS	2600	—	>150,000
Max circuit depth (2Q)	—	—	5000 gates
Tunable couplers	Yes	Yes	Yes
HPC integration	SuperMUC-NG	Karolina	—
Operational	June 2024	2025	July 2024

2.2.3 Architecture Analysis

Topology Considerations

The three systems represent distinct approaches to qubit connectivity:

Square Lattice (IQM Crystal): The square lattice topology with tunable couplers provides regular nearest-neighbour connectivity. Each computational qubit connects to up to four neighbours via tunable couplers, supporting surface-code error correction schemes

[28]. The layout is rotated 45 degrees with respect to the crystal edge, with qubit-qubit connectivity mediated by tunable couplers enabling distances up to 2 mm [28].

Star Topology (IQM Radiance Star): The star architecture connects all 24 qubits to a central resonator, providing effective all-to-all connectivity. This topology minimises swap operations for algorithms requiring global entanglement, though specific benchmark data remains to be published [27, 13].

Heavy-Hexagonal (IBM Heron): IBM’s heavy-hex topology arranges qubits where each connects to two or three neighbours, resembling edges and corners of tessellated hexagons. This design reduces errors from interactions between adjacent qubits while maintaining compatibility with surface-code error correction [18, 1].

Gate Performance and Error Rates

The IQM systems demonstrate median two-qubit CZ gate fidelities at or above 99.46%, with the reference 20-qubit system achieving 99.51% [25, 26]. IBM’s Heron architecture, while not publishing explicit median gate fidelities in the same format, emphasizes circuit-level performance metrics, demonstrating reliable execution of 5000 two-qubit gates [17].

Direct comparison of gate fidelities across platforms requires caution due to differences in benchmarking methodologies, native gate sets, and reporting conventions.

Throughput and Scalability

The CLOPS metric reveals significant architectural differences in classical-quantum coordination efficiency. IBM’s Heron R2 achieves over 150,000 CLOPS [17], approximately 60× higher than the IQM reference system’s 2600 CLOPS [26]. This difference reflects both hardware capabilities and software stack optimization, including IBM’s parametric compilation and runtime improvements [17].

2.2.4 Implications for Hybrid HPC-QC Workloads

For hybrid quantum-classical algorithms relevant to HPC centers:

1. **Variational algorithms:** The high CLOPS of IBM Heron R2 enables fast iteration for variational quantum eigensolvers (VQE) and quantum approximate optimization algorithms (QAOA), where parameter updates require many circuit executions [17].
2. **Algorithm-specific advantages:** The star topology’s reduced swap requirements may benefit algorithms with global entanglement patterns, while regular lattice topologies align with spatially-local Hamiltonian simulations [27].
3. **On-premises integration:** Both IQM systems are designed for tight HPC integration with low-latency classical-quantum coordination, essential for hybrid workflows [25, 31].

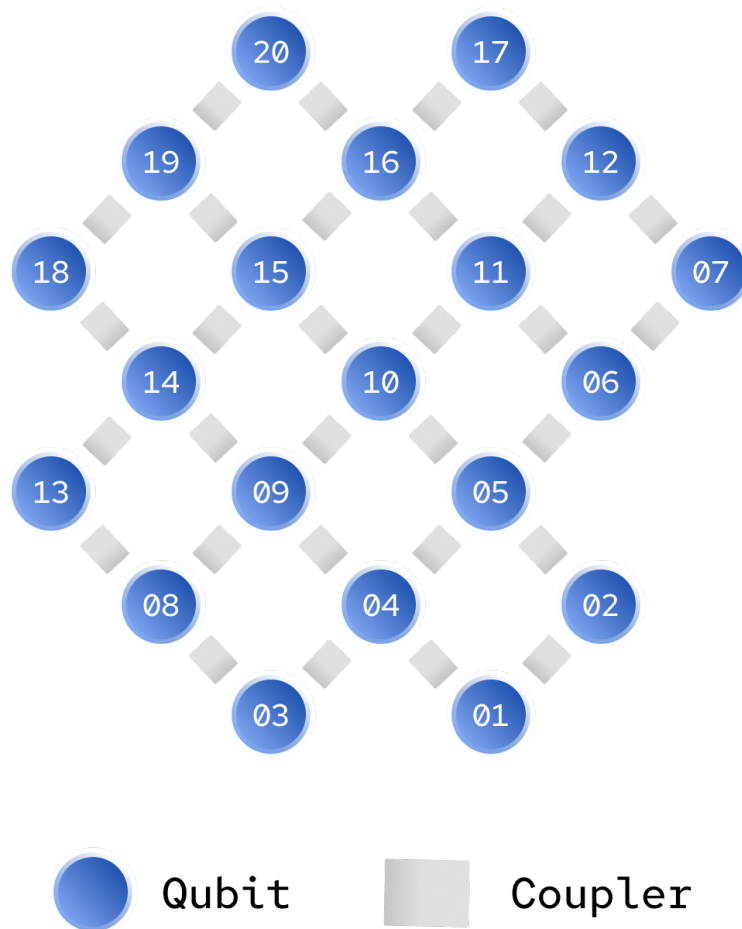


Figure 2.5: IQM Crystal 20-qubit chip design

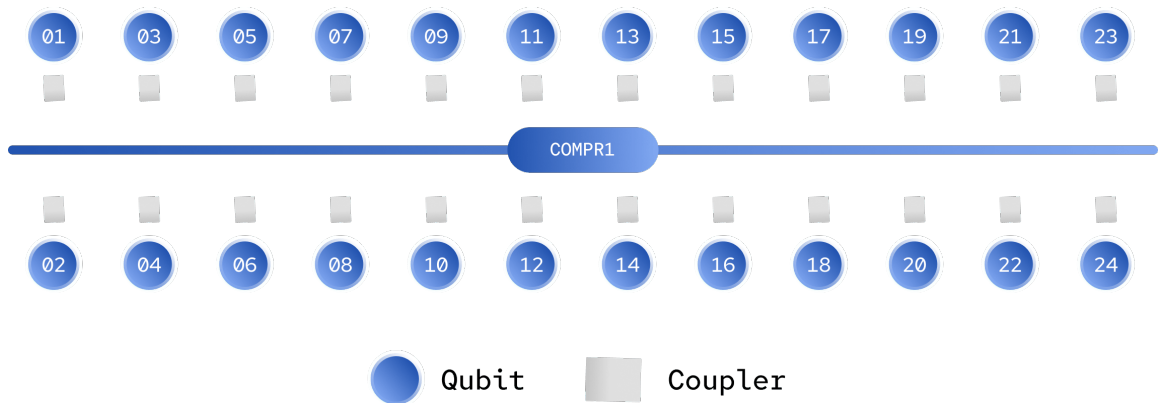


Figure 2.6: IQM Radiance Star 24-qubit chip design

Figure 2.7: Comparison of IQM superconducting quantum processor chip designs: Crystal 20 and Radiance Star 24. Source: IQM Quantum Computers, QPU technology and tech stack documentation [28].

Chapter 3

Exploring Quantum Computing Platforms and Languages

Quantum programming fundamentally differs from classical programming due to its basis in quantum mechanics principles. This has necessitated specialized tools across two distinct paradigms: gate-based quantum computing and quantum annealing.

Gate-based quantum computers operate through quantum circuits composed of gates applied sequentially to qubits. Like classical computers executing instruction sequences, gate-based systems manipulate quantum information through well-defined gate operations. On physical hardware, each gate corresponds to a precisely timed sequence of control signals that modifies qubit state. In simulation, the same operation applies matrix transformations to the quantum state vector.

All gate-based implementations share fundamental components: quantum registers storing qubits and classical registers storing measurement outcomes. This architecture mirrors classical computing paradigms while exploiting quantum mechanical properties.

Several Python-based frameworks have become standard in gate-based quantum computing, leveraging Python’s widespread adoption to lower adoption barriers. Qiskit [20], developed by IBM, is among the most widely deployed frameworks, offering circuit definition, simulation, and execution on quantum hardware. PennyLane [52], developed by Xanadu, provides cross-platform abstractions for differentiable quantum computing and quantum machine learning. Notably, PennyLane extends support to photonic platforms where quantum information encodes in light properties, enabling exploration of continuous-variable quantum computing alongside discrete-variable approaches used by superconducting and trapped-ion systems.

Qrisp [44] offers a high-level Python abstraction layer integrating quantum operations with classical control flow, including conditionals, constraints, modulus operations, and reversibility management. The framework provides built-in implementations of canonical algorithms (QAOA, quantum backtracking, Grover’s algorithm) with well-documented source code and clear interfaces to multiple execution backends.

While domain-specific languages such as Q# [50] and OpenQASM [10] provide direct circuit specification, the complexity of designing and maintaining dedicated languages has made Python-based frameworks the practical standard for most practitioners.

Quantum annealers represent a fundamentally different computational model, specialized for optimization rather than general-purpose quantum computation. These devices

locate ground states of problem Hamiltonians, making them particularly effective for combinatorial optimization tasks.

D-Wave [12] systems exemplify this approach. The D-Wave Ocean SDK enables users to formulate optimization problems as QUBO or Ising models, which are then mapped directly to annealer hardware. While less general than gate-based systems, quantum annealers excel at problems naturally expressible in these formulations, offering practical advantages for specific optimization classes.

Cross-platform portability presents a critical consideration for hybrid quantum-classical workflows. OpenQASM [10] provides a hardware-agnostic intermediate representation for quantum gate-based circuits, enabling circuit portability across diverse backends. This standardization is essential for workflows spanning multiple quantum platforms or transitioning between vendors, reducing vendor lock-in and supporting modular algorithm development across heterogeneous quantum resources.

Frameworks implementing OpenQASM compatibility, including Qiskit and others, facilitate this portability while maintaining backend-specific optimizations when executing on particular hardware.

In this section, we present a foundational gate-based quantum algorithm that creates an equal superposition of computational basis states using a Hadamard gate and then measures the result. When the Hadamard gate is applied to the qubit, it enters a superposition state that is an equal combination of $|0\rangle$ and $|1\rangle$. However, measurement forces this superposition to collapse into a definite classical outcome. Since the superposition is balanced equally between both basis states, the measurement yields 0 or 1 with equal likelihood.

This example will be implemented across multiple quantum programming frameworks: Qiskit, Q#, Qrisp, and OpenQASM. Each platform expresses the same underlying quantum operation in different ways: allocating a qubit, applying a Hadamard transformation to generate superposition, collapsing the state through measurement, and resetting the qubit for reuse. By examining this simple yet essential algorithm across these frameworks, you will gain insight into the similarities and differences in how modern quantum programming languages handle core quantum operations.

3.1 Q#

Q# [50] is a high-level quantum programming language developed by Microsoft based on C# with extensions and distributed as part of the Azure Quantum Development Kit. It is a proprietary language intended for the implementation of quantum algorithms together with classical control logic. Q# is algorithm-oriented rather than circuit-oriented, which makes it particularly suitable for hybrid quantum-classical computing where quantum routines are integrated with classical processing. Q# programs are typically executed using built-in local simulators or quantum hardware accessed through the Azure Quantum platform.

The language provides constructs for defining quantum operations, functions, and control flow. Qubits are explicitly allocated and released, which allows precise control over their lifetime. This design reduces programming errors related to unintended qubit reuse, because a qubit cannot be accessed outside the scope in which it was allocated, and its state must be returned to a known basis state before release.

The Q# language and its standard library are actively developed. The Azure Quantum Development Kit (QDK) [40] underwent a major modernization in version 1.0 released in January 2024, which introduced significant changes to the language syntax and standard library structure. In late 2025, version 1.13 introduced native Q# unit testing with the

`@Test` attribute and integration with the VS Code Test Explorer, along with improved support for importing Q# libraries from GitHub repositories. As a consequence of active development, the available namespaces and operations may change between SDK versions, which requires attention when maintaining or extending existing code.

```
// Start a new quantum procedure (operation) that will give us
// a result 0 with 50% probability or 1 with 50% probability
operation MeasureHadamard() : Result {

    // 1. "use" - Request a fresh qubit from the QPU's memory
    use q = Qubit();

    // 2. "H" - Apply a Hadamard gate; this puts the qubit
    // into superposition --
    // Imagine a coin spinning in the air, being both heads and tails at once
    H(q);

    // 3. "let r = M" - Measure the qubit;
    // this forces the 'spinning coin' to land,
    // collapsing the superposition
    // into a definite classical piece of data (r)
    let r = M(q);

    // 4. "Reset" - Clean up the qubit by returning it to its starting state
    // so it can be safely reused by the next program
    Reset(q);

    // 5. "return" - Send the final measurement value back
    return r;
}
```

Figure 3.1: A Q# operation that prepares a qubit in superposition and measures it.

Within the Microsoft QDK, the Q# language is supported by a dedicated chemistry library. This library provides a framework for implementing and benchmarking algorithms focused on molecular dynamics and quantum chemical simulations.

Q# programs can be executed from Python and integrated into Jupyter Notebook workflows, enabling interactive development and testing alongside classical Python code.

3.2 OpenQASM

OpenQASM [10] is an open and vendor-independent specification for describing quantum circuits developed by IBM. It defines a low-level language that represents quantum operations, measurements, and classical registers in a form close to the circuit execution model. OpenQASM is not intended to express full quantum algorithms, but rather to describe already constructed quantum circuits.

The main purpose of OpenQASM is interoperability. Quantum circuits can be serialized into OpenQASM and transferred between different quantum software frameworks and hardware providers. Both Q# and Qiskit support exporting quantum circuits into this format. This makes OpenQASM suitable for circuit exchange, inspection, and long-term storage.

```
OPENQASM 3.0; // Define the language version
include "stdgates.inc"; // Load the standard gate library

// 1. "qubit" - Declare a fresh qubit 'q' from the QPU's memory
qubit q;

// Declare a classical bit 'c' to store the measurement result
bit c;

// 2. "h" - Apply a Hadamard gate;
// this puts the qubit into superposition
// (Imagine a coin spinning in the air, being both heads and tails at once)
h q;

// 3. "measure" - Measure the qubit;
// this forces the 'spinning coin' to land,
// collapsing the superposition into a definite classical piece of data (c)
c = measure q;
```

Figure 3.2: A single-qubit superposition circuit written in OpenQASM 3.0.

OpenQASM 3.0, officially released in 2023, significantly extended the language with support for classical control flow, timing constraints, and pulse-level control, while preserving compatibility with the circuit model. Version 3.1 added support for the switch-case statement. The specification continues to evolve under the governance of an open-source Technical Steering Committee.

3.3 Qiskit

Qiskit [20] is an open-source quantum computing framework developed primarily by IBM. It is implemented as a Python library and provides tools for constructing quantum circuits, simulating their execution, and running them on real quantum hardware. Qiskit integrates quantum programming into the Python ecosystem, which simplifies interaction between quantum and classical code. The framework has become widely adopted for gate-based quantum computing, and its `QuantumCircuit` class is accepted by interfaces providing access to quantum hardware from other providers, including IQM and the Munich Quantum Software Stack.

Quantum computation in Qiskit is based on explicit construction of quantum circuits. The user defines quantum and classical registers and applies quantum gates to individual qubits. The resulting circuit can be executed on a simulator or submitted to a remote quan-

tum backend. Qiskit also supports exporting quantum circuits to OpenQASM, enabling interoperability with other tools.

```
#!/usr/bin/env python3
from qiskit import QuantumCircuit

# 1. Initialize a circuit with 1 qubit and 1 classical bit
# This requests a fresh qubit from the QPU's memory
qc = QuantumCircuit(1, 1)

# 2. Apply a Hadamard gate to the first qubit (index 0)
# This puts the qubit into superposition
# (Imagine a coin spinning in the air, being both heads and tails at once)
qc.h(0)

# 3. Measure the qubit and store the result in the classical bit
# This forces the 'spinning coin' to land,
# collapsing the superposition into a definite piece of data (0 or 1)
qc.measure(0, 0)
```

Figure 3.3: Python code using the Qiskit SDK to construct a quantum circuit with one qubit and measure it.

Qiskit has been actively developed since its initial release in 2017. Version 2.0, released in April 2025, introduced a new C API and removed deprecated features from the 1.x series. Version 2.2, released in September 2025, expanded the C API with a standalone transpiler function `qk_transpile()`, enabling the construction of complete quantum workflows in compiled languages such as C and C++. This release also introduced the `LitinskiTransformation` pass for fault-tolerant quantum computing compilation.

For beginner users, Qiskit provides visualization utilities for circuits and measurement results, which help with understanding the behavior of simple quantum programs. Moreover, not only the Qiskit framework itself, but also IQM and IBM quantum providers offer packages containing pre-prepared algorithms such as `iqm-qaoa` [24], similar to Qrisp's pre-implemented algorithms mentioned in 3.6.4.

3.4 D-Wave Ocean SDK

D-Wave Ocean [12] is an open-source Python SDK developed by D-Wave Systems for programming quantum annealers. Unlike gate-based frameworks, Ocean is designed for solving optimization problems using the quantum annealing paradigm. Users formulate problems as QUBO models or Ising models, which are then mapped to the physical qubit topology of D-Wave quantum annealers.

The SDK provides tools for problem formulation, automatic minor-embedding of logical variables onto the hardware graph, and hybrid classical-quantum solvers that combine quantum annealing with classical optimization techniques. Ocean also includes samplers for both quantum hardware access and classical simulation, allowing development and testing without immediate hardware access.

D-Wave Ocean has been continuously developed alongside D-Wave hardware generations. The SDK is regularly updated to support new hardware capabilities and solver improvements.

```
#!/usr/bin/env python3
from dimod import Binary
from dwave.system import DWaveSampler, EmbeddingComposite

# 1. Define logical binary variables (0 or 1)
# These represent the decisions in optimization problem
x1, x2 = Binary('x1'), Binary('x2')

# 2. Define the QUBO (Quadratic Unconstrained Binary Optimization)
# This mathematical expression creates an energy landscape
# The system will seek the values of x1 and x2 that minimize this sum
qubo = x1 + x2 - 2 * x1 * x2

# 3. "EmbeddingComposite" - Automatically map our logical problem
# onto the physical graph of the D-Wave superconducting chip
sampler = EmbeddingComposite(DWaveSampler())

# 4. "sample" - Start the physical annealing process 100 times
# The qubits cool down into the lowest energy states (the best solutions)
result = sampler.sample(qubo, num_reads=100)

# 5. Print the discovered solutions and their energy levels
print(result)
```

Figure 3.4: Python code using D-Wave Ocean SDK to define a simple QUBO problem and sample solutions.

3.5 PennyLane

PennyLane [52] is an open-source Python library developed by Xanadu for differentiable quantum computing and quantum machine learning. It provides a unified interface for constructing and executing quantum circuits across multiple hardware backends, including superconducting, trapped-ion, and photonic quantum computers. PennyLane is particularly notable for its native support of photonic quantum computing, where quantum information is encoded in continuous-variable properties of light such as squeezed states and photon number.

The distinguishing feature of PennyLane is its integration of quantum circuits with automatic differentiation frameworks such as PyTorch [42] and JAX [33]. This enables gradient-based optimization of parameterized quantum circuits, which is essential for VQA and QML applications.

PennyLane has been actively developed since its initial release in 2018. Version 0.43, released in October 2025, introduced dynamic wire allocation, enhanced resource estimation capabilities, and new lightweight operation representations for high-level resource estimates.

This release also added support for Python 3.13 and deprecated TensorFlow support, which will be removed in version 0.44.

```
#!/usr/bin/env python3
import pennylane as qml
from pennylane import numpy as np

# 1. "device" - Define the quantum backend
# (e.g., a local simulator or real QPU)
# 'wires=1' specifies that we are using a single qubit
dev = qml.device('default.qubit', wires=1)

# 2. "@qml.qnode" -
# A decorator that binds the Python function to the quantum device
# This converts a standard function
# into a Quantum Node that can execute on the device
@qml.qnode(dev)
def circuit(theta):
    # 3. "RX" - Apply a rotation around the X-axis by angle 'theta'
    # This transforms the qubit state based on the input parameter
    qml.RX(theta, wires=0)

    # 4. "expval" - Return the expectation value of the Pauli-Z measurement
    # This calculates the statistical average outcome
    # without collapsing the state
    return qml.expval(qml.PauliZ(0))

# 5. Execute the circuit with a specific rotation angle (0.5 radians)
theta = np.array(0.5)
result = circuit(theta)

# The result is a continuous value between -1 and 1
# For theta=0.5 radians, the expected result is approximately 0.877
# calculated as cos(0.5),
# since RX rotation around X-axis affects Z-measurement expectation
```

Figure 3.5: Python code using PennyLane to define a simple quantum circuit and compute its expectation value.

3.6 Qrisp Framework

The Qrisp framework [44] represents a significant advancement in the landscape of quantum programming by providing a high-level, Python-based environment that abstracts the complexities of gate-level circuit construction. In many traditional quantum languages, the developer is required to manage qubits at a level analogous to assembly programming. Qrisp, however, allows for the expression of quantum logic through familiar software en-

gineering paradigms, including typed variables, infix arithmetic, and automatic memory management.

3.6.1 Architectural Abstractions and Syntax

A primary objective of Qrisp is to bridge the gap between abstract algorithmic design and executable quantum circuits. The framework introduces several high-level constructs that simplify the development process:

- **Data Typing and Constraints:** Qrisp supports a variety of quantum data types, including characters and modular types, allowing for more complex data structures within a quantum state.
- **Classical Control Flow:** Developers can utilize standard Python control structures, such as loops and conditionals, which the framework then compiles into corresponding quantum operations.
- **Arithmetic Operations:** Through the use of infix notation, operations such as addition or multiplication can be applied directly to quantum variables, significantly reducing the verbosity of the code.

```

#!/usr/bin/env python3
from qrisp import QuantumCircuit, QuantumFloat, h

# 1. Initialize a Quantum Circuit environment
qc = QuantumCircuit()

# 2. "QuantumFloat" - Declare a high-level quantum variable
# (3, -1) sets 3 bits for the mantissa and -1 as the binary exponent
# This allows the variable to represent values in steps of 0.5 ( $2^{-1}$ )
qf = QuantumFloat(3, -1)

# 3. Apply Hadamard gates to the entire variable
# This puts all qubits representing the number into superposition
h(qf)

# 4. "qf + 2" - Perform quantum arithmetic
# Qrisp automatically synthesizes the complex gate logic for addition
qf_result = qf + 2

# 5.a Measure the high-level variable only
# This collapses the state and returns a Python float (e.g., 2.5)
# based on the (3, -1) configuration.
qf_measured = qf_result.measure()

# 5.b Measure the entire circuit context
# This returns a dictionary of raw bitstrings and their probabilities:
# e.g., {'101': 1.0}
qc_result = qc.measure()

```

Figure 3.6: Qrisp code demonstrating high-level quantum arithmetic and variable measurement using a QuantumFloat.

3.6.2 Theoretical Foundations of Reversibility

One of the most technically demanding aspects of quantum computing is the requirement for reversibility. To maintain quantum coherence and avoid unwanted entanglement—often referred to as the garbage qubit or any auxiliary operations must be uncomputed. Qrisp automates this process using a theoretical framework based on Reversible Calculus of Communicating Systems (RCCS) and Concurrent Calculus of Communicating Systems (CCCS) [4].

By employing automata-based logic to solve for the reversibility of operations, Qrisp ensures that algorithms requiring significant auxiliary space, such as Grover’s search algorithm, can be implemented without the manual overhead of designing uncomputation gates. This ensures the integrity of the quantum state throughout the execution of complex algorithmic pipelines.

3.6.3 Performance, Scalability, and Backend Integration

To handle the computational demands of large-scale circuit compilation, Qrisp utilizes a JAX-based backend known as Jasp. This integration allows the framework to scale to problem sizes that would typically exceed the memory or processing capabilities of standard circuit interpreters. Furthermore, Jasp enables the integration of real-time classical computation within the quantum program, which is essential for modern hybrid algorithms.

The framework is designed with interoperability in mind. While it provides its own set of tools, it remains compatible with various execution backends. For instance, Qrisp includes a prepared wrapper for Qiskit-based backends, allowing users to leverage the extensive hardware access provided by IBM and other vendors.

3.6.4 Pre-implemented Algorithms and Documentation

Qrisp is not merely a compiler but a comprehensive library of quantum solutions. It includes built-in, well-documented implementations of several cornerstone algorithms. To inspire, not all of them, but provide some inspiration, a selection is listed below:

- Shor’s Algorithm for integer factorization.
- Grover’s Algorithm for unstructured database searching.
- Quantum Approximate Optimization Algorithm (QAOA) for combinatorial optimization.
- Quantum Backtracking for solving constraint satisfaction problems.

The framework is supported by high-quality documentation with many working examples and transparent source code, making it an accessible and robust tool for both researchers and software engineers looking to explore the potential of quantum computation.

3.7 Comparison of Quantum Programming Environments

The landscape of quantum computing platforms is diverse, ranging from low-level hardware-description languages to high-level algorithmic frameworks. Table 3.1 summarizes the main characteristics of the quantum computing platforms and languages described in this chapter, including the newly discussed Qrisp framework.

Table 3.1: Comparison of quantum computing platforms and languages.

<i>Platform</i>	<i>Type</i>	<i>Portability</i>	<i>Problem Domain</i>	<i>Hardware Support</i>
Q#	DSL	Azure Quantum	Generic	Gate-based
OpenQASM	DSL	High	Generic	Gate-based
Qiskit	Library (Python)	High	Generic, QML	Gate-based
D-Wave Ocean	Library (Python)	D-Wave only	Optimization	Quantum annealer
PennyLane	Library (Python)	High	Generic, QML	Gate-based, Photonic
Qrisp	Library (Python)	High	Generic	Gate-based

The Type column indicates whether the platform is a domain-specific language (DSL) or a library built on top of an existing programming language. The Portability column

describes the range of quantum hardware and simulators that the platform can target. High portability means the platform supports multiple hardware providers. The Problem Domain column specifies the types of problems the platform is designed to address: generic refers to general-purpose quantum computation, optimization refers to combinatorial optimization problems, and QML refers to quantum machine learning. The Hardware Support column lists the types of quantum hardware the platform can target.

3.7.1 Synthesizing the Comparative Results

When analyzing these platforms, a clear distinction emerges between hardware-oriented tools and algorithm-oriented frameworks. OpenQASM and Qiskit represent a more traditional approach where the developer maintains a close relationship with the underlying gate operations. While this provides granular control, it increases the complexity of managing auxiliary qubits and uncomputation, which are essential for maintaining the coherence of the quantum state.

In contrast, Qrisp and Q# strive for a higher level of abstraction. Q# achieves this by being a standalone language with its own compiler and runtime environment. Qrisp, however, offers a unique middle ground by providing high-level abstractions-such as automatic uncomputation and typed variables-while remaining entirely within the Python ecosystem. This allows developers to utilize existing Python libraries and data science tools without switching to a new language environment.

Furthermore, the introduction of JAX-based compilation in Qrisp (Jasp) addresses the classical performance bottlenecks often encountered in Python-based quantum frameworks. Within this ecosystem, Qiskit serves as a robust general-purpose library optimized for low-level circuit synthesis, whereas PennyLane is tailored specifically for differentiable programming and QML. Qrisp distinguishes itself by leveraging the formalisms of RCCS and CCCS to focus on formal reversibility, offering a mathematically grounded framework for scaling complex algorithms through automated resource management.

The choice of platform depends on the specific requirements of the project. Developers targeting NISQ devices might prefer the low-level control of Qiskit, while those focused on large-scale algorithmic development and formal verification may find the abstractions provided by Qrisp or Q# to be more efficient.

3.8 Comparison of Simulation Paradigms of Qrisp and AerSimulator

In the following chapters, particularly Chapter 7, both the Qrisp simulator and AerSimulator are utilized. To account for potential discrepancies in experimental results, this section provides a technical comparison of their underlying implementations. Quantum circuit simulators can differ substantially in design philosophy depending on whether their primary objective is to model physical hardware or to support high-level algorithmic development. This section contrasts two representative implementations: the statevector backend of Qiskit Aer [43], a production-grade C++ simulator designed to emulate quantum hardware, and the Qrisp built-in simulator [45], a Python-native backend designed to support the high-level abstractions of the Qrisp framework [44].

The analysis is grounded in the source files:

- `qiskit-aer.simulators.statevector.statevector_state.hpp`
- `qrisp.simulator.simulator.py`
- `qrisp.circuit.operation.py`
- `qrisp.circuit.standard_operations.py`

3.8.1 Noise-Bearing Circuits Modeling

Before exploring the technical details of simulator implementations, it's essential to understand noise-bearing circuits, as they form the foundation of NISQ era quantum simulators. In an ideal quantum circuit, qubits maintain their state perfectly until a gate is applied. In a noise-bearing circuit, the system is modeled as an open quantum system, where several types of errors occur.

Relaxation and dephasing are fundamental decoherence mechanisms. Relaxation occurs when the qubit loses energy and spontaneously flips from the $|1\rangle$ state to the $|0\rangle$ state, characterized by the time scale T_1 (Longitudinal Relaxation Time), representing the time scale over which a qubit decays due to energy loss to the environment. Dephasing, characterized by T_2 (Transverse Dephasing Time), destroys the relative phase between $|0\rangle$ and $|1\rangle$, eliminating quantum superposition and representing the time scale over which phase information is lost. These two parameters are related by the constraint $T_2 \leq 2T_1$ and together define the coherence time, which limits the maximum depth of a quantum circuit before information is lost to noise.

Beyond relaxation and dephasing, gate errors introduce inaccuracies during physical operations. Physical operations, such as CNOT gates, are never 100% accurate, often introducing small unintended rotations that accumulate across a quantum circuit. Finally, readout errors occur during the measurement process, where a qubit in state $|0\rangle$ may be reported as $|1\rangle$, or vice versa, directly affecting the reliability of the final results obtained from a quantum computation.

To simulate or study noise-bearing circuits, researchers utilize specific mathematical frameworks that extend beyond simple unitary transformations. Instead of representing a system with a pure state-vector $|\psi\rangle$, these circuits employ density matrices ρ to account for mixed states, where the system exists in a statistical ensemble of potential quantum states. The evolution of these states under environmental interference is typically modeled using

Kraus operators. These operators describe how a state changes when subjected to specific noise phenomena is represented by applying a Pauli-X gate with a probability p .

These operations are often categorized into standardized error channels. Frequent models include the Depolarizing Channel, which replaces a qubit’s state with a completely random one, and the Amplitude Damping Channel, which models the physical loss of energy from a system to its environment.

3.8.2 Managing Shot Count in NISQ Devices

The selection of shot count, number of circuit executions or samples, is crucial for NISQ devices, where intrinsic errors degrade result fidelity. To determine the required number of shots, we employ the binomial proportion confidence interval formula $N = \frac{9p(1-p)}{\epsilon^2}$, derived from the three-sigma rule, where p represents the expected probability of the correct result and ϵ is the acceptable error margin. This allows us to estimate how many samples are necessary to identify correct results above a confidence threshold, enabling result filtering when measured error rates remain below the tolerance limit. However, increasing the number of shots does not remove hardware-induced errors such as decoherence, gate errors, or readout errors; rather, it provides better statistical estimates of the underlying error-affected probability distribution. With sufficient shots, the law of large numbers ensures that the empirical distribution of results converges to the true probability distribution; therefore, when circuit errors fall within acceptable bounds, correct outcomes will dominate the measurement histogram, allowing selective post-processing to improve final accuracy.

3.8.3 Physical Qubits

A physical qubit is a single hardware element, such as a superconducting circuit or trapped ion, that is highly susceptible to environmental decoherence and operational gate errors. A logical qubit is an error-corrected unit composed of multiple physical qubits that utilizes redundancy and parity measurements to detect and suppress these errors. In the current NISQ era, researchers primarily utilize physical qubits because the hardware overhead required to maintain a single stable logical qubit can exceed hundreds or thousands of physical units.

3.8.4 Shot Sampling Strategy

One of the most consequential design differences concerns how the two simulators generate measurement outcomes across multiple shots.

AerSimulator operates by physically advancing the state-vector through the circuit for each shot individually. When a measurement is encountered, the method `apply_measure` in `qiskit-aer...statevector_state.hpp` draws a random integer from the current probability distribution using `RngEngine` and then projects the state accordingly via a diagonal collapse matrix. For noise-bearing circuits, each Kraus operator application also consumes a random sample from `RngEngine::rand`, meaning the stochastic path is woven into the gate-level execution [43]. The dominant cost of computing resources therefore scales as $\mathcal{O}(\text{simulation_complexity} \cdot \text{shots})$ for n qubits, since each shot requires a full traversal of the state-vector.

The Qrisp simulator follows a fundamentally different approach that can be characterized as distribution-first sampling. The `run` function in `simulator.py` executes the circuit exactly once to obtain the complete probability vector `cl_prob` over all measure-

ment outcomes [45]. Discrete samples are then generated in a single vectorised call to `numpy.random.choice`, after which the Numba-compiled¹ helper `gen_res_dict` aggregates the result histogram. Formally this reduces the cost to $\mathcal{O}(\text{simulation_complexity} + \text{shots})$ for ideal circuits, because the quantum simulation phase is entirely independent of the requested shot count. A special branch exists for the case `shots = None`, where the exact probability distribution is returned directly without any sampling.

The two strategies are therefore complementary in scope: `AerSimulator` faithfully replicates the stochastic nature of hardware execution including mid-circuit noise, whereas the `Qrisp` simulator is optimised for the common case of ideal, repeated measurements on a fixed circuit.

3.8.5 Gate Set and High-Level Primitives

`AerSimulator` defines its supported instruction set through the `StateOpSet` object declared in `qiskit-aer...statevector_state.hpp` [43]. This set comprises standard physical gates such as `u1`, `u2`, `u3`, `cx`, `rx`, `ry`, `rz`, and their multi-controlled variants, along with operations tied to hardware-level concerns including `kraus`, `roerror`, `multiplexer`, and the full suite of classical-register save operations. The gate set is deliberately closed: any instruction not present in `StateOpSet` raises an exception at runtime, enforcing a strict hardware-model contract.

The `Qrisp` gate set, defined in `qrisp.circuit.standard_operations.py`, largely overlaps with the physical gates above but introduces additional primitives that have no hardware counterpart [46]. Two of the gates are particularly significant.

The first is the `disentangle` operation. When the `Qrisp` uncomputation engine determines that a qubit has been returned to a separable product state, it inserts a `disentangle` instruction into the circuit. In `qrisp.simulator.simulator.py` the main simulation loop intercepts this instruction and invokes `QuantumState.disentangle` rather than treating it as a unitary gate [45]. This allows the simulator to factorise the internal tensor representation, reducing the effective dimension of the state and enabling parallel processing of the resulting product components. `AerSimulator` has no equivalent mechanism; it represents the full n -qubit statevector as a single flat array throughout execution.

The second is the pair `qb_alloc` and `qb_dealloc`, which implement dynamic qubit management. The preprocessor `count_measurements_and_treat_alloc` in `qrisp.simulator.simulator.py` resolves these instructions before the main loop, growing or shrinking the internal `QuantumState` object accordingly [45]. In contrast, `AerSimulator` follows the conventional fixed-width circuit model: memory for the full statevector is allocated once during `initialize_qreg`, as reflected in the `required_memory_mb` method of `statevector_state.hpp` [43]. Dynamic allocation is therefore not supported and must be resolved by the compiler before submission.

3.8.6 Measurement and State Branching

The two simulators also differ in how they represent intermediate measurement outcomes during circuit execution.

¹Numba is an open-source Just-In-Time (JIT) compiler that translates a subset of Python and NumPy code into fast machine instructions using the LLVM compiler toolchain. By decorating functions with `@njit`, Numba allows Python-based quantum simulators to achieve execution speeds comparable to C or C++ by compiling performance-critical loops at runtime.

In AerSimulator, a call to `apply_measure` immediately collapses the statevector to a single outcome, chosen by drawing from `RngEngine`. The method `measure_reset_update` then applies the corresponding diagonal projector and renormalises the vector in place [43]. The simulation therefore always follows a single stochastic path, and the cost of mid-circuit measurements is identical to that of final measurements.

The Qrisp simulator instead uses what the source code refers to as multiverse measurements. Before the main simulation loop, `insert_multiverse_measurements` transforms measurement instructions into a form that allows the `QuantumState` object to branch into multiple parallel outcome states without collapsing any of them [45]. The combined probability vector is collected by `iqs.multi_measure` after all gates have been applied.

The `single_shot_sim` function provides a secondary execution path that does perform shot-by-shot collapse, using `random.random()` to select between the two post-measurement states returned by `QuantumState.last_mes_outcome` [45]. This function is used when the simulator must faithfully track a single trajectory, for example in interactive or incremental simulation sessions.

3.8.7 Performance Architecture

AerSimulator is implemented in C++ and relies on the `QubitVector` class for all linear-algebraic operations on the state-vector. Parallelism is achieved through OpenMP, with the threshold controlled by `omp_qubit_threshold` [43]. Optional CUDA based GPU acceleration is available via the `AER_THRUST_SUPPORTED` compilation flag, which substitutes the default `QubitVector` with a Thrust-backed variant. The RNG state is encapsulated in `RngEngine`, enabling deterministic reproducibility.

The Qrisp simulator is written in Python but mitigates the associated overhead through two complementary mechanisms [45]. First, the function `gen_res_dict`, which histograms the sampled outcomes, is compiled to native machine code at import time using Numba’s `@jit(cache=True)` decorator. Second, the bulk of the linear algebra, the application of unitary matrices to the state tensor, is delegated to NumPy’s C-implemented backend. The result is that the performance-critical inner loop operates at a level comparable to compiled code, while the orchestration layer remains in Python. Additionally, `single_shot_sim` spawns a background thread to pre-calculate the unitary of the next instruction while the current one is being applied, reducing latency through instruction-level pipelining.

The practical consequence is that the performance gap between the two implementations is narrower than the language boundary might suggest, particularly for circuits dominated by large matrix applications rather than control-flow overhead.

3.8.8 Zero Handling and Numerical Stability

A minor but technically relevant difference concerns the treatment of near-zero amplitudes. AerSimulator applies a fixed global threshold, `json_chop_threshold`, when serializing state-vectors for output, trimming amplitudes below this value to exactly zero. The Qrisp simulator takes a dynamic approach: after computing the final probability vector, it applies `np.round(c1_prob, int(-np.log10(np.max(c1_prob))) + 5)`, scaling the number of retained decimal places relative to the maximum observed probability [45]. The vector is then normalized before sampling.

3.8.9 Summary

Table 3.2 summarizes the principal all discussed differences.

AerSimulator is engineered to simulate quantum hardware, exposing a gate set and execution model that closely mirrors physical device constraints, including noise, reset, and gate-level transpilation.

The Qrisp simulator, by contrast, is designed to support the simulation of quantum algorithms, providing native abstractions for automatic uncomputation, dynamic qubit allocation, and exact probability extraction that would be difficult or inefficient to express at the hardware gate level.

For a more comprehensive exploration of simulation paradigms, we refer the reader to the extensive review by Cicero et al. [9]. Their work provides a detailed taxonomy of Schrödinger and Feynman-style simulators while offering critical guidance on hardware acceleration. Specifically, the authors analyze how different architectural backends impact performance: from the high-throughput parallelization capabilities of GPUs and the low-latency, customizable logic of FPGAs, to the versatile memory management of multi-core CPUs. This review provides the necessary context for understanding how the differing underlying architectures of simulators like Qrisp and Aer influence the scalability and execution fidelity of quantum software.

Table 3.2: Feature comparison of AerSimulator and Qrisp Simulator.

Feature	AerSimulator (<code>statevector_state.hpp</code>) [43]	Qrisp Simulator (<code>simulator.py</code>) [45]
Implementation language	C++ with OpenMP and optional Thrust backend	Python with NumPy and Numba JIT compilation
Shot sampling strategy	Per-shot stochastic collapse via <code>RngEngine</code>	Single-pass distribution computation followed by <code>numpy.random.choice</code>
Measurement branching	Immediate collapse to a single path; state is normalized in place	Multiverse branching deferred to <code>multi_measure</code> ; exact probabilities preserved
Mid-circuit reset	Collapse and renormalization via <code>measure_reset_update</code> ; conditional X rection applied if needed	Random path selection over <code>last_mes_outcome</code> in <code>single_shot_sim</code>
Qubit management	Static allocation at circuit initialization via <code>initialize_qreg</code>	Dynamic via <code>qb_alloc</code> and <code>qb_dealloc</code> resolved in preprocessing
Disentanglement	Not supported; full n -qubit statevector maintained throughout	Native <code>disentangle</code> operation factorizes separable qubits
Noise support	Kraus operators, readout error, multiplexer via <code>StateOpSet</code>	Ideal simulation only; no built-in noise channel support
Zero-amplitude handling	Fixed global threshold <code>json_chop_threshold</code>	Adaptive rounding relative to <code>np.max(c1_prob)</code>

Chapter 4

Identifying Use Cases for Hybrid Quantum Computing

Gate-based quantum computers are becoming more accessible through cloud platforms and academic infrastructure providers, which provides an opportunity to evaluate their suitability for accelerating classical computational problems. This accessibility became a deciding factor in selecting a combinatorial problem with an algorithm designed for gate-based quantum computers. Another decisive factor was the applicability of Grover’s algorithm, which is described in [35], a reference used to study the fundamentals of quantum computing and quantum algorithms, as one of the most practically relevant algorithms for near-term gate-based quantum hardware. This chapter documents the process of identifying and implementing a practical use case for hybrid quantum-classical computation.

During the literature review, several papers describing quantum algorithms and circuits were examined. However, not all of them were easily reproducible, as they often require domain-specific knowledge that is not always explicitly stated. The exact set cover problem algorithm described in [34] was selected for implementation due to its clear description and potential for practical application.

When considering real-world problem mappings, scheduling problems appeared as a natural candidate. It should be noted that the set cover problem with overlaps would be more suitable for practical scheduling scenarios than the exact set cover problem defined in Section 4.1. However, analysis revealed that modifying the algorithm to handle overlapping sets would require additional qubits beyond what is currently available on real hardware. Therefore, the implementation was constrained to the exact cover formulation. As a practical example, a physiotherapy clinic scheduling problem (see Section 4.2) was chosen, where a set of patients must be assigned to physiotherapists with specific qualifications. This problem maps naturally to the exact set cover formulation and serves as a representative case for evaluating the quantum algorithm implementation.

Beyond the primary scheduling use case, the experimental evaluation also incorporated other fundamental NP-complete problems to test the versatility of the quantum approach. Specifically, the Boolean Satisfiability Problem (SAT), which seeks a variable assignment that satisfies a given logic formula, and the Knapsack Problem, an optimization task focused on maximizing value within a weight-constrained capacity, were utilized. These problems provide diverse benchmarks for assessing how different logical structures influence quantum execution. A comprehensive discussion, including the specific implementation details

and performance evaluation of these models within the Qrisp framework, is provided in Chapter 7.

As a practical example, a physiotherapy clinic scheduling problem (see Section 4.2) was chosen, where a set of patients must be assigned to physiotherapists with specific qualifications. This problem maps naturally to the exact set cover formulation and serves as a representative case for evaluating the quantum algorithm implementation.

4.1 Exact Cover Problem Definition

Let $\mathcal{U} = \{1, 2, \dots, m\}$ be a finite set of elements (the *universe*) and let $\mathcal{S} = \{S_1, S_2, \dots, S_n\}$ be a collection of subsets of $\mathcal{U}$, where $S_j \subseteq \mathcal{U}$ for all $j \in \{1, \dots, n\}$.

The *Exact Cover Problem* (ECP) asks to find a subcollection $\mathcal{S}^* \subseteq \mathcal{S}$ such that every element of $\mathcal{U}$ is contained in *exactly one* subset of $\mathcal{S}^*$, i.e., the selected subsets form a partition of $\mathcal{U}$ [34].

Introducing binary decision variables $x_j \in \{0, 1\}$ for each $j \in \{1, \dots, n\}$, where $x_j = 1$ if subset S_j is selected and $x_j = 0$ otherwise, the feasibility problem can be stated as:

$$\text{find } x \in \{0, 1\}^n \quad (4.1)$$

$$\text{subject to } \sum_{j:i \in S_j} x_j = 1 \quad \forall i \in \mathcal{U} \quad (4.2)$$

Constraint (4.2) ensures that each element $i \in \mathcal{U}$ is covered by exactly one selected subset.

When each subset S_j is associated with a non-negative cost $c_j \geq 0$, the problem becomes an optimization variant seeking a minimum-cost exact cover [6]:

$$\min \sum_{j=1}^n c_j x_j \quad (4.3)$$

$$\text{subject to } \sum_{j:i \in S_j} x_j = 1 \quad \forall i \in \mathcal{U} \quad (4.4)$$

$$x_j \in \{0, 1\} \quad \forall j \in \{1, \dots, n\} \quad (4.5)$$

The ECP is closely related to the *Set Covering Problem* (SCP), which relaxes constraint (4.4) to $\sum_{j:i \in S_j} x_j \geq 1$, allowing elements to be covered multiple times. ECP is NP-complete, being a special case of the Set Partitioning Problem.

4.2 Selected Use Case Problem Description

We consider a physiotherapy clinic scheduling problem where a set of patients must be assigned to physiotherapists such that:

- Each patient is treated by *exactly one* physiotherapist (exact cover constraint),
- The assigned physiotherapist is qualified for the patient's required procedure,
- The physiotherapist is available during the patient's preferred time slots,
- The total assignment cost is minimized.

4.2.1 Sets and Indices

$\mathcal{T} = \{1, \dots, n\}$	Set of physiotherapists
$\mathcal{P} = \{1, \dots, m\}$	Set of patients
$\mathcal{R} = \{\text{post_op}, \text{sports}, \text{manual}, \text{elderly}, \text{neuro}, \dots\}$	Set of procedures
$\mathcal{S} = \{0, 1, \dots, S_{\max}\}$	Set of time slots

4.2.2 Parameters

For each patient $i \in \mathcal{P}$:

$r_i \in \mathcal{R}$	Required procedure for patient i
$W_i \subseteq \mathcal{S}$	Preferred time slots for patient i

For each physiotherapist $j \in \mathcal{T}$:

$c_j \in \mathbb{R}_{>0}$	Cost of assigning physiotherapist j
$Q_j \subseteq \mathcal{R}$	Qualified procedures for physiotherapist j
$A_j \subseteq \mathcal{S}$	Available time slots for physiotherapist j

4.2.3 Compatibility and Coverage

A physiotherapist j is *compatible* with patient i if and only if:

$$\text{compatible}(i, j) \iff (r_i \in Q_j) \wedge (W_i \cap A_j \neq \emptyset) \quad (4.6)$$

The *coverage set* of physiotherapist j is the set of all patients they can treat:

$$C_j = \{i \in \mathcal{P} : \text{compatible}(i, j)\} \quad (4.7)$$

4.2.4 Decision Variables

$$x_j = \begin{cases} 1 & \text{if physiotherapist } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \quad \forall j \in \mathcal{T} \quad (4.8)$$

4.2.5 Concrete Exact Cover Formulation

Objective Function

Minimize the total cost of selected physiotherapists:

$$\min_{x \in \{0,1\}^n} \sum_{j \in \mathcal{T}} c_j x_j \quad (4.9)$$

Constraints

Each patient must be covered by *exactly one* selected physiotherapist:

$$\sum_{j \in \mathcal{T} : i \in C_j} x_j = 1 \quad \forall i \in \mathcal{P} \quad (4.10)$$

Equivalently, defining the coverage indicator $a_{ij} = \mathbf{1}[i \in C_j]$:

$$\sum_{j=1}^n a_{ij} x_j = 1 \quad \forall i \in \mathcal{P} \quad (4.11)$$

Complete Formulation

$$\begin{aligned}
& \min \quad \sum_{j \in \mathcal{T}} c_j x_j \\
& \text{subject to} \quad \sum_{j: i \in C_j} x_j = 1 & \forall i \in \mathcal{P} \\
& \quad x_j \in \{0, 1\} & \forall j \in \mathcal{T} \\
& \quad C_j = \{i \in \mathcal{P} : r_i \in Q_j \wedge W_i \cap A_j \neq \emptyset\}
\end{aligned} \tag{4.12}$$

4.2.6 Relation to Classical Exact Cover

The formulation maps directly to the *Exact Cover Problem* (ECP):

ECP Concept	Scheduling Interpretation
Universe $\mathcal{U}$	Set of patients $\mathcal{P}$
Subsets $\{S_j\}_{j=1}^n$	Coverage sets $\{C_j\}_{j \in \mathcal{T}}$
Exact cover	Each patient assigned to exactly one physio
Subset cost c_j	Physiotherapist assignment cost

4.2.7 Feasible Problem Size Reasoning

Since most accessible quantum gate-based machines range from 24 qubits (IQM Star24 at IT4Innovations) to 156 qubits (IBM Heron r2 on cloud), only limited problem sizes are directly solvable. Based on the circuit design described in Chapter 5, the required qubit count is:

$$Q = |\mathcal{T}| + |\mathcal{P}| \cdot \lceil \log_2 |\mathcal{T}| \rceil + 1 \tag{4.13}$$

where n is the number of subsets (physiotherapists) and m is the number of universe elements (patients), following the ECP formulation in Section 4.1.

Table 4.1 shows the maximum problem sizes for available quantum hardware.

Table 4.1: Maximum ECP problem sizes for available quantum hardware.

QPU	Qubits	$ \mathcal{T} $	$ \mathcal{P} $	$b = \lceil \log_2 \mathcal{T} \rceil$	Q
IQM Star24	24	4	9	2	23
IBM Heron r2	156	12	38	4	153

In realistic physiotherapy scheduling, the problem size depends on the number of physiotherapists $|\mathcal{T}|$, patients $|\mathcal{P}|$, procedures $|\mathcal{R}|$, and time slots $|\mathcal{S}|$. A typical small clinic scenario with 10 physiotherapists, 20 patients, 5 procedures, and 16 time slots would require:

$$Q = 10 + 20 \cdot \lceil \log_2 10 \rceil + 1 = 10 + 20 \cdot 4 + 1 = 91 \text{ qubits} \tag{4.14}$$

which exceeds IQM Star24 capacity but fits within IBM Heron r2.

For larger instances common in hospital settings (e.g., 50 physiotherapists, 200 patients), the qubit requirement grows to $Q = 50 + 200 \cdot 6 + 1 = 1251$ qubits, far exceeding current hardware capabilities.

Therefore, a hybrid classical-quantum approach is essential. The implemented Branch-and-Bound [49] algorithm reduces the problem size through constraint propagation and

branching decisions, dispatching only sufficiently small subproblems to the quantum solver. Branch-and-Bound frameworks integrated with quantum computing have been successfully applied to various combinatorial optimization problems, including integer linear programming, as demonstrated by Sanavio, Tignone, and Ercolessi [49] using D-Wave quantum annealers. In contrast, our approach employs a Grover’s gate-based quantum algorithm with problem directly encoded to oracle rather than quantum annealing. Implementation details are provided in Chapter 5, while the trade-off between potential quantum speedup and classical-quantum coordination overhead is analyzed in Chapter 8.

4.2.8 LP Relaxation and Dual Bound

Since only feasible branches are targeted to reduce problem size as mentioned in Section 4.2.7, then Linear Programming relaxation and dual bound can be used in order to determine near-optimal solutions. The LP relaxation replaces $x_j \in \{0, 1\}$ with $x_j \in [0, 1]$. The dual problem is:

$$\begin{aligned} \max \quad & \sum_{i \in \mathcal{P}} u_i \\ \text{subject to} \quad & \sum_{i \in C_j} u_i \leq c_j \quad \forall j \in \mathcal{T} \\ & u_i \in \mathbb{R} \quad \forall i \in \mathcal{P} \end{aligned} \tag{4.15}$$

The dual variables u_i represent the value of covering patient i . For any feasible dual solution, $\sum_i u_i$ provides a valid lower bound on the optimal cost. This approach follows the methodology presented in [6], building upon the original LP relaxation and dual bound formulation introduced in [36].

Reduced Cost

The reduced cost of physiotherapist j is:

$$\bar{c}_j = c_j - \sum_{i \in C_j} u_i \tag{4.16}$$

Branching on variables with $|\bar{c}_j| \approx 0$ resolves maximum ambiguity in the LP relaxation (Balas–Carrera strategy).

4.2.9 Complexity

- The Exact Cover Problem is **NP-complete** [36].
- Grover’s algorithm provides quadratic speedup: $O(\sqrt{2^{|\mathcal{T}|}/k})$ oracle queries for k solutions (see Section 5.6.2).
- Hybrid approach: classical Branch-and-Bound with LP relaxation prunes the search space before quantum dispatch.

4.3 Example Instance of Selected ECP

$$\mathcal{P} = \{P1, P2, P3\}$$

$$\mathcal{T} = \{A, B, C, D, E\}$$

Physio	Qualifications Q_j	Slots A_j	Cost c_j
A	{elderly}	{0,1,2,3}	1.0
B	{post_op, elderly}	{0,1,2,3}	1.0
C	{post_op}	{0,1,2,3}	1.0
D	{sports}	{0,1,2,3}	1.0
E	{post_op, sports}	{0,1,2,3}	1.0

Patient	Procedure r_i	Slots W_i
P1	post_op	{0,1,2,3}
P2	sports	{0,1,2,3}
P3	elderly	{0,1,2,3}

Coverage sets:

$$C_A = \{P3\}$$

$$C_B = \{P1, P3\}$$

$$C_C = \{P1\}$$

$$C_D = \{P2\}$$

$$C_E = \{P1, P2\}$$

Valid exact covers (each patient covered exactly once):

- $\{B, D\}$: covers P1, P3 via B; P2 via D
- $\{A, C, D\}$: covers P3 via A; P1 via C; P2 via D
- $\{A, E\}$: covers P3 via A; P1, P2 via E

Chapter 5

Hybrid Quantum Computing ECP Implementation

This chapter describes the implementation of a hybrid classical–quantum solver for the physiotherapy clinic scheduling problem formulated as an Exact Cover Problem (ECP) in Chapter 4. The implementation consists of two main components: a quantum circuit for the ECP based on Grover’s algorithm, and a classical Branch-and-Bound algorithm that dispatches sufficiently small subproblems to the quantum solver. An overview of the approach is presented in Figure 5.1.

5.1 Implementation Architecture Overview

The hybrid solver architecture follows a divide-and-conquer strategy where classical computation handles problem decomposition and pruning, while quantum computation addresses the combinatorial search within small subproblems. This approach is motivated by the qubit constraints of current quantum hardware, as analyzed in Section 4.2.7. The implementation is structured into three main modules:

- Problem mapping module: Transforms the physiotherapy scheduling problem into the ECP formulation defined in Section 4.1.
- Quantum ECP solver: Implements Grover’s algorithm with a controlled counter oracle following [34].
- Hybrid Branch-and-Bound: Classical search algorithm inspired by the methodologies presented in [6] and building upon the foundational LP relaxation techniques from [36]. The algorithm prunes the search space using dual bounds and dispatches sufficiently small subproblems to the quantum solver.

5.2 Problem Mapping: From Scheduling to Exact Cover

The physiotherapy clinic scheduling problem, as defined in Section 4.2, must be transformed into the ECP format suitable for the quantum solver.

Require: Problem instance P , quantum backend Q , threshold θ

Ensure: Set of exact cover solutions S

```

1: Classical Branch-and-Bound:
2: Initialize search state with all elements uncovered
3: while search tree not exhausted do
4:   Apply constraint propagation (forced inclusions, infeasibility detection)
5:   Compute lower bound via LP dual relaxation
6:   if bound  $\geq$  best known cost then
7:     Prune branch
8:   else if subproblem size  $\leq \theta$  then
9:     Dispatch to QUANTUMSEARCH
10:  else
11:    Select branching variable by minimum reduced cost
12:    Branch: include or exclude selected subset
13:  end if
14: end while

15: Quantum Search Subroutine:
16: Encode remaining subproblem as quantum oracle
17:  $n \leftarrow$  number of remaining subsets
18:  $r \leftarrow \lfloor \frac{\pi}{4} \sqrt{2^n} \rfloor$ 
19: Prepare uniform superposition over  $n$  qubits
20: for  $i = 1$  to  $r$  do
21:   Apply oracle (mark valid exact covers)
22:   Apply diffusion operator (amplify marked states)
23: end for
24: Measure and verify candidates against original constraints
25: Add verified solutions to  $S$ 

```

Figure 5.1: Hybrid branch-and-bound algorithm with Grover-based quantum search for the exact cover problem. Classical preprocessing reduces the search space through constraint propagation and dual bound pruning. Subproblems below size threshold θ are solved using Grover’s algorithm with $O(\sqrt{2^{|\mathcal{T}|}})$ oracle queries.

5.2.1 Coverage Set Construction

For each physiotherapist $j \in \mathcal{T}$, the coverage set C_j is computed by evaluating the compatibility function:

$$C_j = \{i \in \mathcal{P} : r_i \in Q_j \wedge W_i \cap A_j \neq \emptyset\} \quad (5.1)$$

where r_i is the required procedure for patient i , Q_j is the set of qualified procedures for physiotherapist j , W_i represents the patient's preferred time slots, and A_j represents the physiotherapist's available time slots.

5.2.2 Slot Intersection Logic

The implementation computes slot compatibility through set intersection. A physiotherapist can serve a patient if and only if:

1. The physiotherapist holds the required qualification for the patient's procedure.
2. There exists at least one common time slot between the patient's preferences and the physiotherapist's availability.

The slot sets are represented as Python sets of integers, where each integer corresponds to a discrete time slot index. The intersection $W_i \cap A_j$ is computed using native set operations, and the result is checked for non-emptiness to determine scheduling feasibility.

5.2.3 Universe and Subset Mapping

The ECP mapping follows the correspondence established in Section 4.2:

- The universe $\mathcal{U}$ corresponds to the patient set $\mathcal{P}$, with each patient assigned an integer index.
- Each subset S_j corresponds to a physiotherapist's coverage set C_j , containing the indices of patients that physiotherapist can serve.
- The subset cost c_j corresponds to the physiotherapist assignment cost.

The transformation is implemented in the `to_ecp_format()` method, which produces:

```
universe = [0, 1, ..., m] # Patient indices
subsets = {physio_id: {patient_indices}} # Coverage mapping
```

5.3 Quantum Exact Cover Solver

The quantum solver implements Grover's algorithm for the Exact Cover Problem following the circuit design proposed by Jiang and Wang [34]. The key contribution of their approach is a novel controlled counter mechanism that efficiently checks whether each universe element is covered exactly once.

Grover's algorithm operates on the principle of amplitude amplification. Given an unstructured search space of $N = 2^{|\mathcal{T}|}$ possible solutions, a classical computer requires $\mathcal{O}(N)$ evaluations to find a target solution with high probability. Grover's algorithm achieves quadratic speedup by executing only $\mathcal{O}(\sqrt{N})$ quantum oracle calls. The algorithm begins by preparing an equal superposition of all computational basis states using Hadamard gates.

In each iteration, an oracle marks the solution state by introducing a phase flip (π -rotation), followed by the diffusion operator that inverts all amplitudes about the mean. This two-step process, oracle marking and inversion about the average, iteratively increases the amplitude of solution states while decreasing amplitudes of non-solutions. After approximately $\frac{\pi}{4}\sqrt{N}$ iterations, measurement collapses the quantum state onto a solution with high probability. For the Exact Cover Problem, the oracle evaluates whether a given subset configuration covers each element exactly once, making Grover's algorithm suitable for this combinatorial search task.

5.3.1 Circuit Architecture

The quantum circuit employs three types of qubits:

Subset Selection Qubits (S) For $|\mathcal{T}|$ subsets (physiotherapists), the circuit allocates $|\mathcal{T}|$ qubits in register S . Each qubit S_j represents the binary decision variable x_j from the ECP formulation (4.1)–(4.2):

$$x_j = 1 \iff S_j = |1\rangle \quad (5.2)$$

Counter Qubits (U) For $|\mathcal{P}|$ universe elements (patients), the circuit allocates $|\mathcal{P}| \cdot b$ counter qubits, where $b = \lceil \log_2 |\mathcal{T}| \rceil$ is the number of bits required to represent the maximum possible coverage count (up to $|\mathcal{T}|$ physiotherapists covering a single patient). Each patient $i \in \mathcal{P}$ has a b -bit counter that tracks how many selected physiotherapists cover them.

Ancilla Qubit A single ancilla qubit, initialized in the state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, enables phase kickback when the oracle condition is satisfied.

The total qubit count is:

$$Q = |\mathcal{T}| + |\mathcal{P}| \cdot \lceil \log_2 (|\mathcal{T}| + 1) \rceil + 1 \quad (5.3)$$

which matches the formula used for feasibility analysis in Section 4.2.7.

5.3.2 Controlled Counter Implementation

The controlled counter is the central innovation from [34]. It implements a conditional increment operation: when a control qubit S_j (representing a selected physiotherapist) is in state $|1\rangle$, the counters for all patients $i \in C_j$ increment by one.

The increment operation uses a binary ripple-carry adder pattern. To ensure carry bits are calculated correctly without requiring auxiliary qubits, the gates must be applied in descending order of significance. For a b -bit counter with bits $(c_0, c_1, \dots, c_{b-1})$ where c_0 is the least significant bit and $b = \lceil \log_2 (|\mathcal{T}| + 1) \rceil$, the increment proceeds as follows with LSB flip and carry propagation. the LSB flip applies a CNOT gate with the control qubit targeting the LSB c_0 . This flips the LSB whenever the control is $|1\rangle$. The carry propagation applies for each higher bit c_i (where $i > 0$) a multi-controlled X gate (MCX gate) with controls on the control qubit and all lower bits $c_0, c_1, \dots, c_{i-1}$, targeting bit c_i . This implements the carry chain: bit c_i flips only when the control is $|1\rangle$ and all lower bits were $|1\rangle$ before the flip. The gates are applied in MSB-to-LSB order for increment and LSB-to-MSB order for decrement (uncomputation). The implementation logic is summarized in the following pseudocode:

```

for i in range(b - 1, -1, -1):
    if i == 0:
        qc.cx(control_bit, count_bits[0])
    else:
        controls = [control_bit] + list(count_bits[:i])
        qc.mcx(controls, count_bits[i])

```

5.3.3 Oracle Construction

The oracle marks quantum states corresponding to valid exact covers by checking whether all counters equal exactly one. The oracle consists of three phases:

Phase 1: Counter Increment For each physiotherapist $j \in \mathcal{T}$ and each patient $i \in C_j$, the controlled counter for patient i is incremented conditioned on selection qubit S_j . After this phase, if the selection of physiotherapists corresponds to a valid exact cover, every patient counter in the register U will contain the value 1 (binary representation: $c_0 = 1$ and $c_k = 0$ for $k > 0$).

Phase 2: Equality Check and Phase Flip To verify that all $|\mathcal{P}|$ counters equal exactly 1, a multi-controlled logic is applied:

1. Bit Inversion: For every patient $i \in \mathcal{P}$, apply X gates to all bits in the counter register except the least significant bit ($c_1, \dots, c_{b-1}$). This transforms the „exactly one“ target pattern $(1, 0, \dots, 0)$ into an all-ones pattern $(1, 1, \dots, 1)$.
2. Phase Kickback: Apply a multi-controlled X gate with *all* bits of *all* patient counters ($|\mathcal{P}| \cdot b$ qubits) as controls and the ancilla qubit as the target. Since the ancilla is in the state $|-\rangle$, this induces a phase flip of -1 only if every patient is covered by exactly one physiotherapist.
3. Restoration: Re-apply X gates to the non-LSB bits to restore the counter states.

Phase 3: Counter Uncomputation The counter increments from Phase 1 are reversed in LSB-to-MSB order to restore all counter qubits to $|0\rangle$. This uncomputation is essential to entangle the phase information only with the selection register S , preventing the unwanted information encoded in other counter qubits from interfering with the Grover diffusion operator.

5.3.4 Diffusion Operator

The diffusion operator D performs the core amplification mechanism of Grover’s algorithm by inverting the amplitude about the average. Mathematically, it is expressed as:

$$D = H^{\otimes h} (2|0\rangle\langle 0| - I) H^{\otimes h} \quad (5.4)$$

where $H^{\otimes h}$ represents h Hadamard gates applied in parallel to all subset selection qubits. The central component $(2|0\rangle\langle 0| - I)$ is a reflection operator in the computational basis. The projection operator $|0\rangle\langle 0|$ isolates the all-zeros state, while the identity I acts on the entire Hilbert space. By construction, this operator leaves the all-zero computational basis state $|0\rangle^{\otimes h}$ unchanged and applies a phase shift of π to all other computational basis states. The

surrounding Hadamard transformations conjugate this reflection, effectively implementing the inversion about the average operation in the original basis space. This sequence mapping to the Hadamard basis, reflecting about $|0\rangle^{\otimes h}$, and returning to the computational basis, geometrically rotates the quantum state vector toward the solution subspace with each application. The implementation uses Hadamard gates and a multi-controlled Z gate:

```
# 1. Transform back from the computational basis to the Hadamard basis
qc.h(s_qubits)

# 2. Flip all qubits (0 -> 1 and 1 -> 0)
# This prepares the state so that the all zeros state is now all ones
qc.x(s_qubits)

# 3. Perform a Multi-Controlled Z gate (MCZ)
# This flips the phase of the |11...1> state only.
# In Qiskit, we often use H-MCX-H to create/replace the MCZ:
qc.h(s_qubits[-1]) # Prep the last qubit for the flip
qc.mcx(list(s_qubits[:-1]), s_qubits[-1]) # Apply the multi-controlled flip
qc.h(s_qubits[-1]) # Return the last qubit to its basis

# 4. Reverse the 'X' flip to return the qubits to their original orientation
qc.x(s_qubits)

# 5. Transform back to the computational basis
# This completes the "inversion about the average"
qc.h(s_qubits)
```

Figure 5.2: The Grover Diffusion Operator implemented in Qiskit.

5.3.5 Grover Iteration Count

The optimal number of Grover iterations is determined by:

$$r = \left\lceil \frac{\pi}{4} \sqrt{\frac{2^{|\mathcal{T}|}}{k}} \right\rceil \quad (5.5)$$

where $N = 2^{|\mathcal{T}|}$ is the search space size and k is the (estimated) number of solutions. In the implementation, k is conservatively estimated as the number of universe elements $|\mathcal{P}|$, since at least $|\mathcal{P}|$ subsets would be needed in the worst case to cover all elements. The iteration count is directly connected with quantum speedup $O\left(\sqrt{\frac{2^{|\mathcal{T}|}}{k}}\right)$ as mentioned in Section 5.6.2.

5.4 Hybrid Branch-and-Bound Algorithm

The classical Branch-and-Bound component reduces the problem size through systematic search space exploration, enabling quantum acceleration on the resulting smaller subproblems.

5.4.1 Search State Representation

Each node in the search tree maintains a **SearchState** containing:

- **included_physios**: Set of physiotherapists already selected.
- **excluded_physios**: Set of physiotherapists excluded from consideration.
- **uncovered_patients**: Set of patients not yet assigned.
- **available_physios**: Set of physiotherapists still available for selection.
- **dual_vars**: Dual variables for LP relaxation bounding.
- **lower_bound**: Cumulative cost of included physiotherapists.

5.4.2 Constraint Propagation

Before branching, the algorithm applies constraint propagation to reduce the search space:

Forced Inclusion If a patient can be covered by exactly one available physiotherapist, that physiotherapist must be included. The algorithm iterates through uncovered patients, counts covering physiotherapists, and forces inclusion when the count equals one.

Infeasibility Detection If any patient cannot be covered by any available physiotherapist, the current branch is infeasible and is pruned.

Redundant Physiotherapist Removal Physiotherapists that cannot cover any remaining uncovered patient, or would create overlapping coverage with already-covered patients, are removed from consideration.

5.4.3 LP Dual Relaxation Bound

The implementation employs the LP dual relaxation approach described in Section 4.2. Let $\mathcal{P}_{uncov} \subseteq \mathcal{P}$ be the set of patients not yet covered by the current partial assignment, and $\mathcal{T}_{avail} \subseteq \mathcal{T}$ be the set of physiotherapists available for selection. The dual variables u_i are assigned to each patient $i \in \mathcal{P}_{uncov}$, subject to:

$$\sum_{i \in C_j \cap \mathcal{P}_{uncov}} u_i \leq c_j \quad \forall j \in \mathcal{T}_{avail} \quad (5.6)$$

where c_j represents the weight or cost associated with physiotherapist j . A greedy dual ascent procedure is used to estimate the maximum values for u_i :

1. Initialize $u_i = 0$ for all $i \in \mathcal{P}_{uncov}$.
2. For each uncovered patient i , compute the maximum feasible increase δ_i :

$$\delta_i = \min_{j: i \in C_j} \left(c_j - \sum_{k \in C_j \cap \mathcal{P}_{uncov}} u_k \right) \quad (5.7)$$

3. If $\delta_i > 0$, increment u_i by δ_i and update the remaining capacities of the associated physiotherapists.
4. Repeat until no further increases are possible without violating the constraints.

The sum of these dual variables provides a valid lower bound (LB) on the total cost required to complete the cover:

$$\text{LB} = \sum_{j \in \mathcal{T}_{\text{selected}}} c_j + \sum_{i \in \mathcal{P}_{\text{uncov}}} u_i \quad (5.8)$$

5.4.4 Branching Strategy

The implementation follows the Balas–Carrera branching strategy [6], selecting the physiotherapist $j \in \mathcal{T}_{\text{avail}}$ whose reduced cost $\bar{c}_j$ is closest to zero:

$$\bar{c}_j = c_j - \sum_{i \in C_j \cap \mathcal{P}_{\text{uncov}}} u_i \quad (5.9)$$

where:

- c_j is the cost associated with physiotherapist j .
- u_i is the dual value assigned to patient i during the dual ascent procedure.
- $\mathcal{P}_{\text{uncov}}$ is the set of patients not yet covered by the current partial solution.

A reduced cost $\bar{c}_j = 0$ indicates that the physiotherapist is a prime candidate for the optimal cover according to the current lower bound estimation. Selecting j with the minimum $\bar{c}_j$ effectively explores the most promising branches of the search tree first.

Among physiotherapists with similar reduced costs, the one covering more uncovered patients is preferred. This strategy resolves maximum ambiguity in the LP relaxation.

For each selected physiotherapist j , two child branches are created:

- Include branch: Physiotherapist j is added to the solution; patients in C_j are marked as covered.
- Exclude branch: Physiotherapist j is removed from consideration.

5.4.5 Quantum Dispatch Criteria

A subproblem is dispatched to the quantum solver when it satisfies the following criteria:

1. Number of available physiotherapists $\leq \text{MAX_QUANTUM_PHYSIOS}$ (default: 4)
2. Number of uncovered patients $\leq \text{MAX_QUANTUM_PATIENTS}$ (default: 20)
3. Total qubit count $\leq \text{MAX_QUANTUM_QUBITS}$ (default: 24)

The qubit count is computed using the formula from Section 4.2.7. These thresholds are conservative to ensure reliable simulation and can be adjusted for different quantum backends.

5.4.6 Quantum Subproblem Execution

When a subproblem meets the dispatch criteria:

1. The remaining problem is converted to ECP format with local patient indices.
2. The quantum solver is instantiated with the subproblem’s universe and subsets.
3. Grover iterations are computed based on the subproblem size.
4. The circuit is executed with a fixed number of repetitions, known as shots (see Subsection 3.8.1). A default of 2,048 shots was selected to balance statistical precision against the computational time complexity of the simulators. While higher shot counts are often utilized on physical hardware to mitigate stochastic noise, 2,048 remains a sufficient baseline for capturing the probability distribution in a simulated environment.
5. Measurement outcomes are filtered by probability threshold and validated¹.
6. Solutions are combined with already-included physiotherapists and recorded.

5.4.7 Classical Fallback

If the quantum solver fails (due to exceptions or hardware errors), the algorithm falls back to a classical backtracking solver for the subproblem. This fallback mechanism is essential for several practical reasons. Current quantum hardware exhibits inherent limitations that prevent reliable execution of Grover’s algorithm on non-trivial problem instances. Quantum decoherence causes exponential decay of quantum information, while gate errors and crosstalk corrupt quantum states during circuit execution. For small problem sizes, accumulated noise may prevent the quantum solver from reaching sufficient amplitude concentration in solution states, resulting in measurement outcomes indistinguishable from random noise. Additionally, quantum processors may be unavailable due to maintenance, scheduling constraints, or resource contention in shared cloud-based environments. Rather than returning no solution, the fallback mechanism ensures the hybrid algorithm maintains correctness and completeness: any quantum acceleration gained on feasible instances is preserved, while infeasible instances still receive valid solutions through deterministic classical computation. This approach aligns with the NISQ computing paradigm, where quantum advantage is limited to specific problem scales and noise characteristics, necessitating graceful degradation to classical methods when quantum execution cannot meet reliability thresholds.

¹The implementation includes a `validate_solution` method but omits it at runtime due to computational overhead, despite validation being theoretically necessary in the NISQ era. This reflects a pragmatic trade-off: quantum speedup requires accepting higher failure rates than classical solvers, treating validation as an optional safeguard rather than mandatory verification.

```

def _backtrack(physios, uncovered, selected, idx):
    if not uncovered:
        record_solution(selected)
        return
    if idx >= len(physios):
        return
    pid = physios[idx]
    covered = coverage[pid] & uncovered
    if covered:
        _backtrack(physios, uncovered - covered,
                    selected | {pid}, idx + 1)
    _backtrack(physios, uncovered, selected, idx + 1)

```

5.5 Implementation Details

5.5.1 Technology Stack

The implementation uses the following technologies:

- Qiskit [20]: IBM’s quantum computing SDK for circuit construction and simulation.
- Qiskit Aer: High-performance quantum circuit simulator.
- Qiskit IBM Runtime: Interface for executing circuits on IBM quantum hardware.
- Python standard library: Data structures and algorithms.

5.5.2 Execution Modes

The quantum solver supports three execution modes controlled by environment variables:

- Local simulation (`SIMULATOR=qiskit`): Uses Qiskit Aer for fast local simulation.
- Cloud simulation (`SIMULATOR=ibm`): Uses IBM’s cloud-based noise-aware simulator.
- Hardware execution (`RUN_ON_HW=True`): Executes on actual IBM quantum hardware.

5.5.3 Solution Validation

The implementation includes comprehensive solution validation:

1. Coverage check: Verify that all patients are covered.
2. Overlap check: Verify that no patient is covered by multiple physiotherapists.
3. Existence check: Verify that all selected physiotherapists exist in the problem.

5.6 Complexity Analysis

5.6.1 Classical Branch-and-Bound Complexity

The worst-case time complexity of the classical Branch-and-Bound is $O(2^{|\mathcal{T}|} \cdot |\mathcal{T}||\mathcal{P}|)$ where $|\mathcal{T}|$ is the number of physiotherapists and $|\mathcal{P}|$ is the number of patients. The exponential factor $2^{|\mathcal{T}|}$ arises from the binary branching on each physiotherapist (include or exclude). At each node, the algorithm performs:

- Constraint propagation: iterates through uncovered patients and available physiotherapists to detect forced inclusions and infeasibility, requiring $O(|\mathcal{T}||\mathcal{P}|)$ operations.
- Dual relaxation bound computation: iterates through patients and physiotherapists to compute slack values, requiring $O(|\mathcal{T}||\mathcal{P}|)$ operations.
- Branching variable selection: evaluates reduced costs for all available physiotherapists, requiring $O(|\mathcal{T}||\mathcal{P}|)$ operations.
- Feasibility check: verifies that remaining patients can be covered, requiring $O(|\mathcal{T}||\mathcal{P}|)$ operations.

Thus, each of the $O(2^{|\mathcal{T}|})$ nodes requires $O(|\mathcal{T}||\mathcal{P}|)$ work, yielding the total complexity of $O(2^{|\mathcal{T}|} \cdot |\mathcal{T}||\mathcal{P}|)$. However, constraint propagation and dual bounding significantly reduce the explored search space in practice.

5.6.2 Quantum Speedup

For subproblems dispatched to the quantum solver, Grover's algorithm provides a quadratic speedup over classical exhaustive search. For a specific subproblem involving a subset of physiotherapists $\mathcal{T}_{sub} \subseteq \mathcal{T}$ and an estimated k valid covers, the complexity comparisons are as follows:

- Classical exhaustive search: $O(2^{|\mathcal{T}_{sub}|})$ evaluations of the coverage constraints.
- Grover's algorithm: $O\left(\sqrt{\frac{2^{|\mathcal{T}_{sub}|}}{k}}\right)$ quantum oracle calls (equivalent to the iteration count r ; see Section 5.3.5).

Each oracle call incurs a circuit depth complexity proportional to the cost of the controlled counter implementation, which scales as $O(|\mathcal{T}_{sub}| \cdot |\mathcal{P}| \cdot \log|\mathcal{T}_{sub}|)$. While the quantum approach provides a significant reduction in the number of configurations evaluated, the overall speedup in the hybrid ECP solver depends on the frequency of quantum dispatch and the relative overhead of quantum state preparation and measurement.

5.6.3 Overall Hybrid Complexity

The hybrid approach achieves its efficiency through:

1. Reducing the problem through classical constraint propagation.
2. Pruning branches using LP dual bounds.
3. Accelerating the remaining combinatorial search with quantum speedup.

The effectiveness depends on the problem structure and the proportion of computation handled by the quantum solver.

5.7 Alignment with Initial Paper Design of ECP Solver

The quantum ECP solver implementation directly follows the circuit design proposed by Jiang and Wang [34]. Key correspondences include:

- Oracle structure: The three-phase oracle (increment counters, check equality, uncompute) matches the paper’s Figure 5 and Section III-C.
- Counter design: The controlled counter uses the same binary ripple-carry increment logic with multi-controlled X gates.
- Qubit allocation: The allocation of subset qubits, counter qubits, and ancilla follows the paper’s specification.
- Circuit depth²: The oracle achieves $O(|\mathcal{P}| \cdot \log|\mathcal{T}|)$ circuit depth through the counter carry chain, as claimed in the paper.

The implementation in this thesis extends the original work by:

- Integrating the quantum solver into a hybrid Branch-and-Bound framework.
- Adding support for weighted (cost-minimizing) exact cover problems.
- Providing multiple execution backends (simulation and hardware).

²In quantum informatics, circuit depth represents the critical path of the execution: the minimum number of time steps required to complete all gates. It measures the longest sequence of sequential operations, accounting for the fact that independent gates can be executed in parallel.

Chapter 6

Error Mitigation and Correction

As we navigate the Noisy Intermediate-Scale Quantum (NISQ) era, results obtained from quantum algorithms are full of errors and uncertainty. It is hard to achieve real speedup and get correct results from currently available HW. The Exact Cover Problem (ECP), while theoretically solvable via Grover’s algorithm with quadratic speedup, is highly sensitive to the decoherence and gate infidelity prevalent in current superconducting processors. In this chapter, I explore two distinct strategies to combat these errors: active Quantum Error Correction (QEC) using rotated planar codes and post-processing via Matrix-free Measurement Mitigation (M3). To provide an empirical foundation for these strategies, I conducted a series of experiments to characterize the noisiness of contemporary quantum hardware. This investigation involves a comparative study between the VLQ IQM machine, hosted at the IT4Innovations National Supercomputing Center, and the IBM Marrakesh system, with a Heron R2 chip. By analyzing the depth of the implemented Error Correction Procedures, I evaluate how the gate-level overhead required for correction logic influences the overall fidelity on these specific superconducting architectures, each characterized by different topological constraints and error profiles.

6.1 Quantum Error Correction and the Rotated Planar Code

Quantum error correction encompasses a variety of approaches, as the field remains a central focus of research and development. One of the suggested approaches by IQM Academy is based on surface code which is designed to detect both bit-flips and phase flips, X errors and Z errors, by utilizing stabilizers that anti-commute with mentioned error type. The surface code impose higher demand on qubit count and quantum circuit complexity, however may improve overall result correctness as stated in [2].

6.1.1 Lattice Geometry and Resource Efficiency

The rotated planar code represents an evolution of the standard surface code, optimized for lattice superconducting architectures where physical footprint is a constraint. While a standard planar code of distance d requires approximately $2d^2 - 1$ physical qubits (see Section 3.8.3), the rotated variant achieves the same fault-tolerant distance d using only:

The code distance d is defined as the smallest number of physical errors that can result in a logical error. More precisely, we characterize this using the weight $|P|$ of a Pauli

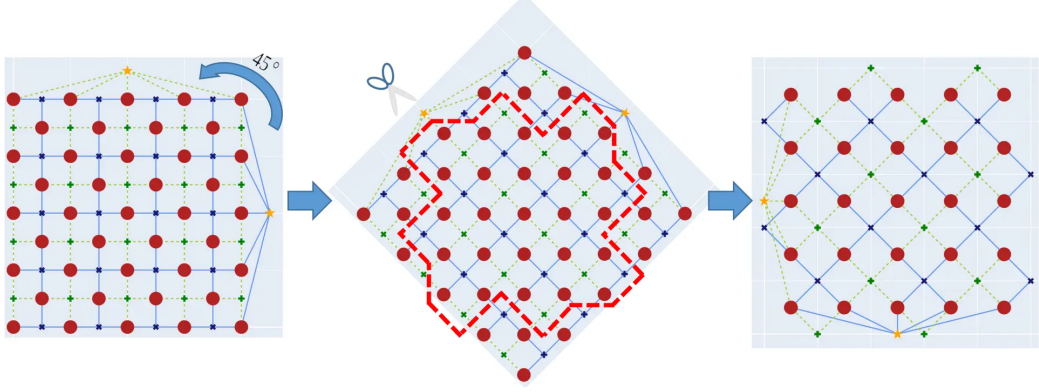


Figure 6.1: Planary Code to Rotated Planary Code Transformation. **Source:** IQM Academy

operator $P = P_1 \otimes \dots \otimes P_n$, which is the number of non-identity elements P_i ¹. As long as the number of physical errors satisfies the condition $d > 2|P|$, the decoding algorithm can successfully map the syndrome back to the original state, and a logical error cannot occur as described in [23].

This reduction is achieved by rotating the lattice 45° and identifying a square lattice with d data qubits on each side. For the ECP implementation, this efficiency is critical; it allows for higher-distance protection on current IBM hardware (like the 127-qubit *Eagle* processors) where every physical qubit (see Section 3.8.3) spared reduces the overall noise cross-section.

6.1.2 The Stochastic Nature of Decoding

In the stabilizer formalism, an error E is detected if it anti-commutes with a stabilizer g_i , resulting in a -1 eigenvalue (the syndrome). The decoding problem is to find a correction operator C that toggles the same stabilizer measurements as the error E . For each stabilizer generator g_i , this can be expressed by defining the syndrome bit as defined in equation 6.1 and the correction C is chosen such defined in equation 6.2.

$$g_i E = (-1)^{s_i(E)} E g_i \quad (6.1)$$

$$s_i(C) = s_i(E) \quad \forall i \quad (6.2)$$

This implies that the residual operator CE commutes with all stabilizer generators as defined in equation 6.3, so the corrected state is again in the stabilized subspace.

$$[g_i, CE] = 0 \quad \forall i \quad (6.3)$$

The correction is logically successful only if the residual operator CE belongs to the stabilizer group $\mathcal{S}$; if CE corresponds to a non-trivial logical operator, the syndrome is removed but a logical error remains.

To automate this selection, we model the probability of an error chain E . Given a set of physical qubits (see Section 3.8.3) with independent error probabilities p_i , the likelihood of a specific error configuration E (where $E_i = 1$ if an error occurred on qubit i) is:

$$P(E) = \prod_i (1 - p_i)^{1-E_i} p_i^{E_i} \quad (6.4)$$

¹For example, $\otimes I \otimes I = 0$, while $\otimes Z \otimes I = 2$

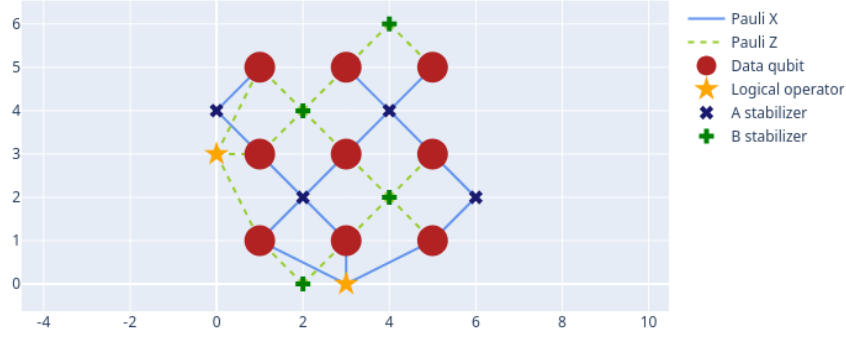


Figure 6.2: Rotated Planary Code Example. **Source:** IQM Academy

By taking the negative logarithm and ignoring constant terms, we define the edge weight w_i for the syndrome graph as:

$$w_i = -\ln \left(\frac{p_i}{1 - p_i} \right) \quad (6.5)$$

This mathematical transformation allows us to treat the maximum likelihood problem as a Minimum Weight Perfect Matching (MWPM) problem. The goal becomes minimizing the total weight $\sum w_i E_i$ of the correction chain. This section is based on the lecture notes by John Preskill [41] and the educational materials provided by IQM Academy [23].

6.1.3 The Complexity Paradox: Dijkstra and Blossom V

The implementation of MWPM is technically demanding, requiring the synthesis of two classical algorithms:

1. Dijkstra's Algorithm: Used to compute the shortest paths (distance matrix) between each pair of syndrome vertices.
2. Kolmogorov's Blossom V Algorithm: Used to find the optimal matching that minimizes the sum of these distances.

The classical complexity of this process scales as $\mathcal{O}(n^3 \ln n)$, where n is the number of qubits. Herein lies the complexity raise: we are utilizing Grover's algorithm to achieve a quadratic speedup for ECP, yet the the decoder requires a classical computation that scales cubically. If the time required to find the minimal graph via Dijkstra and Blossom exceeds the coherence time of the qubits, the system enters a state of logical breakdown, where new errors emerge faster than the classical controller can resolve them.

6.2 Measurement Mitigation: The M3 Implementation

While the rotated planar code addresses gate and decoherence errors, the final sampler result is often skewed by readout errors. In my Grover implementation for ECP, I utilized Matrix-free Measurement Mitigation (M3).

Technically, M3 approximates the inverse of the assignment matrix A in a reduced subspace. Rather than solving for the entire 2^n Hilbert space, M3 focuses on the bitstrings

$y_{measured}$ actually produced by the Sampler. Using an iterative linear solver (GMRES), M3 computes the corrected distribution:

$$\vec{x}_{mitigated} = A_{sub}^{-1} \vec{y}_{measured} \quad (6.6)$$

This approach was validated using GHZ-state benchmarks on the IBM hardware, providing a baseline of how measurement crosstalk propagates across the specific qubit layout used for the Grover oracle. The result is a sharpened probability distribution, supporting the ECP solution assignments (e.g., $|A, C, D\rangle$) to emerge.

6.3 Implementation: Applying M3 to Grover’s ECP

In the final phase of this investigation, I applied the M3 mitigation technique to the Qiskit implementation of the ECP Grover Solver. The workflow, as documented in the accompanying `analyze_circuit_on_IBM_HW_mthree.ipynb`, follows a three-step process:

1. Mapping and Benchmark: Before running the Grover circuit, the hardware (e.g., *IBM Marrakesh*) is benchmarked. M3 uses this benchmark to understand the readout fidelity of the specific physical qubits used in the layout.
2. Execution: The Grover circuit for the ECP instance is executed, producing a noisy count distribution where the correct assignment for the ECP is often obscured by readout crosstalk.
3. Subspace Mitigation: The `M3Mitigation.apply()` method is called on the raw counts. This corrects the distribution based on the GHZ-state benchmarks and localized bit-flip probabilities.

6.3.1 Observations

The quantitative results indicate that M3 effectively mitigated readout errors, reducing the distribution spread from approximately 3.9 percentage points (Table 6.1) to 1.2 percentage points (Table 6.2) around the uniform baseline of 6.25% (1/16). Because M3 is designed to invert only the final measurement assignment matrix, this narrowing is directly attributable to successful readout error mitigation, which visibly sharpened the probability peaks of the exact ECP solutions. However, while M3 targets the final sampler distribution, it cannot correct for gate errors and decoherence accumulated over the duration of the Grover iterations.

Consequently, the residual near-uniform distribution lies outside the scope of M3 mitigation. We hypothesize that the apparent rank improvement of the target state (B, C) to third place represents a post-mitigation statistical fluctuation near the noise floor rather than the recovery of a true Grover signal. This observation empirically confirms the depth limitations discussed in section 6.4. Since the circuit depth for ECP scales with N , readout mitigation must eventually be paired with active error correction to yield correct results. While this establishes the necessity of structural protections—such as the rotated planar code—for future fault-tolerant systems, the computational burden of real-time graph-matching decoders poses a significant challenge. In light of this complexity, the implementation of such error-correcting codes is omitted from this work, which focuses instead on the immediate practical benefits and limitations of M3 mitigation.

Table 6.1: Original results from IBM Marrakesh. The target solution (B, C) is ranked 12th. The corresponding circuit features a depth of 449 and was evaluated using 8,000 shots and one iteration.

Rank	States	Probability	Percentage
1	D	0.083500	8.35%
2	A, D	0.080250	8.03%
3	B, D	0.079000	7.90%
4	C, D	0.068750	6.88%
5	B	0.068500	6.85%
6	A, B, D	0.068250	6.83%
7	A	0.062375	6.24%
8	$\emptyset$ (Empty)	0.060625	6.06%
9	A, B, C, D	0.060500	6.05%
10	A, B	0.059375	5.94%
11	B, C, D	0.058000	5.80%
12	B, C	0.057250	5.73%
13	A, C, D	0.054875	5.49%
14	A, B, C	0.048500	4.85%
15	C	0.045875	4.59%
16	A, C	0.044375	4.44%

Table 6.2: Results from IBM Marrakesh after M3 Mitigation. The target solution (B, C) has ascended to 3rd place. The corresponding circuit features a depth of 449 and was evaluated using 8,000 shots and one iteration.

Rank	States	Probability	Percentage
1	B	0.068535	6.85%
2	B, C, D	0.068319	6.83%
3	B, C	0.067794	6.78%
4	C	0.067747	6.77%
5	A, D	0.064074	6.41%
6	A, B, C	0.063191	6.32%
7	D	0.063065	6.31%
8	C, D	0.062467	6.25%
9	$\emptyset$ (Empty)	0.062433	6.24%
10	A, C	0.061564	6.16%
11	B, D	0.059618	5.96%
12	A, C, D	0.059226	5.92%
13	A, B, D	0.058948	5.89%
14	A, B, C, D	0.058517	5.85%
15	A, B	0.058420	5.84%
16	A	0.056083	5.61%

6.4 Empirical Depth Benchmarking: IT4I VLQ Star24 vs. IBM Marrakesh

To evaluate the operational limits of the hardware used for our ECP solvers, I designed a depth-sensitive benchmark script. This script executes a series of increasing-depth CX gate-cycle circuits (see Figure 6.6) on both local simulators and real-world hardware. The primary objective is to determine the certainness $P(|000\rangle)$, the probability that a 3-qubit system returns to its initial state after a series of entangled operations-as a function of circuit depth.

6.4.1 Experimental Setup and Methodology

The benchmark utilizes a three-qubit circular entanglement pattern where CX gate gates are applied in cycles: $CX(q0, q1)$, $CX(q1, q2)$, and $CX(q2, q0)$. After every three repetitions, an ideal noiseless system returns to the $|000\rangle$ state. The experiment scales the number of repetitions exponentially-up to 8,000-to observe the decay of the quantum state across different architectures.

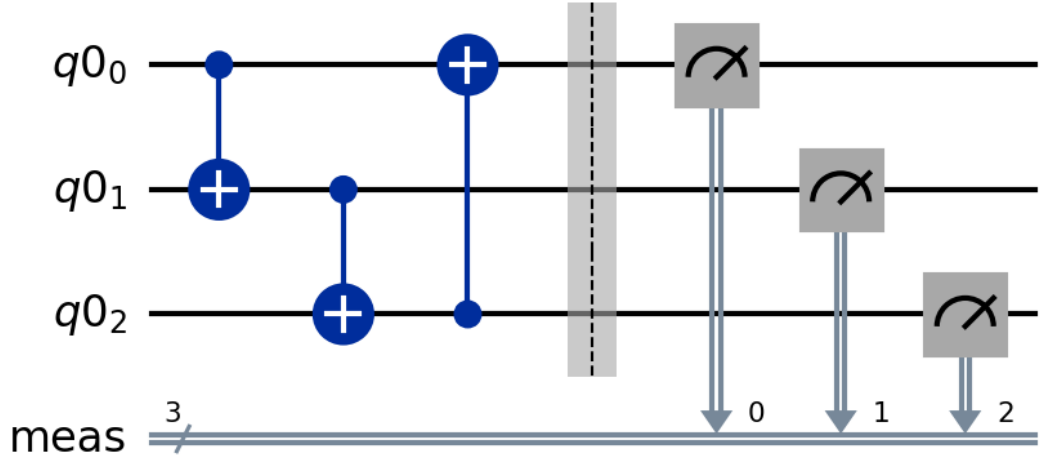


Figure 6.3: Comparison of a three-qubit circular entanglement circuit - Logical

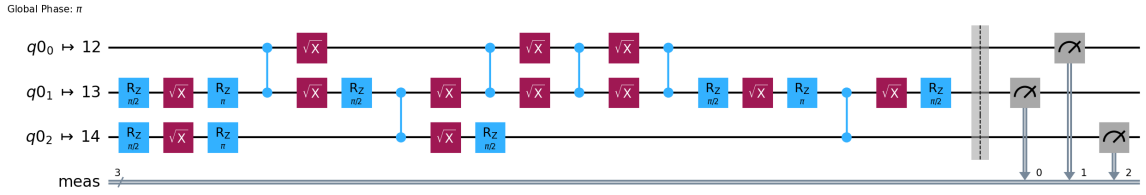


Figure 6.4: Comparison of a three-qubit circular entanglement circuit - IBM Marrakesh

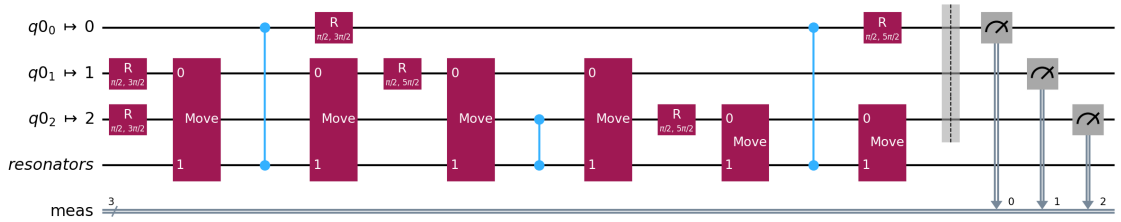


Figure 6.5: Comparison of a three-qubit circular entanglement circuit - IQM Star24 (VLQ)

Figure 6.6: Comparison of a three-qubit circular entanglement circuit across different back-ends. The transpiled versions for IBM Marrakesh and IQM Star24 (VLQ) reflect the routing overhead necessitated by their respective connectivity maps.

The hardware targets for the real-world execution include the *VLQ Star24* (an IQM-based 24-qubit machine at IT4Innovations) and the *IBM Marrakesh*. The implementation utilizes the `iqm.qiskit_iqm` and `qiskit_ibm_runtime` libraries to manage remote execution and transpilation with an optimization level of 2.

6.4.2 Comparative Analysis of Experiment Results

Figure 6.7 illustrates the results of the benchmark, comparing the survival probability of the $|000\rangle$ state against the transpiled circuit depth.

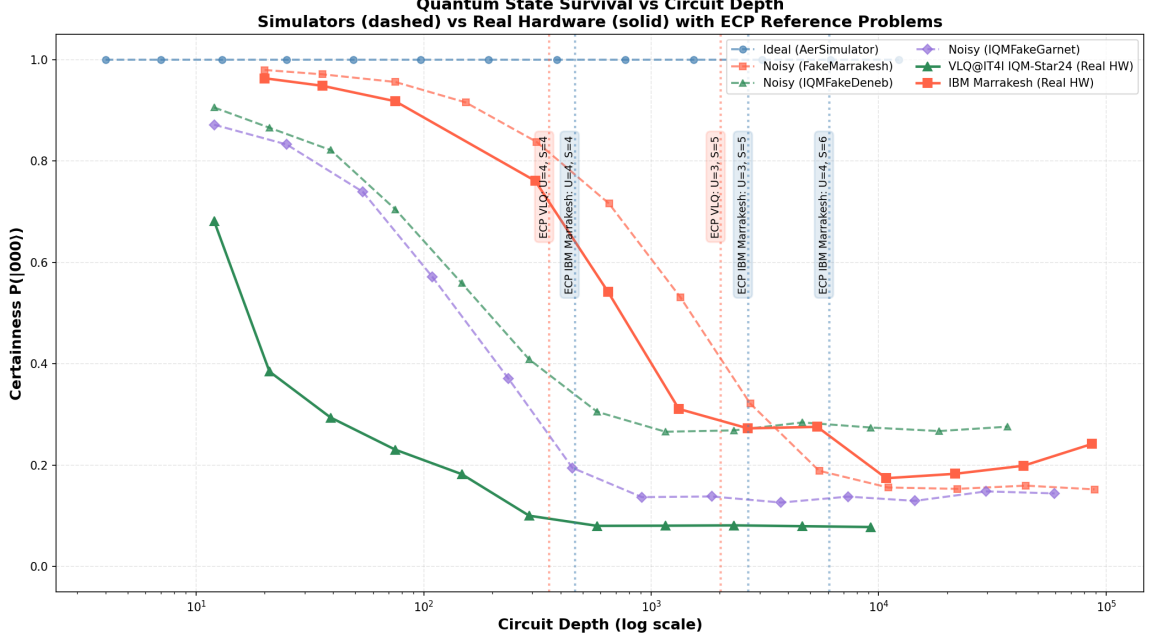


Figure 6.7: Quantum State Survival: Certainty vs. Circuit Depth across simulators and real hardware.

The data reveals several critical insights into the current state of NISQ hardware:

- **Systemic Fidelity Gap:** The *IBM Marrakesh* system (red solid line) demonstrates higher state survival rates compared to the *VLQ Star24* (green solid line). While IBM hardware maintains a certainty above 0.8 at a depth of approximately 4×10^2 , the VLQ machine approaches the random noise floor ($P \approx 0.125$) significantly earlier, near a depth of 3×10^2 .
- **Simulator Accuracy:** For the IBM Marrakesh, the real hardware performance tracked closely with the *FakeMarrakesh* simulator at low depths but showed a slower decay at high depths, whereas the IQM simulators (*Deneb* and *Garnet*) exhibited a more rapid loss of certainty.
- **Feasibility of ECP Instances:** The vertical dashed lines indicate the circuit depths required for the ECP instances solved in this work. Small instances (e.g., $U=4, S=4$) reside in a region where IBM hardware yields usable signals (above 0.7), but larger instances ($U=4, S=6$) push the circuits into a depth exceeding 103, where both real systems converge toward the baseline of random results.

The experimental results highlight significant architectural disparities between the platforms, a finding corroborated by technical feedback from IQM regarding the benchmark performance. As noted by IQM engineers, the star-shaped topology of the Star-24 QPU necessitates decomposing native CZ gate gates into MOVE-CZ-MOVE sequences to facilitate interactions via the central resonator. This decomposition introduces a topological overhead

that scales more aggressively than the nearest-neighbor interactions on IBM’s lattice-based Marrakesh QPU.

Consequently, the increased gate count and extended circuit duration cause the system to become “limited by T_1 relaxation times quite fast.” Furthermore, the analysis suggests that performance could be optimized by utilizing a fidelity-aware mapping strategy that selects the “best” three qubits based on individual CZ gate and MOVE gates fidelities within the Star-24 system.

Nevertheless, we found that quantum computers hit a performance wall when trying to solve complex optimization tasks—a barrier created by the maximum complexity level before hardware errors dominate. While high-level frameworks like Qrisp simplify the development of the oracle and the overall algorithmic structure, the physical gate errors on current hardware remain the dominant bottleneck. The data suggests that for combinatorial problems like ECP to be solved reliably as they scale in complexity, improvements in physical gate fidelity or more efficient circuit synthesis are required to keep the total depth within these measured limits.

Chapter 7

Qrisp Framework: ECP - Knapsack - SAT Implementation

In this chapter, I discuss the implementation of quantum algorithms by leveraging the high-level abstractions provided by the Qrisp framework, specifically focusing on the Exact Cover Problem (ECP). Building upon the low-level Qiskit implementation introduced in previous chapters, which followed the methodology of existing literature, I explore two distinct algorithmic strategies for solving ECP instances: Grover’s search and the Quantum Approximate Optimization Algorithm (QAOA). The following sections detail how Qrisp allows for the encoding of the quantum oracle and cost functions through high-level syntax, which represents a significantly more intuitive approach than manual, gate-level circuit synthesis.

To establish a comparative baseline, I developed a Grover-based ECP solver using low-level gate operators that mirrors the logic of my earlier Qiskit implementation. Furthermore, to evaluate the versatility of these high-level abstractions, I implemented two additional solvers for the Boolean Satisfiability Problem (SAT) and the Knapsack Problem. These implementations demonstrate that, by utilizing Qrisp’s functional programming style and automatic uncomputation, complex quantum logic remains accessible to software engineering practitioners. This approach highlights that quantum algorithmic development can be made readable and maintainable for IT professionals without requiring deep expertise in low-level quantum gate decomposition.

7.1 ECP QAOA Implementation

As an alternative to exact search algorithms, I have implemented the Quantum Approximate Optimization Algorithm (QAOA) for the Exact Cover Problem. Unlike Grover’s algorithm, which provides a deterministic quadratic speedup for finding a specific state, QAOA is a variational heuristic designed to find sufficiently good solutions for combinatorial optimization problems.

The implementation utilizes the `exact_cover_problem` operator provided by Qrisp’s optimization module. This abstraction allows for a direct mapping from the problem’s set-based description to a quantum Hamiltonian without manual formulation of Ising or QUBO models.

```

def solve(self, depth=3, backend=None, shots=1024):
    # sets_list is a list of sets representing the ECP subsets
    sets_list = [self.subsets[name] for name in self.subset_names]

    # Define the QAOA problem using the built-in ECP operator
    QAOA_problem = exact_cover_problem(sets_list)

    # Execute the variational optimization
    results = QAOA_problem.run(depth=depth, backend=backend, shots=shots)
    return results

```

Listing 7.1: QAOA implementation for ECP in Qrisp.

Comparison with Qiskit: In a standard Qiskit implementation, solving ECP via QAOA typically requires translating the constraints into a QUBO problem manually or using the `qiskit-optimization` module. Each constraint (every element must be covered exactly once) must be expressed as a penalty term $(\sum_{i \in S_j} x_i - 1)^2$ in the cost function.

7.2 ECP: Grover’s Algorithm Using Quantum Gates Operation

To maintain continuity with the research presented in earlier chapters, I first implemented a gate-based version of the ECP solver within the Qrisp environment. This implementation serves as a bridge, replicating the manual oracle construction used in the Qiskit baseline, which is based on the work of Jiang & Wang (2023) [34].

The oracle is constructed using a series of multi-controlled X gates (*mcx*) and custom controlled counters to verify that each element of the universe is represented by exactly one selected subset.

```

def gate_level_oracle(self, subset_qubits, element_counters, phase_qubit):
    # Manually increment counters based on subset selections
    for i, subset_name in enumerate(self.subset_names):
        for element in self.subsets[subset_name]:
            element_idx = self.universe.index(element)
            self.controlled_increment(
                subset_qubits[i], element_counters[element_idx])

    # Check if all counters equal 1
    self.check_exact_cover(element_counters, phase_qubit)

    # Manual uncomputation is required in this gate-level approach
    self.check_exact_cover(element_counters, phase_qubit)
    for i in reversed(range(len(self.subset_names))):
        for element in self.subsets[self.subset_names[i]]:
            element_idx = self.universe.index(element)
            self.controlled_decrement(
                subset_qubits[i], element_counters[element_idx])

```

Listing 7.2: Manual oracle construction mimicking the Qiskit baseline.

This hybrid approach allowed for a direct verification of the Qrisp execution pipeline while ensuring that the underlying logic remained identical to the established literature baseline.

7.3 ECP: Grover’s Algorithm Fully Utilizing Qrisp Abstraction

By utilizing the full suite of Qrisp’s high-level features, the complexity of the ECP oracle is significantly reduced. This implementation utilizes *QuantumArray* to represent subset selections and *QuantumFloat* for element counting.

The core advantage of this approach is the `@auto_uncompute` decorator. Qrisp tracks the addition operations and applies their inverses automatically, eliminating the manual labor and potential for error in the uncomputation phase.

```
@auto_uncompute
def ECP_oracle(self, subset_selectors: QuantumArray, phase=np.pi):
    # Initialize a counter for each element in the universe
    element_counts = [QuantumFloat(2) for _ in self.universe]

    for i, subset_name in enumerate(self.subset_names):
        # High-level conditional logic
        with subset_selectors[i]:
            for element in self.subsets[subset_name]:
                element_idx = self.universe.index(element)
                element_counts[element_idx] += 1

    # Condition: All elements are covered exactly once (count == 1)
    is_exact_cover = QuantumBool()
    x(is_exact_cover) # Assume True
    for count in element_counts:
        is_exact_cover &= (count == 1)

    p(phase, is_exact_cover)
```

Listing 7.3: ECP Oracle using high-level Qrisp abstractions.

Comparison with Qiskit: The Qiskit original reference implementation requires hundred and more lines of code to define a `ControlledCounter` class and wire it correctly to handle bit-carries and ancilla management. In contrast, the Qrisp implementation shown in Listing 7.3 leverages native arithmetic and the `auto_uncompute` feature to express the same logic in approximately 15 lines of highly readable, maintainable code. This demonstrates the efficiency of high-level abstractions in bridging the gap between abstract problem definitions and low-level quantum circuit execution.

7.4 Benchmark Instances and Expected Results for Exact Cover Problem

To evaluate and benchmark the performance of the QAOA and Grover-based ECP implementations in Qrisp, six distinct problem instances were curated. These instances range

from trivial validation cases to complex scenarios involving multiple overlapping subsets and larger universes. Notably, the literature baseline instance is derived from the work of Jiang Wang (2023) [34], which was also utilized for the evaluation of the ECP solver implementation in Qiskit in previous chapters.

Each instance is defined by a universe U and a collection of subsets S . A solution is considered valid if the union of the selected subsets equals U and all selected subsets are pairwise disjoint.

1. **Literature Baseline (Paper Example)** This instance serves as the primary comparative link to established research.
 - $U = \{0, 1, 2\}$
 - $S = \{S_A : \{2\}, S_B : \{0, 2\}, S_C : \{0\}, S_D : \{1\}, S_E : \{0, 1\}\}$
 - Expected Solutions: $\{S_A, S_C, S_D\}$, $\{S_A, S_E\}$, and $\{S_B, S_D\}$
2. **Experiment 0: Minimal Trivial Case** A basic validation test to ensure the solvers can identify a single, perfect-match subset.
 - $U = \{0, 1, 2\}$
 - $S = \{S_A : \{2\}, S_B : \{0, 1, 2\}\}$
 - Expected Solution: $\{S_B\}$
3. **Experiment 1: Disjoint Verification** Evaluates the ability to distinguish between a single element subset and a multi-element subset that completes the cover.
 - $U = \{0, 1, 2\}$
 - $S = \{S_A : \{2\}, S_B : \{0, 2\}, S_C : \{1\}, S_D : \{2\}\}$
 - Expected Solution: $\{S_B, S_C\}$
4. **Experiment 2: Multiple Valid Paths** A 4-element universe designed to test if the solvers can find multiple distinct covers of the same size.
 - $U = \{0, 1, 2, 3\}$
 - $S = \{S_A : \{1\}, S_B : \{0, 2\}, S_C : \{1\}, S_D : \{3\}\}$
 - Expected Solutions: $\{S_A, S_B, S_D\}$ and $\{S_B, S_C, S_D\}$
5. **Experiment 3: Increased Overlap Complexity** This instance introduces six subsets for a 4-element universe, creating a high degree of interference for the Grover oracle and higher-order constraints for the QAOA Hamiltonian.
 - $U = \{0, 1, 2, 3\}$
 - $S = \{S_A : \{1\}, S_B : \{0, 2\}, S_C : \{2\}, S_D : \{2, 3\}, S_E : \{3\}, S_F : \{0\}\}$
 - Expected Solutions: $\{S_A, S_B, S_E\}$, $\{S_A, S_D, S_F\}$, and $\{S_A, S_C, S_E, S_F\}$
6. **Experiment 4: Larger Universe Constraint** A 5-element universe intended to test the scalability of the high-level Qrisp arithmetic abstractions and the depth of the resulting circuit.
 - $U = \{0, 1, 2, 3, 4\}$
 - $S = \{S_A : \{1\}, S_B : \{0, 2\}, S_C : \{2\}, S_D : \{2, 3\}, S_E : \{4\}, S_F : \{0\}\}$
 - Expected Solution: $\{S_A, S_D, S_E, S_F\}$

7.5 Knapsack: Grover’s Algorithm Fully Utilizing Qrisp Abstraction

To demonstrate the versatility of high-level abstractions, I implemented a solver for the Knapsack problem. This problem requires selecting items with specific weights such that the total weight falls within a target capacity—in this implementation, specifically a weight strictly less than the capacity and greater than zero.

This solver makes extensive use of *QuantumFloat*, which allows for fixed-point arithmetic directly on quantum states. By representing weights as *QuantumFloat* variables rather than raw qubits, the logic becomes readable to any developer familiar with classical programming. The framework handles the synthesis of the underlying ripple-carry adders or other arithmetic circuits, ensuring that the developer can focus on the algorithmic constraints.

The implementation consists of an evaluation function and the Grover oracle. The evaluation logic (Listing 7.4) utilizes Qrisp’s conditional execution blocks to sum the weights of items based on the state of the `selectors` array.

```
def evaluate_knapsack(self, selectors):
    # Calculate required bits for the sum of all possible weights
    bit_count_max_weight = \
        int(np.ceil(np.log2(sum(self.items.values())))) + 1
    total_weight = QuantumFloat(bit_count_max_weight)

    for item_idx, w in self.items.items():
        # Conditional addition: add weight only if selector bit is |1>
        with selectors[item_idx]:
            total_weight += w
    return total_weight
```

Listing 7.4: Knapsack weight evaluation logic using *QuantumFloat*.

The oracle (Listing 7.5) then applies the boolean constraints to this `total_weight`. Because this method is decorated with `@auto_uncompute`, Qrisp automatically generates the inverse gates required to clean up the `total_weight` and `cond` variables after the phase shift is applied, ensuring the quantum state remains clean for subsequent Grover iterations.

```
@auto_uncompute
def knapsack_oracle(self, selectors: QuantumArray, phase=np.pi):
    total_weight = self.evaluate_knapsack(selectors)
    # Define constraints using high-level boolean operators
    cond = (total_weight < self.weight) & (total_weight > 0.0)
    # Flip the phase of states that satisfy the condition
    p(phase, cond)
```

Listing 7.5: Knapsack Grover Oracle with automatic uncomputation.

7.5.1 Benchmark Instances and Expected Results for Knapsack Problem

To evaluate the capabilities of the Knapsack solver implemented in Qrisp, three distinct problem instances were selected for experimentation. In these instances, the state is represented by a bitstring $b \in \{0,1\}^n$, where $b_i = 1$ indicates that the i -th item is selected for the knapsack and $b_i = 0$ indicates it is excluded.

- **Experiment 1:** This instance uses a weight limit $W_{limit} = 5$ and three items with weights $\{2, 2, 6\}$. The search identifies 3 valid solutions where the total weight $W_{total} < W_{limit}$ and $W_{total} > 0$:
 - 100 (Weight: 2)
 - 010 (Weight: 2)
 - 110 (Weight: 4)
- **Experiment 2:** This instance uses a weight limit $W_{limit} = 6$ and five items with weights $\{2, 3, 4, 6, 6\}$. The search identifies 4 valid solutions:
 - 10000 (Weight: 2)
 - 01000 (Weight: 3)
 - 00100 (Weight: 4)
 - 11000 (Weight: 5)
- **Experiment 3:** This instance uses a weight limit $W_{limit} = 6$ and five items with weights $\{2, 3, 4, 1, 2\}$. This configuration yields a larger set of 13 valid solutions:
 - 10000 (Weight: 2)
 - 01000 (Weight: 3)
 - 00100 (Weight: 4)
 - 00010 (Weight: 1)
 - 00001 (Weight: 2)
 - 11000 (Weight: 5)
 - 00011 (Weight: 3)
 - 10001 (Weight: 4)
 - 10010 (Weight: 3)
 - 01010 (Weight: 4)
 - 01001 (Weight: 5)
 - 00110 (Weight: 5)
 - 10011 (Weight: 5)

7.6 SAT Grover’s Algorithm Fully Utilizing Qrisp

The Boolean Satisfiability Problem (SAT) was chosen as a final example due to its fundamental role in computer science. The implementation accepts inputs in a format similar to Conjunction Normal Form (CNF), where a formula Φ is represented as a conjunction of clauses: $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$.

The input to the solver is structured as an array of sets (or tuples), where each set represents a clause. Variables are identified by their integer index. To handle literals, positive integers represent the variable itself, while negative integers represent its negation. Specifically, for a variable x_i , the literal x_i is encoded as the index i , and the literal $\neg x_i$ is encoded as $-(i+1)$. For example, an input of $[(0,), (-2,), (2,)]$ translates to the logic formula $x_0 \wedge \neg x_1 \wedge x_2$. This mapping was chosen as a concise way to define the problem and to simplify the construction of the quantum algorithm.

```
@auto_uncompute
def sat_oracle(self, vars_array: QuantumArray, phase=np.pi):
    clause_qubits = []
    for clause in self.clauses:
        clause_sat = QuantumBool()
        x(clause_sat) # Assume True

        ctrl_qubits = []
        for lit in clause:
            temp_lit = QuantumBool()
            if lit >= 0: # Positive literal
                x(temp_lit)
                cx(vars_array[lit], temp_lit)
            else: # Negative literal
                cx(vars_array[abs(lit) - 1], temp_lit)
            ctrl_qubits.append(temp_lit)

        # If ALL literals in the clause are False, the clause is False
        mcx(ctrl_qubits, clause_sat)
        clause_qubits.append(clause_sat)

    # The formula is satisfied if ALL clauses are True
    all_sat = QuantumBool()
    mcx(clause_qubits, all_sat)
    p(phase, all_sat)
```

Listing 7.6: SAT Oracle implementing CNF logic in Qrisp.

Although the implementation uses some gate-level operators like MCX gate for the clause verification logic, it remains wrapped in Qrisp’s high-level *auto_uncompute* context. This ensures that the complex chain of AND/OR logic required to satisfy multiple clauses does not leave residual information on auxiliary qubits. The solver demonstrates how Grover’s algorithm can be used to identify assignments that satisfy all clauses simultaneously, providing a clear example of how Qrisp can be used to prototype combinatorial solvers quickly.

7.6.1 Benchmark Instances and Expected Results for SAT Problem

The implementation was validated using three formula instances of increasing complexity. Solutions are represented as bitstrings where 1 denotes *True* and 0 denotes *False* for the variables $(x_0, x_1, \dots, x_n)$.

- **Experiment 1:** A 3-variable formula defined as $(x_0) \wedge (\neg x_1) \wedge (x_2)$. There is a single correct solution:

– 101

- **Experiment 2:** A 4-variable formula defined as $(x_0 \vee x_1 \vee x_2) \wedge (\neg x_0 \vee \neg x_1 \vee \neg x_2) \wedge (x_0 \vee x_3) \wedge (\neg x_3)$. This instance identifies 3 correct solutions:

– 1000

– 1100

– 1010

- **Experiment 3:** A formula involving variables (x_0, x_1, x_2) defined as $(x_0 \vee x_1 \vee x_2) \wedge (\neg x_0 \vee \neg x_1) \wedge (\neg x_3)$. This yields 2 correct solutions:

– 100

– 010

7.7 Qrisp Framework: Integration with VLQ and IBM HW Backends

One of the strongest arguments for using Qrisp is its backend-agnostic architecture. The framework provides a *BackendClient* class that defines a standardized interface for circuit execution. When a Qrisp program is executed, the high-level logic is compiled into an OpenQASM string, which is then passed to the target backend.

Qrisp includes built-in support for the *QiskitBackend*, allowing for immediate integration with IBM’s fleet of quantum processors, such as the *IBM Marrakesh* used in my experiments. For more specialized hardware, such as the *VLQ Star24* at IT4Innovations, integration is achieved through a middleware layer known as Quantum-as-a-Service (QaaS).

7.7.1 Quantum-as-a-Service and the Modern Software Stack

The Quantum-as-a-Service (QaaS) platform [30], developed at IT4Innovations, acts as a vital bridge between high-level frameworks and localized quantum hardware. Similar to the Munich Quantum Software Stack (MQSS), QaaS addresses several critical challenges in the current quantum ecosystem:

- **Environment Fragility:** Quantum providers often require specific, conflicting versions of libraries like Qiskit or NumPy. QaaS isolates these dependencies, preventing the „dependency hell“ often encountered when multiple hardware providers are installed in a single Python environment.

- **Heterogeneous Hardware:** As part of the EuroHPC JU initiative, Europe hosts diverse hardware types - including superconducting, neutral-atom, and trapped-ion systems. QaaS aims to provide a unified API and uniform access to quantum computers deployed for research and science within EuroHP JU initiative and other European projects, allowing Qrisp circuits to run on different architectures with easy access and without unnecessary code changes. Changes may be needed for special architectures, but the aim is not to force them.
- **Interface Volatility:** Quantum hardware interfaces change rapidly. Middleware like QaaS or MQSS absorbs these changes, ensuring that research code remains functional even when the underlying hardware provider updates their API.
- **Advanced Hardware Features:** Specialized access, such as pulse-level control (*Pulla* or *Pulse* on IQM systems), is often restricted to specific hosting entities. Middleware allows for secure, authenticated access to these low-level features while maintaining a high-level user interface.

By utilizing QaaS, I was able to execute the Qrisp-based ECP solvers on both the IBM Marrakesh and the IT4Innovations VLQ system using nearly identical code, demonstrating the viability of high-level frameworks for cross-platform quantum research.

7.8 Qrisp: Summary and Outcomes of Results

To evaluate the proposed implementations, a comprehensive suite of execution scripts was developed:

- `ecp_qrisp_example.py`
- `knapsack_qrisp_example.py`
- `sat_qrisp_example.py`

Along with their respective hardware-interfacing counterparts. The problem instances defined in Sections 7.4, 7.5.1, and 7.6.1 were executed across a variety of backends: the native *Qrisp Simulator*, the Qiskit *AerSimulator* (ideal), the *IBM FakeMarrakesh* noisy model (simulating the noise profile of the Heron R2 architecture), and finally the physical *IBM Marrakesh* quantum processor. Due to technical unavailability during the evaluation window, benchmarks on the IT4I/VLQ Star24 backend could not be completed. This section details the comparative methodology and the performance outcomes of the Qrisp-based solvers.

Statistical Distinguishability and Z-Score Methodology

To objectively determine if a quantum solver successfully identifies the correct solutions amidst experimental noise, a four-stage statistical distinguishability pipeline was implemented. This ensures that a correct result is not merely a product of random fluctuations in a noise-dominated distribution.

1. **Filtering:** Trivial states and decoherence sinks (e.g. the all-zero state $|00\dots0\rangle$ or invalid bitstrings in specific encodings) are removed to prevent them from skewing the probability mass.

2. Renormalization: The remaining probabilities are rebalanced to sum to 1.0, and the effective shot count is adjusted relative to the valid data pool.
3. Identifying Boundary Probabilities:
 - p_1 : The worst-case success, the lowest probability measured among the set of expected valid solutions.
 - p_2 : The best-case failure, the highest probability measured among the set of incorrect states.
4. Z-Score Calculation: A two-proportion Z-test is performed to compare p_1 and p_2 . The resulting Z-score quantifies the confidence that the solver’s success is statistically significant, calculated as:

$$Z = \frac{p_1 - p_2}{\sqrt{\frac{p_1(1-p_1)}{N_{\text{eff}}} + \frac{p_2(1-p_2)}{N_{\text{eff}}}}} \quad (7.1)$$

where N_{eff} is the effective shot count post-renormalization. The bounds are interpreted as follows:

- $Z > 2.58$: Highly significant (99% confidence).
- $Z > 1.96$: Significant (95% confidence).
- $Z \leq 0$: Noise-dominated or failed resolution.

7.8.1 Evaluation of Circuit Depth and Result Certainty

The experimental data confirms that circuit depth is the primary determinant of success on near-term hardware. As demonstrated in the depth benchmark experiments (see Section 6.4.2), there is a clear threshold beyond which hardware noise collapses the state’s certainness.

On the IBM Marrakesh hardware, successful resolution was achieved in only three instances. The most robust results were observed for the SAT solver instances, as the SAT solver represents one of the most direct mappings of logic to gates in Qrisp. Specifically, instances with transpiled depths of 109 and 233 gates yielded results with 99% confidence. In contrast, the ECP Grover solver reached a 95% confidence verdict only for the most trivial case (*Experiment 0*), which required a depth of 340 gates. All other ECP and Knapsack instances produced circuits exceeding the coherence limits of the Heron R2 chip, resulting in noise-dominated distributions where valid solutions were indistinguishable from failures (see Tables 7.4 and 7.5).

When comparing these results with the FakeMarrakesh simulator, the simulator’s results were 99% distinguishable for all three SAT problem instances. Overall, the simulator only incorrectly predicted distinguishable results for one case: SAT Instance 2.QAOA was not executed on the physical hardware due to its extreme computational demand; specifically, each optimization iteration requires the circuit to be re-executed on the hardware. For a single problem instance, this can accumulate to thousands of quantum seconds, whereas Grover’s algorithm requires only a single circuit execution per task. Consequently, a direct hardware comparison for QAOA is unavailable. However, on the FakeMarrakesh simulator, it achieved respectable results for Experiment 0 (see Table 7.1) and Experiment 1 (see Table 7.2), though it failed for the larger problem distilled from [34]. These results remained

uncertain, as shown in Table 7.3, likely due to an insufficient number of iterations or the specific heuristic chosen for the optimizer.

Because the simulations required significant time-often running for tens of minutes-future execution would be better suited for more powerful hardware or an HPC node. Consequently, we were only able to execute it a few times on our available system (Ryzen 5 5600, 6 cores, 32GB RAM). Despite these constraints, the research community continues to see significant potential in the QAOA approach, with new frameworks and solvers emerging frequently. For example, QHyper by Cyfronet [37] is a Python library covering both gate-based (QAOA) and quantum annealing approaches (such as those for D-Wave systems).

The confidence levels produced by the *AerSimulator* and the *Qrisp Simulator* are not identical. These differences can be attributed to the distinct simulation approaches and noise-modeling techniques discussed in Section 3.8.

These findings demonstrate that the high gate count inherent in complex oracles remains the primary bottleneck for practical execution on current NISQ architectures. The quantitative breakdown of this performance-including the specific boundary probabilities p_1, p_2 and the resulting Z-score calculations for each experiment-is provided in Appendices 1 and 2, offering a detailed evaluation of the statistical certainty achieved across different backends.

Table 7.1: Results for the Exact Cover Problem (ECP) Experiment 0, utilizing the QAOA solver on the *IBM FakeMarrakesh* backend. Evaluation executed with 3000 shots and maximum of 500 iterations.

Rank	States	Probability	Percentage
1	B	0.807	80.70%
2	A	0.08566666666666667	8.57%
3	Nothing Selected	0.025	2.50%

Table 7.2: Results for the Exact Cover Problem (ECP) Experiment 1, utilizing the QAOA solver on the *IBM FakeMarrakesh* backend. Evaluation executed with 3000 shots and maximum of 1000 iterations.

Rank	States	Probability	Percentage
1	B, C	0.319	31.90%
1	A, C, D	0.16733333333333333	16.74%
2	A, B, C	0.088	8.80%
3	C	0.08533333333333333	8.53%
4	B, C, D	0.07566666666666666	7.57%
5	A, C	0.06566666666666666	6.57%
6	C, D	0.06566666666666666	6.57%
7	B	0.037333333333333336	3.73%
8	A, B, C, D	0.027666666666666666	2.77%
9	A, D	0.017	1.70%
10	Nothing Selected	0.012333333333333333	1.23%
11	D	0.012	1.20%
12	A	0.008333333333333333	0.83%
13	B, D	0.008333333333333333	0.83%
14	A, B	0.008	0.80%
15	A, B, D	0.0023333333333333335	0.23%

Table 7.3: Results for the Exact Cover Problem (ECP) Paper Example, utilizing the QAOA solver on the *IBM FakeMarrakesh* backend. Evaluation executed with 3000 shots and maximum of 1250 iterations.

Rank	States	Probability	Percentage
1	B, C, D	0.12066666666666667	12.07%
2	A, C, E	0.11933333333333333	11.93%
3	B, E	0.092	9.20%
4	A, D	0.08833333333333333	8.83%
5	A, C, D	0.051	5.10%
6	A, C, D, E	0.043333333333333335	4.33%
7	A, B, C, D	0.041666666666666664	4.17%
8	Nothing Selected	0.03666666666666667	3.67%
9	E	0.035666666666666666	3.57%
10	B	0.029666666666666668	2.97%
11	B, D	0.025	2.50%
12	A, E	0.023666666666666666	2.37%
13	A, C	0.021666666666666667	2.17%
14	C, D	0.021333333333333333	2.13%
15	A	0.021	2.10%
16	D	0.020666666666666667	2.07%
17	C, E	0.020333333333333333	2.03%
18	C	0.019	1.90%
19	A, B, D, E	0.016666666666666666	1.67%
20	A, B, E	0.016333333333333333	1.63%
21	C, D, E	0.016333333333333333	1.63%
22	B, C, E	0.015333333333333332	1.53%
23	B, C	0.014	1.40%
24	B, C, D, E	0.013	1.30%
25	B, D, E	0.012666666666666666	1.27%
26	A, B, D	0.012333333333333333	1.23%
27	A, B, C, E	0.012	1.20%
28	A, B, C	0.01	1.00%
29	A, B	0.009333333333333334	0.93%
30	A, B, C, D, E	0.008	0.80%
31	A, D, E	0.007	0.70%
32	D, E	0.006	0.60%

Table 7.4: Performance evaluation of quantum algorithms on various problem instances and backends.

Abbreviations: ECP = Exact Cover Problem, KNAP = Knapsack Problem, SAT = Boolean Satisfiability Problem, QAOA = Quantum Approximate Optimization Algorithm, Grover (Gate) = Grover’s algorithm (gate level), Grover (High) = Grover’s algorithm (high abstraction level).

Backends/Metrics: IM = IBM Marrakesh, IFM = IBM FakeMarrakesh, OC = Original Circuit, D = Depth, Q = Qubits, V = Verdict.

NOTE: IBM FakeMarrakesh = Qiskit AerSimulator using IBM’s Marrakesh noisy model; IBM Marrakesh = IBM quantum device (Heron r2 chip).

Verdicts: 99% = z-score > 2.58, 95% = z-score > 1.96, FAIL = unable to recognize correct solutions, INC = Inconclusive.

Problem	Instance	Algorithm	IM (D)	IM+IFM (Q)	IM (V)	IFM (D)	IFM (V)	OC (D)	OC (Q)	Shots
ECP	PaperExample	QAOA	-	156	-	131-136	FAIL	39-46	5	3000
ECP	PaperExample	Grover (Gate)	4326	156	FAIL	4139	FAIL	15	117	2000
ECP	PaperExample	Grover (High)	1583	156	FAIL	1532	FAIL	16	330	2000
ECP	Experiment0	QAOA	-	156	-	16-17	99%	14-20	2	3000
ECP	Experiment0	Grover (Gate)	192	156	FAIL	197	INC	2	2	2000
ECP	Experiment0	Grover (High)	340	156	95%	319	INC	96	13	2000
ECP	Experiment1	QAOA	-	156	-	89-94	99%	29-34	4	3000
ECP	Experiment1	Grover (Gate)	954	156	FAIL	901	INC	44	11	2000
ECP	Experiment1	Grover (High)	835	156	FAIL	833	FAIL	196	15	2000
ECP	Experiment4	QAOA	-	156	-	93-98	FAIL	31-37	6	3000
ECP	Experiment4	Grover (Gate)	7045	156	FAIL	7112	FAIL	180	22	2000
ECP	Experiment4	Grover (High)	2852	156	FAIL	2888	FAIL	542	25	2000
KNAP	Instance1	Grover (High)	3114	156	FAIL	3114	FAIL	640	12	1000
KNAP	Instance2	Grover (High)	4464	156	FAIL	4473	FAIL	781	15	1200
KNAP	Instance3	Grover (High)	3645	156	FAIL	3645	FAIL	671	14	1200
SAT	Instance1	Grover (High)	109	156	99%	109	99%	55	10	1000
SAT	Instance2	Grover (High)	506	156	FAIL	499	99%	139	18	1000
SAT	Instance3	Grover (High)	233	156	99%	235	99%	79	13	1000

Table 7.5: Performance evaluation of quantum algorithms on various problem instances and backends.

Abbreviations: ECP = Exact Cover Problem, KNAP = Knapsack Problem, SAT = Boolean Satisfiability Problem, QAOA = Quantum Approximate Optimization Algorithm, Grover (Gate) = Grover’s algorithm (gate level), Grover (High) = Grover’s algorithm (high abstraction level).

Backends/Metrics: IA = Ideal AerSimulator, QS = Qrisp Simulator, OC = Original Circuit, D = Depth, Q = Qubits, V = Verdict.

NOTE: IBM FakeMarrakesh = Qiskit AerSimulator using IBM’s Marrakesh noisy model; IBM Marrakesh = IBM quantum device (Heron r2 chip).

Verdicts: 99% = z-score > 2.58, 95% = z-score > 1.96, FAIL = unable to recognize correct solutions, INC = Inconclusive.

Problem	Instance	Algorithm	IA (D)	IA (Q)	IA (V)	QS (Q)	QS (V)	OC (D)	OC (Q)	Shots
ECP	PaperExample	QAOA	39	5	FAIL	5	FAIL	39-46	5	3000
ECP	PaperExample	Grover (Gate)	457	15	99%	15	99%	15	117	2000
ECP	PaperExample	Grover (High)	243	16	99%	16	FAIL	16	330	2000
ECP	Experiment0	QAOA	2	2	99%	2	99%	14-20	2	3000
ECP	Experiment0	Grover (Gate)	20	6	99%	6	99%	2	2	2000
ECP	Experiment0	Grover (High)	69	13	INC	13	INC	96	13	2000
ECP	Experiment1	QAOA	29	4	99%	4	99%	29-34	4	3000
ECP	Experiment1	Grover (Gate)	144	11	99%	11	99%	44	11	2000
ECP	Experiment1	Grover (High)	146	15	99%	15	99%	196	15	2000
ECP	Experiment4	QAOA	31	6	FAIL	6	FAIL	31-37	6	3000
ECP	Experiment4	Grover (Gate)	868	22	99%	22	99%	180	22	2000
ECP	Experiment4	Grover (High)	411	25	99%	25	99%	542	25	2000
KNAP	Instance1	Grover (High)	533	12	99%	12	99%	640	12	1000
KNAP	Instance2	Grover (High)	652	15	99%	15	99%	781	15	1200
KNAP	Instance3	Grover (High)	542	14	FAIL	14	FAIL	671	14	1200
SAT	Instance1	Grover (High)	37	10	99%	10	99%	55	10	1000
SAT	Instance2	Grover (High)	109	18	99%	18	99%	139	18	1000
SAT	Instance3	Grover (High)	54	13	99%	13	99%	79	13	1000

7.8.2 Discussion of Knapsack and SAT Complexity Trade-offs

A critical observation during the implementation of the Knapsack solver was the impact of high-level arithmetic abstractions. While Qrisp’s `QuantumFloat` and `QuantumArray` offer significant developer productivity, they synthesize into highly complex circuits involving numerous ancilla qubits and deep uncomputation logic.

To mitigate this, an attempt was made to reduce the Knapsack problem to a Boolean Satisfiability (SAT) problem. The hypothesis was that representing weight constraints as CNF clauses might bypass the depth overhead of `QuantumFloat` arithmetic. A simplified Knapsack instance was designed using three weight categories (capacities up to 50, 100, and 150) and a set of four items.

Case Study: Categorical Weight Mapping for SAT

In this section, we explore a case study of categorical weight mapping for a knapsack problem.

Problem setup

Let boolean variables $x_i \in \{0, 1\}$ indicate selection of item i . Items are categorized as:

- $a, b \in A$ (Light)
- $c, d \in B$ (Medium)
- $e \in C$ (Heavy)

Constraints (valid sets):

- Any combination of up to three items from category A .
- One item from A and one item from B is allowed.
- Exactly one item from category B .
- No items from category C may be combined with any other item.

SAT encoding (exclusivity logic)

For each forbidden combination add a CNF clause.

Exclusivity of category B (forbid c and d together):

$$\neg(x_c \wedge x_d) \iff (\neg x_c \vee \neg x_d).$$

Category C isolation (heavy item e cannot coexist with any other item):

$$(\neg x_e \vee \neg x_a) \wedge (\neg x_e \vee \neg x_b) \wedge (\neg x_e \vee \neg x_c) \wedge (\neg x_e \vee \neg x_d).$$

Overfill prevention (forbid two A items plus one B ; example clause):

$$(\neg x_a \vee \neg x_b \vee \neg x_c).$$

Select at least one item:

$$(x_a \vee x_b \vee x_c \vee x_d \vee x_e).$$

Final CNF

For the example items a, b, c, d, e mapped to integers 0, 1, 2, 3, 4 respectively, the CNF becomes:

$$\begin{aligned}
 f(x) = & \underbrace{(-3 \vee -4)}_{\text{B exclusivity}} \wedge \underbrace{(-5 \vee -1) \wedge (-5 \vee -2) \wedge (-5 \vee -3) \wedge (-5 \vee -4)}_{\text{C isolation}} \\
 & \wedge \underbrace{(-1 \vee -2 \vee -3) \wedge (-1 \vee -2 \vee -4)}_{\text{Overfill (2A + 1B)}} \\
 & \wedge \underbrace{(0 \vee 1 \vee 2 \vee 3 \vee 4)}_{\text{Select at least 1 item}}.
 \end{aligned}$$

Practical implications for NISQ hardware

Consequently, the categorical-to-SAT mapping leads to an exponential growth in forbidden subsets and clause counts, resulting in a Grover oracle that is too deep for current NISQ hardware (e.g., IBM Marrakesh). Therefore, maintaining higher-level arithmetic representations in Qrisp proves more scalable, as it better respects the coherence-time limitations of contemporary devices. In the case study presented above, the circuit depth for IBM Marrakesh reached 1,290 using 35 qubits. This is well above the coherence levels of both IBM Marrakesh and VLQ; furthermore, it requires more qubits than the VLQ architecture possesses. Even when considering a variant where selecting zero items was permitted-allowing for the removal of one clause-the depth remained high at 1,121 with 29 qubits.

Chapter 8

Outcomes and Summary

This final chapter synthesizes the practical implementations and experimental data presented throughout this thesis to provide a comprehensive assessment of the current state of hybrid quantum computing. The work is analyzed through the lens of modern computational infrastructure and the hardware limitations that currently define the boundaries of quantum acceleration.

The chapter is structured into three primary components. First, we examine the systemic requirements for integrating QPUs into HPC environments, focusing on data bandwidth and the co-location of resources. Second, we provide a summary comparison of the execution results across various backends-ranging from ideal simulators to physical NISQ hardware-to identify the current „complexity ceiling“ for quantum acceleration. Finally, we conclude with a distillation of key findings and lessons learned, offering a roadmap for future developments in hybrid algorithm design and the hardware improvements necessary to transition from proof-of-concept experiments to viable industrial applications.

8.1 Discussion about HPC-QC Integration

The deployment of quantum computers alongside High-Performance Computing (HPC) resources is becoming a standard practice within European research and science hosting entities. As argued in [38], integrating quantum computers directly into HPC centers is essential for transforming them from experimental laboratory devices into practical quantum accelerators. By co-locating the hardware, researchers eliminate the high latency associated with cloud access, enabling the efficient execution of hybrid algorithms. In these workflows, classical supercomputers handle heavy data processing while QPUs solve specific, computationally complex sub-problems. This integration leverages existing HPC job-scheduling infrastructure (such as SLURM) and industrial-grade cooling systems, providing a standardized environment that makes quantum computing power accessible to domain scientists through a unified software stack.

To facilitate this, close cooperation between schedulers is required to synchronize time-window reservations on both devices. Alternatively, the use of interruptible code-which can be re-triggered once the preceding classical or quantum phase completes-is necessary. The optimal approach depends on the problem type: some workloads may have a well-defined set of quantum circuits with pre-determined input parameters after a few iterations, whereas others, such as the Quantum Approximate Optimization Algorithm (QAOA), require the generation of new parameters dynamically after each quantum circuit execution.

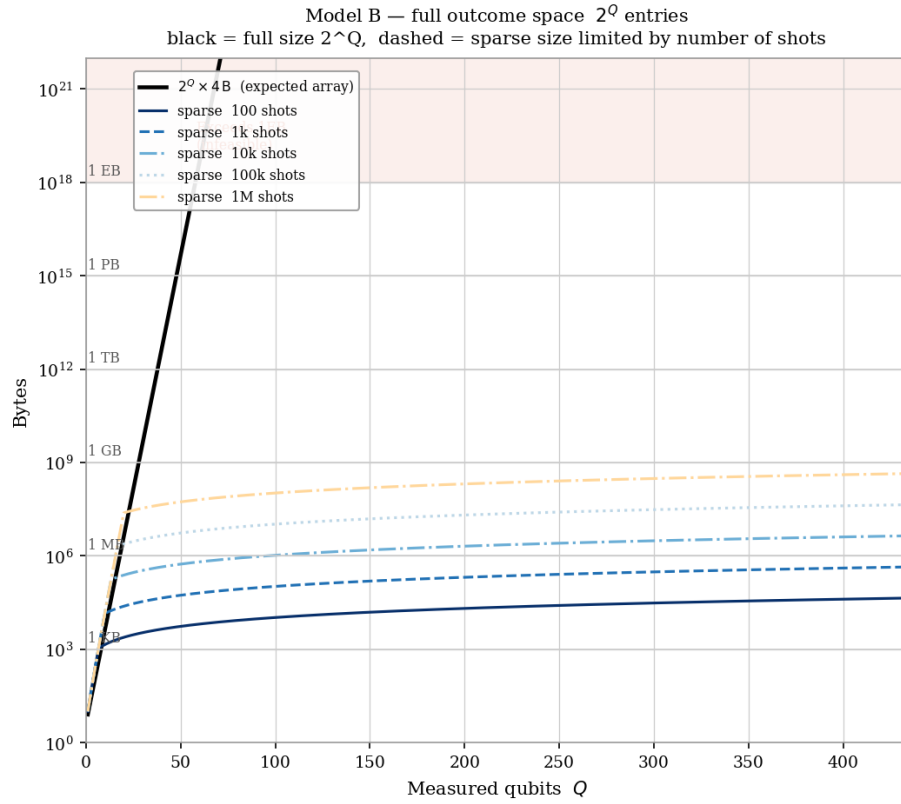
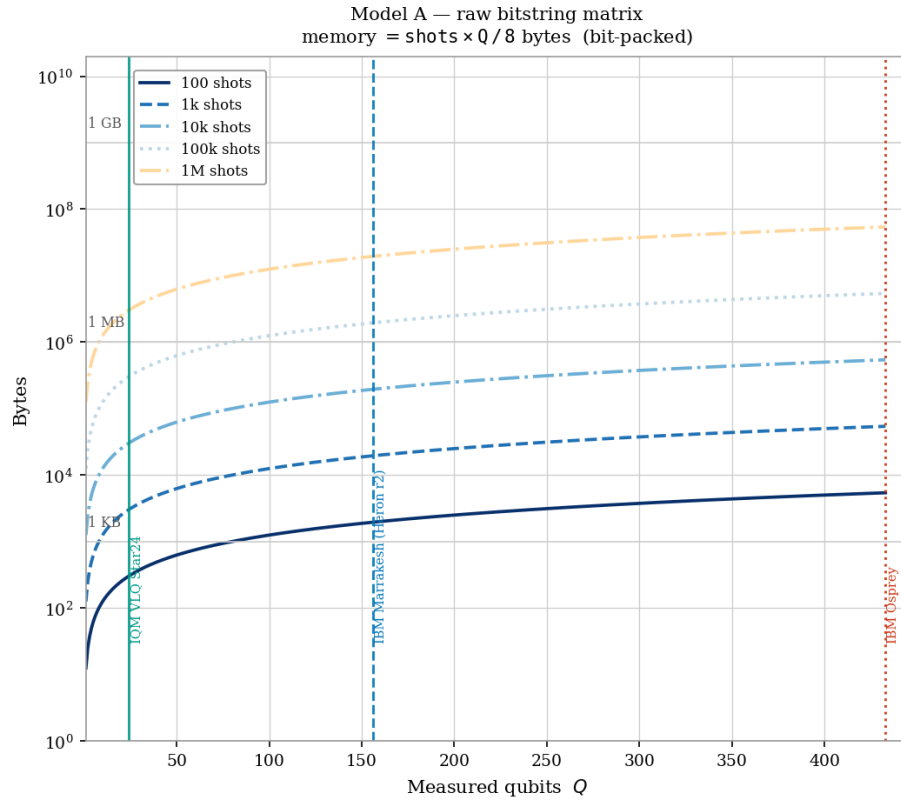


Figure 8.1: Memory Requirements for Sampled Quantum Results

The argument that memory and bandwidth are limiting factors further justifies HPC integration. Even with current near-term chips of up to 433 qubits, the data generated peaks at approximately 1GB. For continuous hybrid applications, transferring this volume of data across a high-speed 100Gbps internal link offers a significant speedup; such a transfer takes a mere 80 ms, ensuring that the interconnect does not become a bottleneck for the iterative classical-quantum loop. Table 8.1 describes the memory requirements for sampling results based on the number of shots, qubit count, and the output model. Model A represents an array of sampled bit-arrays without aggregation. Model B presents two variants: Full, where a counter is allocated for every possible outcome, and Sparse, where counters are only allocated for outcomes that actually occurred. Since the number of unique occurrences is capped by the number of shots, the actual memory footprint is often smaller than the theoretical maximum shown in the diagram.

8.2 Results of Evaluation on real HW and Simulators

The experimental evaluation of the implemented quantum solvers demonstrates a clear correlation between circuit complexity and the reliability of results on current NISQ architectures. By executing problem instances across local simulators, noisy models, and physical quantum hardware, we identified the current operational boundaries for practical hybrid applications.

To objectively quantify the success of each execution, a statistical distinguishability pipeline was utilized to compute Z-scores (see Section 7.8). These scores indicate the confidence that a correct result is clearly distinguishable from incorrect states in returned state-vector

Comparison of Results for Different Backends

Native and Ideal Simulators: Both the Qrisp Simulator and the AerSimulator (ideal) successfully identified correct solutions across all problem domains with 99% confidence. Discrepancies in the specific certainty levels between the two are attributed to their differing simulation architectures.

IBM FakeMarrakesh (Noisy Model): The noisy simulator correctly predicted distinguishable results (99% confidence) for all three SAT problem instances. However, it exhibited failures in larger problem sizes, such as the ECP solver distilled from existing literature, likely due to insufficient optimizer iterations or specific heuristic choices.

Physical IBM Marrakesh (Heron R2): Successful resolution on real hardware was limited to three specific instances where circuit depth remained manageable:

- **SAT Solver:** Produced the most robust hardware results, with instances at transpiled depths of 109 and 233 gates yielding 99% confidence.
- **ECP Grover Solver:** Achieved a 95% confidence verdict only for the most trivial case (Experiment 0), which required 340 gates.
- **Knapsack and Complex ECP:** All other instances exceeded the coherence limits of the Heron R2 chip, resulting in noise-dominated distributions where valid solutions were indistinguishable from failures.

Based on the outcomes detailed in Section 6.4.2, it can be concluded that hybrid approaches-such as the branch-and-bound method applied to the physiotherapy clinic schedul-

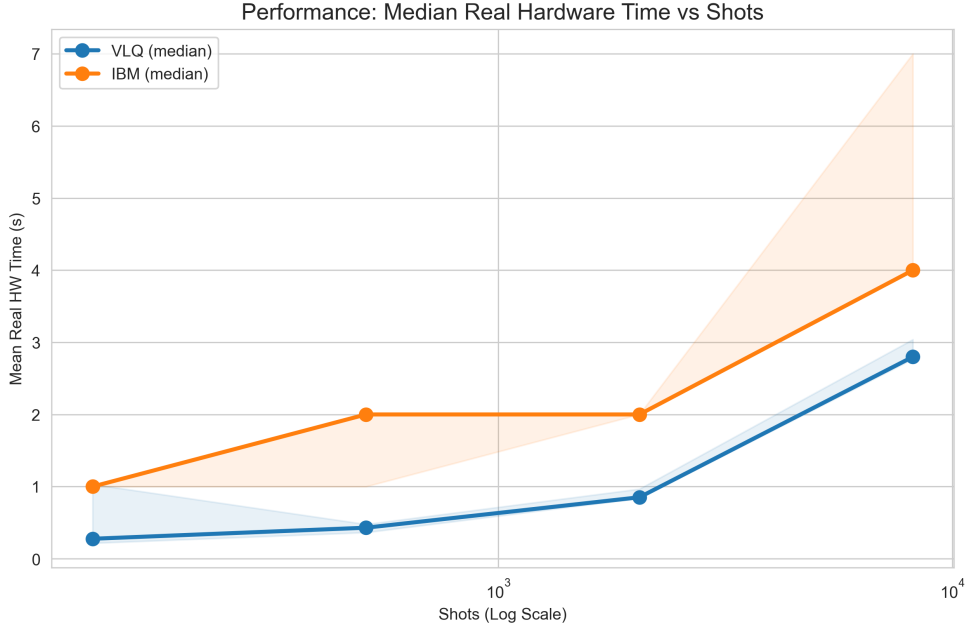


Figure 8.2: Impact of number of shots on hardware execution latency. The results demonstrate how increasing the number of shots influences exact execution time on QPU.

ing case study (ECP) in Chapter 5 provide an effective strategy for minimizing problem sizes. This reduction is essential for achieving acceleration on contemporary hardware, which is primarily limited by CX gate errors, as experimentally measured in Section 6.4.

The experimental data indicates that only problem instances with a circuit depth of approximately 250 gates (SAT Instance3) or fewer yield results that are statistically distinguishable from noise. Because the ECP introduces highly interconnected circuits due to the complex constraints within its oracle, it remains a particularly challenging problem for current quantum processors. Consequently, for better practical applicability, the development of lower-fidelity CX gates is paramount; increasing circuit depth capability through error reduction is currently more critical for algorithmic success than simply increasing the total qubit count.

8.3 Key Findings and Lessons Learned

As quantum computing continues to mature, establishing a clear understanding of its current capabilities and practical limitations is essential. In this thesis, we analyzed and implemented quantum solvers for three fundamental NP-complete problems: Exact Set Cover (Section 5.3), the Knapsack Problem (Section 7.5), and the Boolean Satisfiability Problem (Section 7.6). These implementations demonstrate the potential for utilizing quantum speedup in combinatorial and multi-criteria optimization, specifically in scenarios where identifying all optimal solutions is a critical requirement.

Our findings indicate that quantum hardware is not yet at a stage where it can outperform state-of-the-art classical algorithms for large-scale instances. The benchmark data in Figures 8.3, 8.2, and 8.6 support this conclusion: the reported hardware durations represent exact execution time (excluding the significant overhead of the provider’s queuing and processing workflows), and the circuit depth refers to the logical design prior to tran-

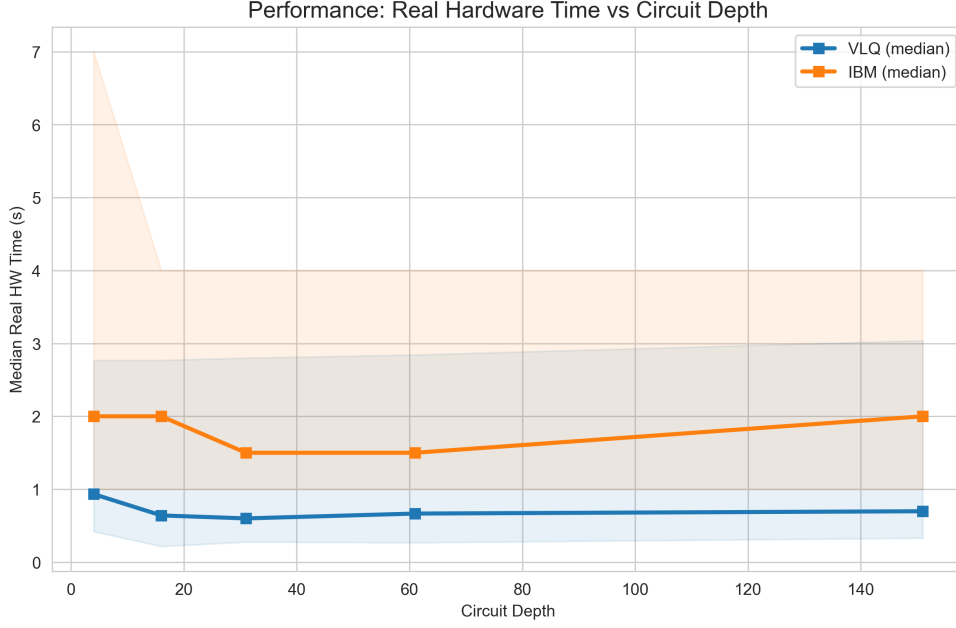


Figure 8.3: Impact of logical circuit depth on hardware execution latency. The results demonstrate how increasing the number of sequential gate operations, the critical path, influences exact execution time on QPU. The depth is before the circuit is transpiled to specific HW.

spilation. To illustrate, the smallest ECP instance (Experiment0) has a depth of 192 and was executed with 2000 shots. From the heatmaps, one can estimate its execution time at approximately 1 to 2 seconds (1 second for VLQ, 2 seconds for IBM Marrakesh), as shown in Table 7.5. However, we are approaching a transition point through hybrid methodologies. As demonstrated in Chapter 5, strategies such as the Branch-and-Bound algorithm can effectively prune the search space, reducing problem instances to a size that fits within current quantum register constraints.

A primary takeaway from our experimental evaluation is the severe impact of circuit depth on output fidelity (see Figure 6.7). For multi-constraint and real-world problem sizes, current hardware lacks the necessary qubit volume and gate fidelity to maintain a distinguishable signal. The high error rates associated with two-qubit (CX gate) gates represent the most significant bottleneck; consequently, the priority for hardware advancement must be the development of lower-noise control gates rather than simply increasing qubit counts. Furthermore, as shown in Table 4.1, most real-world instances do not yet fit into available quantum memory without significant decomposition.

Despite these hardware-driven limitations, this work serves as a successful proof of concept. It has been demonstrated that the Grover-based search architecture, when integrated into a hybrid framework, possesses the structural potential to accelerate the resolution of real-world problems once hardware coherence times and gate fidelities reach a sufficient threshold.

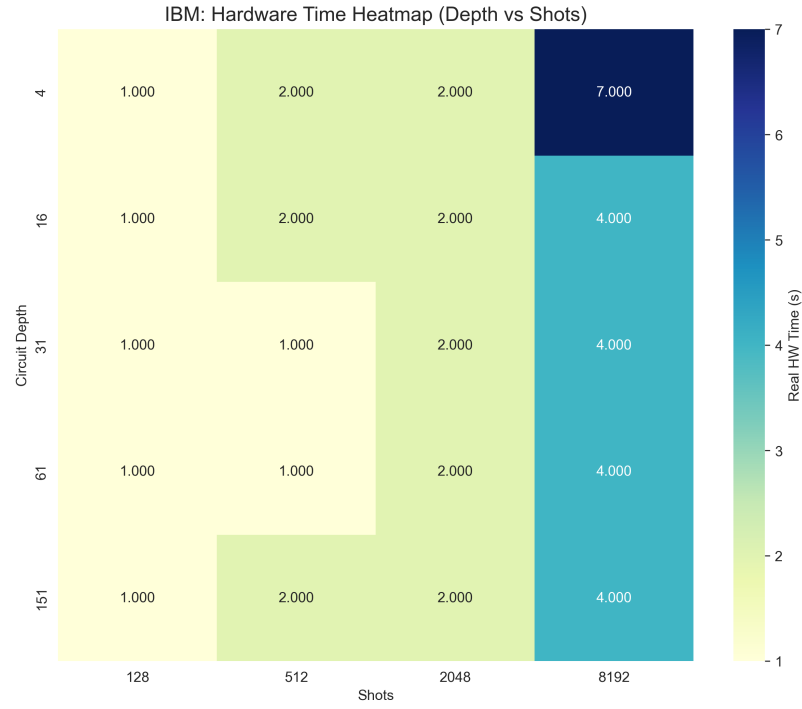


Figure 8.4: IBM Marrakesh: Execution scaling analysis.



Figure 8.5: IQM Star24 (VLQ): Execution scaling analysis.

Figure 8.6: Comparative heatmaps showing strictly hardware execution time as a function of circuit depth and shot count. This demonstrates how processing time scales on IBM Marrakesh and IQM Star24 across various workload complexities.

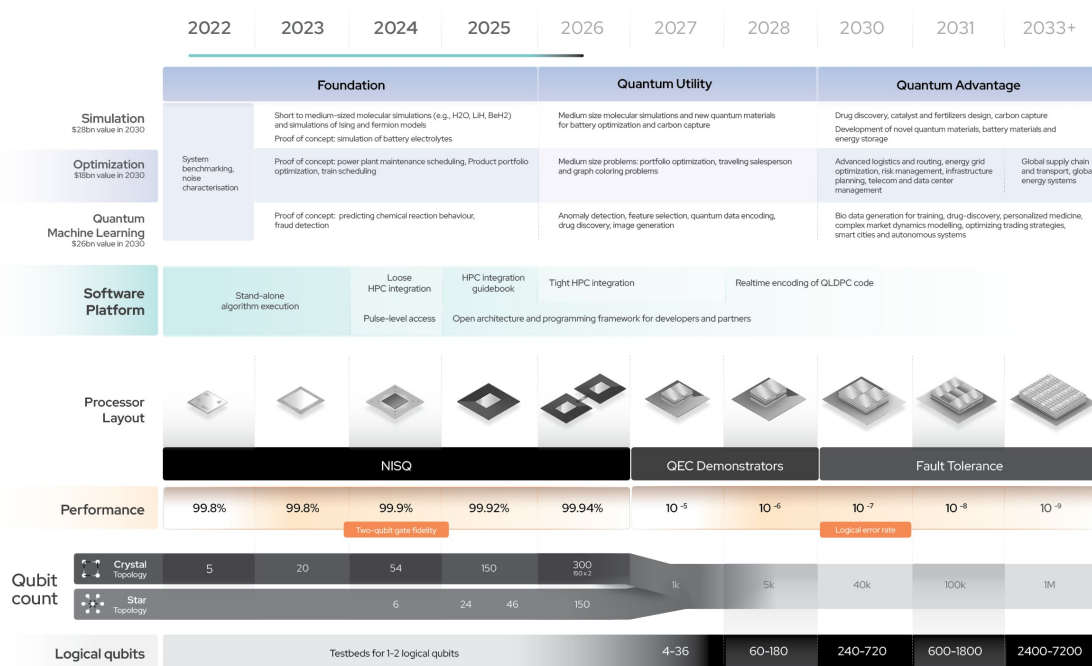


Figure 8.7: IQM Quantum Computers technology roadmap (2023–2030+) showing NISQ optimization, QEC demonstrators, and fault-tolerant scaling phases. *Source:* <https://meetiqm.com/technology/roadmap/>

Future research will focus on two primary avenues. First, we aim to implement advanced circuit manipulation methods, such as circuit cutting and circuit shallowing techniques, to decompose complex oracles into manageable sub-circuits without a loss of logical information. Second, while this study focused on superconducting architectures, future work will expand the evaluation to alternative quantum modalities. Specifically, we intend to benchmark these hybrid solvers on photonic and trapped-ion quantum computers, which are becoming increasingly accessible through the EuroHPC Joint Undertaking (EuroHPC JU) [14]. Testing across these diverse hardware platforms will provide critical insights into how different physical realizations of qubits—each with unique connectivity and error profiles—impact the performance and scalability of combinatorial optimization solvers. Furthermore, the anticipated delivery of more powerful hardware, as outlined in the IQM roadmap (Figure 8.7), will be a vital catalyst for this transition. The convergence of these next-generation architectures with the optimization strategies developed in this work will be fundamental to fully realizing the potential of quantum-integrated high-performance computing.

Bibliography

- [1] ABUGHANEM, M. IBM Quantum Computers: Evolution, performance, and future directions. *ArXiv preprint*, 2024, arXiv:2410.00916. Available at: <https://arxiv.org/abs/2410.00916>.
- [2] ACHARYA, R.; ABANIN, D. A.; AGHABABAIE BENI, L.; ALEINER, I.; ANDERSEN, T. I. et al. Quantum error correction below the surface code threshold. *Nature*. Springer Science and Business Media LLC, december 2024, vol. 638, no. 8052, p. 920–926. ISSN 1476-4687. Available at: <http://dx.doi.org/10.1038/s41586-024-08449-y>.
- [3] AGHAEI RAD, H.; AINSWORTH, T.; ALEXANDER, R. N. et al. Scaling and networking a modular photonic quantum computer. *Nature*, January 2025, vol. 638, no. 8052, p. 912–919. Available at: <https://doi.org/10.1038/s41586-024-08406-9>. First scalable photonic quantum computer prototype with 35 chips and 12 physical qubits.
- [4] AUBERT, C. and CRISTESCU, I. Structural Equivalence in Reversible Calculus of Communicating Systems (Abstract). In: *ICE 2020*. 2020. Available at: https://aubert.perso.math.cnrs.fr/recherche/structural_equivalence_in_rccs_abstract.pdf.
- [5] BURD, B. Introduction to Quantum Computing. In: *Quantum Computing Algorithms: Discover how a little math goes a long way*. Packt Publishing, 2023. ISBN 9781804610565.
- [6] CAPRARA, A.; TOTH, P. and FISCHETTI, M. Algorithms for the Set Covering Problem. *Annals of Operations Research*, 2000, vol. 98, p. 353–371.
- [7] CHICANO, F.; LUQUE, G.; DAHI, Z. A. and GIL MERINO, R. Combinatorial optimization with quantum computers. *ArXiv preprint*, March 2024, vol. 2412, p. 15778. Available at: <https://arxiv.org/abs/2412.15778>.
- [8] CHOUCAIR, C. *IQM Star Topology Amplifies All-to-All Qubit Connectivity Through a Central Resonator* The Quantum Insider. April 2025. Available at: <https://thequantuminsider.com/2025/04/30/iqm-star-topology-amplifies-all-to-all-qubit-connectivity-through-a-central-resonator/>. Accessed: 2026-04-28.
- [9] CICERO, A.; MALEKI, M. A.; AZHAR, M. W.; KOCKUM, A. F. and TRANCOSO, P. *Simulation of Quantum Computers: Review and Acceleration Opportunities*. 2025. Available at: <https://arxiv.org/abs/2410.12660>.
- [10] CROSS, A.; JAVADI ABHARI, A.; ALEXANDER, T.; DE BEAUDRAP, N.; BISHOP, L. S. et al. OpenQASM 3: A Broader and Deeper Quantum Assembly Language. *ACM Transactions on Quantum Computing*. Association for Computing Machinery

- (ACM), 2022, vol. 3, no. 3, p. 1–50. ISSN 2643-6817. Available at: <http://dx.doi.org/10.1145/3505636>.
- [11] D WAVE. *Performance gains in the D-Wave Advantage2 system at the 4,400-qubit scale*. Whitepaper 14-1083A-A. May 2025. Available at: https://www.dwavequantum.com/media/wakjcpsf/adv2_4400q_whitepaper-1.pdf.
 - [12] D-WAVE QUANTUM INC.. *D-Wave Documentation*. 2025. Available at: <https://docs.dwavequantum.com/>. Accessed: 2026-01-10.
 - [13] EUROHPC JOINT UNDERTAKING. *Advancing European quantum computing: Signature of the procurement contract for the EuroHPC quantum computer located in Czechia*. 2024. Available at: https://eurohpc-ju.europa.eu/advancing-european-quantum-computing-signature-procurement-contract-eurohpc-quantum-computer-located-2024-09-26_en. Press release, 26 September 2024.
 - [14] EUROHPC JOINT UNDERTAKING. *EuroHPC Joint Undertaking - Quantum Computers* EuroHPC Websitest. 2026. Available at: https://www.eurohpc-ju.europa.eu/eurohpc-quantum-computers/our-quantum-computers_en. Retrieved April 2026.
 - [15] GOOGLE QUANTUM AI AND COLLABORATORS. Quantum error correction below the surface code threshold. *Nature*, December 2024, vol. 638, p. 920–926. Available at: <https://doi.org/10.1038/s41586-024-08449-y>. First demonstration of exponential logical error suppression with surface codes.
 - [16] HARTMUT NEVEN. *Meet Willow, our state-of-the-art quantum chip*. Available at: <https://blog.google/innovation-and-ai/technology/research/google-willow-quantum-chip/>. Accessed: 2026-04-28.
 - [17] IBM QUANTUM. *IBM Quantum delivers on 2022 100x100 performance challenge*. 2024. Available at: <https://www.ibm.com/quantum/blog/qdc-2024>. IBM Quantum Computing Blog, November 2024.
 - [18] IBM QUANTUM. *Processor types*. 2024. Available at: <https://quantum.cloud.ibm.com/docs/en/guides/processor-types>. IBM Quantum Documentation.
 - [19] IBM QUANTUM. *IBM Quantum processors* IBM Research official documentation. 2025. Available at: <https://www.ibm.com/quantum>. Retrieved November 2025.
 - [20] IBM QUANTUM. *Qiskit Documentation*. 2025. Available at: <https://quantum.cloud.ibm.com/docs>. Accessed: 2026-01-10.
 - [21] IBM RESEARCH COMMUNICATIONS. *IBM unveils 400 qubit-plus quantum processor and next-generation IBM Quantum System Two* IBM Newsroom. 9. November 2022. Available at: <https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two>. Announcement of 433-qubit Osprey processor and IBM Quantum System Two modular architecture.

- [22] IONQ INC.. *Forte quantum processor: 30-qubit trapped-ion system* IonQ official product documentation. 2025. Available at: <https://ionq.com/products/forte>. All-to-all connectivity, room temperature operation.
- [23] IQM. *Advantage2 quantum annealer: Technical specifications* IQM Academy - interactive learning materials. Available at: <https://www.iqmacademy.com/learn/qec/>. Retrieved November 2026.
- [24] IQM. *Iqm-qaoa* <https://pypi.org/project/iqm-qaoa/>. 2024.
- [25] IQM QUANTUM COMPUTERS. *Germany launches its first hybrid quantum computer at Leibniz Supercomputing Centre*. 2024. Available at: <https://meetiqm.com/press-releases/germany-launches-its-first-hybrid-quantum-computer-at-leibniz-supercomputing-centre/>. Press release, 18 June 2024.
- [26] IQM QUANTUM COMPUTERS. *IQM Quantum Computers achieves a new benchmark result on 20-qubit quantum computer*. 2024. Available at: <https://meetiqm.com/press-releases/iqm-quantum-computers-achieves-a-new-benchmark-result-on-20-qubit-quantum-computer/>. Press release, 20 February 2024.
- [27] IQM QUANTUM COMPUTERS. *IQM Quantum Computers to deliver Czech Republic's first quantum computer with unique star topology*. 2024. Available at: <https://meetiqm.com/press-releases/iqm-quantum-computers-to-deliver-czech-republics-first-quantum-computer-with-unique-star-topology/>. Press release, 26 September 2024.
- [28] IQM QUANTUM COMPUTERS. *QPU technology*. 2024. Available at: <https://meetiqm.com/technology/qpu/>. Technical documentation.
- [29] IQM QUANTUM COMPUTERS. *Tech Stack — IQM Quantum Tech*. 2025. Available at: <https://meetiqm.com/technology/tech-stack/>. Accessed: 2026-01-10.
- [30] IT4INNOVATIONS. *Quantum-as-a-Service*. Available at: <https://github.com/It4innovations/quantum-as-a-service/>. Retrieved April 2026.
- [31] IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER. *Europe takes a quantum leap: LUMI-Q consortium signs contract to establish quantum computer in the Czech Republic*. 2024. Available at: <https://www.it4i.cz/en/about/infoservice/press-releases/europe-takes-a-quantum-leap-lumi-q-consortium-signs-contract-to-establish-quantum-computer-in-the-czech-republic>. Press release, 25 September 2024.
- [32] IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER. *LUMI-Q Consortium unveils the VLQ quantum computer for the Czech and European science community*. 2025. Available at: <https://www.it4i.cz/en/about/infoservice/news/lumi-q-consortium-unveils-the-vlq-quantum-computer-for-the-czech-and-european-science-community>. Press release, September 2025.
- [33] JAX DEVELOPERS. *JAX: Composable Transformations of Python + NumPy Programs*. 2025. Available at: <https://github.com/jax-ml/jax>. Accessed: 2026-01-10.

- [34] JIANG, J. R. and WANG, Y. J. Quantum Circuit Based on Grover’s Algorithm to Solve Exact Cover Problem. In: *2023 VTS Asia Pacific Wireless Communications Symposium (APWCS)*. Tainan City, Taiwan: [b.n.], 2023, p. 1–5.
- [35] KAISER, S. C. and GRANADE, C. *Learn Quantum Computing with Python and Q: A Hands-On Approach*. Manning Publications, 2021. ISBN 9781617296130. Available at: https://books.google.cz/books/about/Learn_Quantum_Computing_with_Python_and.html. Accessed: 2026-01-10.
- [36] KARP, R. M. Reducibility Among Combinatorial Problems. In: MILLER, R. E. and THATCHER, J. W., ed. *Complexity of Computer Computations*. New York: Plenum Press, 1972, p. 85–103.
- [37] KRAKOW, A. C. C. C. of the AGH University of. *QHyper - quantum related optimizers* QHyper - official documentation. 2026. Available at: <https://qhyper.readthedocs.io/en/latest/>. Retrieved April 2026.
- [38] MANSFIELD, E.; SEEGERER, S.; VESANEN, P.; ECHAVARRIA, J.; METE, B. et al. *First Practical Experiences Integrating Quantum Computers with HPC Resources: A Case Study With a 20-qubit Superconducting Quantum Computer*. 2025. Available at: <https://arxiv.org/abs/2509.12949>.
- [39] MARTINIS, J. M. and OSBORNE, K. *Superconducting qubits and the physics of Josephson junctions*. Les Houches School Lecture Notes. Boulder, CO: National Institute of Standards and Technology, 2004. Available at: <https://web.physics.ucsb.edu/~martinisgroup/classnotes/finland/LesHouchesJunctionPhysics.pdf>. Comprehensive introduction to Josephson junction physics, BCS theory, quasiparticle tunneling, and superconducting qubit design. Originally presented at Les Houches 2004.
- [40] MICROSOFT. *Microsoft Quantum Development Kit*. Available at: <https://github.com/microsoft/qdk>. Accessed: 2026-04-29.
- [41] PRESKILL, J. *Quantum Error Correction* Lecture Notes for Ph219/CS219: Quantum Computation. Available at: <https://theory.caltech.edu/~preskill/ph219/chap7.pdf>. Chapter 7.
- [42] PYTORCH COMMUNITY. *PyTorch Documentation*. 2025. Available at: <https://docs.pytorch.org/docs/main/index.html>. Accessed: 2026-01-10.
- [43] QISKIT AER CONTRIBUTORS. *Qiskit Aer: statevector_state.hpp*. 2024. Available at: https://github.com/Qiskit/qiskit-aer/blob/main/src/simulators/statevector/statevector_state.hpp.
- [44] QRISP. *Qrisp: A Framework for Compilable and Reusable Quantum Software*. 2026. Available at: <https://github.com/eclipse-qrisp/Qrisp>.
- [45] QRISP. *Qrisp simulator: qrisp-simulator.py*. 2026. Available at: <https://github.com/eclipse-qrisp/Qrisp/blob/main/src/qrisp/simulator/simulator.py>.

- [46] QRISP. *Qrisp standard operations: `qrisp-standard_operations.py`*. 2026. Available at: https://github.com/eclipse-qrisp/Qrisp/blob/main/src/qrisp/circuit/standard_operations.py.
- [47] QUANTINUUM LIMITED. *H2-1 quantum computer: 56-qubit trapped-ion system with record gate fidelities* Quantinuum official documentation. 2025. Available at: <https://www.quantinuum.com/products/h-series/h2>. Benchmarking results: JPMorgan Chase and RCS demonstrator (2024).
- [48] RENGIER, M.; VERJAUW, J.; WURZ, N.; HOSSEINKHANI, A.; OCKELOEN KORPPI, C. et al. *A Superconducting Qubit-Resonator Quantum Processor with Effective All-to-All Connectivity*. 2025. Available at: <https://arxiv.org/abs/2503.10903>.
- [49] SANAVIO, C.; TIGNONE, E. and ERCOLESSI, E. Hybrid Classical–Quantum Branch-and-Bound Algorithm for Solving Integer Linear Problems. *Entropy*, 2024, vol. 26, no. 4. ISSN 1099-4300. Available at: <https://www.mdpi.com/1099-4300/26/4/345>.
- [50] SVORE, K.; GELLER, A.; TROYER, M.; AZARIAH, J.; GRANADE, C. et al. Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL. In: *Proceedings of the Real World Domain Specific Languages Workshop 2018*. ACM, February 2018. RWDSL2018. Available at: <http://dx.doi.org/10.1145/3183895.3183901>.
- [51] WINELAND, D. J.; MONROE, C.; ITANO, W. M.; LEIBFRIED, D.; KING, B. E. et al. *Experimental issues in coherent quantum-state manipulation of trapped atomic ions*. 1998. Available at: <https://arxiv.org/abs/quant-ph/9710025>.
- [52] XANADU QUANTUM TECHNOLOGIES INC.. *PennyLane Documentation*. 2025. Available at: <https://docs.pennylane.ai/en/stable/index.html>. Accessed: 2026-01-10.

Table 1: QRISP framework — evaluation of ECP solver performance, implementation approaches, and quantified result certainty. Experiments and implementation details are in Chapter 7; z-score methodology is in Section 7.8.

experiment	problem	backend	solver	filtered mass	p1	p1 state	p2	p2 state	z-score	verdict
Experiment0	ECP	FakeMarrakesh	QAOA	0.025000	0.827692	B	0.087863	A	56.7889	99%
Experiment0	ECP	FakeMarrakesh	Grover (Gate)	0.042000	0.881002	B	0.064196	{A,B}	50.6390	99%
Experiment0	ECP	FakeMarrakesh	Grover (High)	0.255000	0.335570	B	0.333557	A	0.1165	INC
Experiment1	ECP	FakeMarrakesh	QAOA	0.012333	0.322983	{B,C}	0.169423	{A,C,D}	13.7201	99%
Experiment1	ECP	FakeMarrakesh	Grover (Gate)	0.073000	0.080367	{B,C}	0.074973	C	0.6136	INC
Experiment1	ECP	FakeMarrakesh	Grover (High)	0.345500	0.053476	{B,C}	0.135982	C	-7.2079	FAIL
PaperExample	ECP	FakeMarrakesh	QAOA	0.036667	0.024567	{A,E}	0.125260	{B,C,D}	-14.5373	FAIL
PaperExample	ECP	FakeMarrakesh	Grover (Gate)	0.051000	0.025817	{A,C,D}	0.055321	D	-4.6070	FAIL
PaperExample	ECP	FakeMarrakesh	Grover (High)	0.109000	0.021886	{A,C,D}	0.071268	E	-6.9950	FAIL
Experiment4	ECP	FakeMarrakesh	QAOA	0.000000	0.072333	{A,D,E,F}	0.187333	{A,B,D,E}	-13.2510	FAIL
Experiment4	ECP	FakeMarrakesh	Grover (Gate)	0.016500	0.017285	{A,D,E,F}	0.022877	A	-1.2502	FAIL
Experiment0	ECP	Marrakesh	Grover (Gate)	0.592500	0.319018	B	0.363190	A	-1.8809	FAIL
Experiment0	ECP	Marrakesh	Grover (High)	0.248500	0.363939	B	0.324684	A	2.2648	95%
Experiment1	ECP	Marrakesh	Grover (Gate)	0.067000	0.069668	{B,C}	0.087889	{A,C}	-2.0660	FAIL
Experiment1	ECP	Marrakesh	Grover (High)	0.255000	0.059732	{B,C}	0.170470	C	-9.4709	FAIL
PaperExample	ECP	Marrakesh	Grover (Gate)	0.057000	0.024390	{B,D}	0.054613	{A,D}	-4.7647	FAIL
PaperExample	ECP	Marrakesh	Grover (High)	0.047500	0.025197	{A,E}	0.060367	B	-5.3638	FAIL
Experiment4	ECP	Marrakesh	Grover (Gate)	0.025000	0.023590	{A,D,E,F}	0.032308	D	-1.6515	FAIL
Experiment4	ECP	Marrakesh	Grover (High)	0.026500	0.012327	{A,D,E,F}	0.029276	A	-3.7054	FAIL
Experiment0	ECP	AerSimulator	QAOA	0.007333	0.783747	B	0.163533	{A,B}	47.9318	99%
Experiment0	ECP	AerSimulator	Grover (Gate)	0.000000	1.000000	B	0.000000	None	63.2456	99%
Experiment0	ECP	AerSimulator	Grover (High)	0.229000	0.344358	B	0.338521	{A,B}	0.3418	INC
Experiment1	ECP	AerSimulator	QAOA	0.001333	0.571762	{B,C}	0.228304	{A,C,D}	27.1342	99%
Experiment1	ECP	AerSimulator	Grover (Gate)	0.039500	0.473191	{B,C}	0.043207	C	30.4495	99%
Experiment1	ECP	AerSimulator	Grover (High)	0.930500	0.805755	{B,C}	0.035971	{A,B,C}	12.9987	99%
PaperExample	ECP	AerSimulator	QAOA	0.009667	0.039717	{A,C,D}	0.126893	{B,C,D}	-12.1566	FAIL

Continued on next page

Table 1: (continued) QRISP framework — evaluation of ECP solver performance, implementation approaches, and quantified result certainty.

experiment	problem	backend	solver	filtered mass	p1	p1 state	p2	p2 state	z-score	verdict
PaperExample	ECP	AerSimulator	Grover (Gate)	0.000000	0.322000	{B,D}	0.000500	{B,C}	27.6449	99%
PaperExample	ECP	AerSimulator	Grover (High)	0.952000	0.177083	{A,C,D}	0.041667	{A,B,C,D,E}	3.0060	99%
Experiment4	ECP	AerSimulator	QAOA	0.000000	0.013333	{A,D,E,F}	0.317000	{A,B,D,E}	-31.6724	FAIL
Experiment4	ECP	AerSimulator	Grover (Gate)	0.008500	0.359052	{A,D,E,F}	0.016641	{B,C}	27.6040	99%
Experiment4	ECP	AerSimulator	Grover (High)	0.986500	0.666667	{A,D,E,F}	0.111111	{A,C,D,F}	4.1089	99%
Experiment0	ECP	Qrisp Simulator	QAOA	0.001953	0.523483	B	0.434442	A,B	5.6308	99%
Experiment0	ECP	Qrisp Simulator	Grover (Gate)	0.000000	1.000000	B	0.000000	None	63.2456	99%
Experiment0	ECP	Qrisp Simulator	Grover (High)	0.265625	0.347074	B	0.345745	A,B	0.0757	INC
Experiment1	ECP	Qrisp Simulator	QAOA	0.000000	0.668945	B,C	0.107422	A,C	36.4367	99%
Experiment1	ECP	Qrisp Simulator	Grover (Gate)	0.052734	0.485567	B,C	0.042268	B	30.9510	99%
Experiment1	ECP	Qrisp Simulator	Grover (High)	0.868164	0.540741	B,C	0.059259	C,D	12.0485	99%
PaperExample	ECP	Qrisp Simulator	QAOA	0.000977	0.065494	{A,E}	0.109482	{B,E}	-4.9207	FAIL
PaperExample	ECP	Qrisp Simulator	Grover (Gate)	0.000000	0.323242	{A,E}	0.000000	None	27.7689	99%
PaperExample	ECP	Qrisp Simulator	Grover (High)	0.894531	0.018519	{A,C,D}	0.074074	A	-2.7092	FAIL
Experiment4	ECP	Qrisp Simulator	QAOA	0.000000	0.016602	{A,D,E,F}	0.182617	{A,B,E,F}	-17.5301	FAIL
Experiment4	ECP	Qrisp Simulator	Grover (Gate)	0.008789	0.359606	{A,D,E,F}	0.018719	{A,B,D,E,F}	27.4007	99%
Experiment4	ECP	Qrisp Simulator	Grover (High)	0.974609	0.538462	{A,D,E,F}	0.076923	{B,D,E,F}	5.0000	99%

Abbreviations: ECP = Exact Cover Problem. Backends: FakeMarrakesh = Qiskit AerSimulator using IBM's Marrakesh noisy model;

Marrakesh = IBM quantum device (Heron r2 chip). Verdicts: „99%“ = z-score > 2.58; „95%“ = z-score > 1.96;

„INC“ = Inconclusive; „FAIL“ = unable to recognize correct solutions.

Table 2: QRISP framework — evaluation of SAT and Knapsack solver performance, implementation approaches, and quantified result certainty. Experiments and implementation details are in Chapter 7; z-score methodology is in Section 7.8.

experiment	problem	backend	solver	filtered mass	p1	p1 state	p2	p2 state	z-score	verdict
Experiment1	KNAP	FakeMarrakesh	GROVER	0.000000	0.090000	100	0.161000	0	-6.7773	FAIL
Experiment2	KNAP	FakeMarrakesh	GROVER	0.000000	0.023333	10000	0.045000	101	-3.7717	FAIL
Experiment3	KNAP	FakeMarrakesh	GROVER	0.000000	0.000000	1000	0.039167	11001	-8.9386	FAIL
Experiment1	KNAP	Marrakesh	GROVER	0.000000	0.097000	110	0.161000	0	-6.0378	FAIL
Experiment2	KNAP	Marrakesh	GROVER	0.000000	0.028333	1000	0.050000	10	-3.5319	FAIL
Experiment3	KNAP	Marrakesh	GROVER	0.000000	0.000000	1000	0.063333	0	-11.4372	FAIL
Experiment1	KNAP	AerSimulator	GROVER	0.000000	0.268000	100	0.039000	11	20.0894	99%
Experiment2	KNAP	AerSimulator	GROVER	0.000000	0.185833	1000	0.014167	11110	18.0953	99%
Experiment3	KNAP	AerSimulator	GROVER	0.000000	0.000000	1000	0.060000	11000	-11.1226	FAIL
Experiment1	KNAP	Qrisp Simulator	GROVER	0.000000	0.258000	010	0.043000	000	19.0147	99%
Experiment2	KNAP	Qrisp Simulator	GROVER	0.000000	0.175833	10000	0.015000	01011	17.3117	99%
Experiment3	KNAP	Qrisp Simulator	GROVER	0.000000	0.000000	1000	0.054167	11000	-10.5522	FAIL
Experiment1	SAT	FakeMarrakesh	GROVER	0.000000	0.654000	101	0.063000	100	38.9713	99%
Experiment2	SAT	FakeMarrakesh	GROVER	0.000000	0.134000	1000	0.059000	1110	8.0322	99%
Experiment3	SAT	FakeMarrakesh	GROVER	0.000000	0.326000	100	0.073000	0	20.0202	99%
Experiment1	SAT	Marrakesh	GROVER	0.000000	0.425000	101	0.118000	111	21.8292	99%
Experiment2	SAT	Marrakesh	GROVER	0.000000	0.062000	1000	0.094000	0	-3.7734	FAIL
Experiment3	SAT	Marrakesh	GROVER	0.000000	0.203000	100	0.153000	0	4.1336	99%
Experiment1	SAT	AerSimulator	GROVER	0.000000	0.778000	101	0.044000	0	47.1756	99%
Experiment2	SAT	AerSimulator	GROVER	0.000000	0.301000	1100	0.008000	0	25.6358	99%
Experiment3	SAT	AerSimulator	GROVER	0.000000	0.490000	100	0.000000	None	36.0280	99%
Experiment1	SAT	Qrisp Simulator	GROVER	0.000000	0.781200	101	0.031200	000	48.2916	99%
Experiment2	SAT	Qrisp Simulator	GROVER	0.000000	0.316400	1100	0.003900	0000	26.9455	99%
Experiment3	SAT	Qrisp Simulator	GROVER	0.000000	0.500000	010	0.000000	None	36.5148	99%

Continued on next page

Table 2: (continued) QRISP framework — evaluation of SAT and Knapsack solver performance.

experiment	problem	backend	solver	filtered mass	p1	p1 state	p2	p2 state	z-score	verdict
Abbreviations: KNAP = Knapsack, SAT = Boolean Satisfiability Problem. Backends: FakeMarrakesh = Qiskit AerSimulator using IBM's Marrakesh noisy model; Marrakesh = IBM quantum device (Heron r2 chip).										
Verdicts: „99%“ = z-score > 2.58; „95%“ = z-score > 1.96; „FAIL“ = unable to recognize correct solutions.										

.1 Preparation of Environment for Experiments and Usage Guide

To successfully replicate the experiments and execute the implemented solvers within the scope of this thesis, it is essential to understand the administrative and technical requirements for accessing the high-performance quantum hardware utilized in the research.

.1.1 Repository Structure

The repository is organized as follows:

```
.  
--- Data - Example data, Measured results, etc.  
--- Literature - Publications, references, manuals, etc.  
--- Sources - Root folder for the source code.  
--- Thesis - LaTeX / MS Word sources of the thesis.  
Readme.md - Project overview and installation summary.
```

.1.2 Hardware Access and Grant Requirements

The primary hardware used for the European-based experiments is the **IQM Star24 (VLQ)** system located at the IT4Innovations National Supercomputing Center. Access to this physical quantum processing unit (QPU) is not public and requires a formalized allocation of computational resources.

- **Computational Project Grants:** To utilize the VLQ, a user must have a computational project granted through **IT4Innovations Open Calls** or **European Open Calls** (such as those provided by the EuroHPC Joint Undertaking). These grants provide the necessary project identifiers required to authenticate against the hardware middleware.
- **Infrastructure Integration:** The VLQ is integrated with the petascale supercomputer **Karolina**, and its usage is managed via an HPC allocation-based model with priority queuing for granted projects.

.1.3 Software Stack and Middleware Installation

Once a computational project is granted, you must configure the specialized middleware designed to bridge high-level frameworks like Qrisp with the IT4Innovations hardware stack.

Installing py4lexis

The `py4lexis` library is required for managing authentication sessions via the LEXIS platform.

- **Source Repository:** Install it by specifying the extra index URL:

```
pip3.11 install --index-url \
https://opencode.it4i.eu/api/v4/projects/107/packages/pypi/simple \
py4lexis
```
- **Purpose:** It provides the `LexisSession` class to retrieve access tokens. Access to the LEXIS platform is available via `MyAccessID`.

Installing QaaS (Quantum-as-a-Service)

The **QaaS** platform, developed at IT4Innovations, provides a unified API for accessing the IQM hardware.

- **Specific Version:** For compatibility, use the v0.2.0 branch:

```
pip3.11 install \
git+https://github.com/It4innovations/quantum-as-a-service.git@v0.2.0
```

- **Requirements:** Ensure you also install the dependencies listed in `Sources/requirements_qaas.txt`.

.1.4 Environment Configuration and Authentication

To maintain a secure and reproducible environment, sensitive credentials should be managed through environment variables.

- **IBM API Token:** For experiments involving the **IBM Marrakesh (Heron R2)** backend, obtain your API token from the IBM Quantum web application.
- **Environment File (.env):** Create a `.env` file in the root of your repository:

```
export IBM_TOKEN="your_ibm_api_token"
export LEXIS_RESOURCE="fta-00-00"
```
- **Loading the Environment:** Run `source .env` or set them interactively before script execution.

.1.5 Running the Solvers and Experiments

The repository is structured to handle the transition from simulation to real hardware execution seamlessly.

- **Hybrid ECP Solver:** The main solver (`Sources/hybrid_physio_scheduling.py`) dispatches subproblems to a quantum search when they meet specific size thresholds (e.g., 24 qubits for the VLQ).
- **Qrisp Implementations:** Solvers for ECP, Knapsack, and SAT are located in `Sources/qrisp_implementation/`.
- **Hardware Wrappers:** Use scripts in `Sources/qrisp_implementation/qrisp_hw_servers/`, specifically `qaas_server_wrapper.py` for the VLQ and `ibm_server_wrapper.py` for IBM backends.

Execution Note: All Python scripts must be executed from the repository root using the module flag to ensure internal dependencies are loaded correctly:

```
# Standard Format: python3 -m Sources.subdir.filename
python3 -m Sources.qrisp_implementation.ecp_qrisp_example
```

.1.6 Verification and Benchmarking

- **Z-Score Calculation:** Use `Sources/z_score_calculator.py`. A Z-score > 2.58 indicates a 99% confidence level.
- **Depth Benchmarking:**
`Sources/quantum_algorithm_error_investigation/
q_algorithm_depth_benchmark/depth_benchmark_vlq-marrakesh.py`.
- **Latency Analysis:**
`Sources/benchmark_shots_depth_time/
visualize_benchmark_shots_depth_time.py`.

.1.7 Data Directory and Experimental Logs

The Data directory serves as the central repository for raw outputs, statistical analyses, and visualizations.

Hardware Benchmarking and Latency Plots

Located in `Data/benchmark_depth_shots_plots`. Includes heatmaps for IBM Marrakesh (`5_heatmap_ibm_hw_time.png`) and IQM Star24 (`5_heatmap_vlq_hw_time.png`), as well as the raw CSV datasets.

Qrisp Experimental Results

Located in `Data/qrisp_experiments`, containing logs for ECP, Knapsack, and SAT across AerSimulator, FakeMarrakesh, and physical hardware.

- **Formatted Results:** Cleaned versions are in the `formatted_results` sub-folder.
- **SAT-Knapsack Study:** Data on the exponential growth of clause counts is in `sat-knapsack-study-case`.

Statistical Summaries and Z-Score Verdicts

The file `qrisp_z_score_summary.csv` provides the final evaluation for every experiment, including verdicts like **99%**, **95%**, or **FAIL**.

Resource and Application Prototypes

- **Memory Demands:** `hw_result_memory_demands.png` visualizes sampling requirements for Model A vs. Model B.
- **Scheduling Examples:**
`Hybrid_application_prototype_output/scheduling_example1.txt` shows the final physiotherapy clinic schedule.

Quantum Algorithm Depth Benchmark Data

Located in `Data/q_algorithm_depth_benchmark_data`.

- **Raw Results:** .npz files for IBM Marrakesh, IT4I VLQ, and simulators.
- **Comparative Visualizations:**
Diagrams showing state decay (`certainness_vs_depth_all_backends.png`).
- **Transpilation:** Images in `transpiled_circuits/` show topological overhead, such as **MOVE-CZ-MOVE** sequences for Star24.