Università di Napoli "Federico II"

Università di Camerino

Consiglio Nazionale delle Ricerche

Ph.D. in Quantum Technologies
36° Cycle

Quantum Computational Intelligence

SSD: INF/01

Dissertation by

Roberto Schiattarella

Supervisor

Prof. Giovanni Acampora

Naples, a.y. 2023/2024

To my family.

Contents

List of Publications

- Giovanni Acampora, **Roberto Schiattarella** and, Autilia Vitiello. "On the Implementation of Fuzzy Inference Engines on Quantum Computers". IEEE Transactions on Fuzzy Systems, vol. 31, no. 5, pp. 1419-1433, May 2023. doi: `10.1109/TFUZZ.2022.3202348`.
- Giovanni Acampora, **Roberto Schiattarella**, and Autilia Vitiello. "Using quantum amplitude amplification in genetic algorithms". Expert Systems with Applications, page 118203, 2022. doi: `10.1016/j.eswa.2022.118203`
- Giovanni Acampora, Michele Grossi, Micheal Schenk, and **Roberto Schiattarella**. "Quantum Fuzzy Inference Engine for Particle Accelerators Control". Submitted to IEEE Transactions on Quantum Engineering.
- Halima G. Ahmad, **Roberto Schiattarella**, Pasquale Mastrovito, Angela Chiatto, Anna Levochkina, Martina Esposito, Domenico Montemurro, Giovanni P. Pepe, Alessandro Bruno, Francesco Tafuri, Autilia Vitiello, Giovanni Acampora, Davide Massarotti. "Mitigating Errors on Superconducting Quantum Processors through Fuzzy Clustering". Submitted to Advanced Quantum Technologies
- Francesco Di Colandrea, Lorenzo Amato, **Roberto Schiattarella**, Alexandre Dauphin, and Filippo Cardano. "Retrieving space-dependent polarization transformations via near-optimal quantum process tomography". Opt. Express 31, 31698-31717, 2023. doi: `10.1364/OE.491518`

- Giovanni Acampora, Ferdinando Di Martino, Alfredo Massa, **Roberto Schiattarella**, and Autilia Vitiello. "D-nisq: A reference model for distributed noisy intermediate-scale quantum computers". Information Fusion, pages 16-28. Elsevier, 2022. doi: `10.1016/j.inffus.2022.08.003`

- Giovanni Acampora, **Roberto Schiattarella**, and Alfredo Troiano. "A dataset for quantum circuit mapping". Data in Brief, page 107526. Elsevier, 2021. doi: `10.1016/j.dib.2021.107526`

- Giovanni Acampora and **Roberto Schiattarella**. "Deep neural networks for quantum circuit mapping". Neural Computing and Applications, pages 121. Springer, 2021. doi: `10.1007/s00521-021-06009-3`

- Giovanni Acampora, Amir Pourabdollah, and **Roberto Schiattarella**. "Fuzzy logic on quantum annealers". IEEE Transactions on Fuzzy Systems, pages 11, 2021. doi: `10.1109/TFUZZ.2021.3113561`.

- Amir Pourabdollah, Giovanni Acampora, and **Roberto Schiattarella** . "Fuzzy inference on quantum annealers". In 2023 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2023. doi: `10.1109/FUZZ52849.2023.10309732`

- Giovanni Acampora, Alfredo Massa, **Roberto Schiattarella**, and Autilia Vitiello. "Distributing Fuzzy Inference Engines on Quantum Computers". In 2023 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2023. doi: `10.1109/FUZZ52849.2023.10309786`

- Giovanni Acampora, Michele Grossi, and **Roberto Schiattarella**. "A comparison of quantum computer architectures in running fuzzy inference engines". In 2023 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2023. doi: `10.1109/FUZZ52849.2023.10309673`

- Giovanni Acampora and **Roberto Schiattarella**. "On the Effect of Quantum Noise in Quantum Genetic Algorithms". In 2023 IEEE Congress on Evolutionary Computation (CEC), pages 1-8, doi: 10.1109/CEC53210.2023.10254078.
- Giovanni Acampora, Angela Chiatto, Stefano De Luca, Roberta Di Pace, Alfredo Massa, **Roberto Schiattarella**, and Autilia Vitiello. "Application of Quantum Genetic Algorithms to Network Signal Setting Design". In 2023 IEEE Congress on Evolutionary Computation (CEC), pages 1-8. doi: 10.1109/CEC53210.2023.10254158.
- Giovanni Acampora, Angela Chiatto, Luigi Coraggio, Giovanni Di Gregorio, **Roberto Schiattarella**, and Autilia Vitiello. "Genetic Algorithms for Constructing Effective Nuclear Shell-Model Hamiltonians". In 2023 IEEE Congress on Evolutionary Computation (CEC), pages 1-8, doi: 10.1109/CEC53210.2023.10254090.
- Amir Pourabdollah, Giovanni Acampora, and **Roberto Schiattarella**. "Implementing defuzzification operators on quantum annealers". In 2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pages 16, 2022. doi: 10.1109/FUZZ-IEEE55066.2022.9882576.
- Giovanni Acampora, **Roberto Schiattarella**, and Autilia Vitiello. "Quantum mating operator: A new approach to evolve chromosomes in genetic algorithms". In 2022 IEEE Congress on Evolutionary Computation (CEC), pages 18, 2022. doi: 10.1109/CEC55065.2022.9870425.
- Giovanni Acampora, **Roberto Schiattarella**, and Autilia Vitiello. "Quantum genetic selection: Using a quantum computer to select individuals in genetic algorithms". Proceedings of the Genetic and Evolutionary Computation Conference Companion New York, NY, USA, 2021. Association for Computing

Machinery. doi: 10.1145/3449726.3459505

- Giovanni Acampora, Ferdinando Di Martino, **Roberto Schiattarella**, and Autilia Vitiello. "Measuring distance between quantum states by fuzzy similarity operators". In 2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pages 16, 2021. doi: 10.1109/FUZZ45933.2021.9494403.

Introduction

Artificial Intelligence (AI) is becoming increasingly important for the advancement of humankind, with the potential to transform aspects of our lives ranging from healthcare and education to transportation and entertainment, among others. Many of these transformations are already before our eyes and we are continually drawing benefit from them in our daily lives. However, the development of AI turns out to be increasingly complex, because increasingly complex are the scenarios in which these techniques must be applied. Indeed, these are increasingly characterized by high numbers of variables to be considered, large amounts of data to be handled, and increasingly short response times required by the algorithms. An AI field that responds well to these demands is Computational Intelligence (CI), which focuses on developing algorithms and models that enable computers to perform tasks that would typically require human-like intelligence. Indeed, like human intelligence, CI seeks to develop systems that can autonomously adapt to changing environments, learn from experience, and make decisions based on incomplete or uncertain information. CI encompasses a diverse range of techniques, including machine learning, neural networks, fuzzy logic, evolutionary computation, and swarm intelligence, among others. Today, intelligent systems based on these theories are used in a wide range of applications, including robotics, data analysis, decision-making, optimization, and pattern recognition.

The great success of these algorithms is closely related to the great development of computers from the 1970s to the present. Indeed, as predicted back in 1965 by Gordon

Moore, co-founder of Intel, the number of transistors on a microchip would double approximately every two years, leading to a corresponding increase in computing power. Turned out to be true, this law has been a driving force behind the rapid advancement of computing technology over the past few decades. However, as the size of transistors approaches the limits of what is 'physically possible', some experts believe that Moore's law may no longer hold true in the future. And if Moore's law no longer applies, it would no longer be possible to increase the computational power of each individual processor and consequently, the increasingly complex artificial intelligence algorithms can no longer be efficiently carried out. Considering this scenario, one might think that it will no longer be possible to further develop the above CI algorithms because the hardware used for their computation will not be proficient enough to run them.

However, it was realized that if it is not physically possible to further miniaturize the transistors in the processors, what needs to be changed is the kind of physics we need to look at. In the world of the infinitely small, in fact, quantum mechanics applies, and it was on this theory that Richard Feynman in 1982 proposed to build a new type of calculator, a quantum computer precisely, that would be able to perform computations unattainable for any classical processor. In recent years these calculators proposed by Feynman are becoming a reality thanks to large investments by major companies and research centers. Although these devices are still in a primordial state and several problems limit their current applicability, such as a limited number of qubits and high error rates affecting the computation, computer scientists and physicists are developing new algorithms based on this innovative computing paradigm to solve problems more efficiently than equivalent classical algorithms. One field that can particularly benefit from quantum computing is indeed the field of CI.

To pave the way in that direction, this thesis aims to investigate this new research scenario, in which the CI and QC coexist and support each other, that it is possible to denote as ***'Quantum Computational Intelligence'***. On the one hand, the proposed work introduces innovative quantum CI algorithms in the context of optimization and reasoning, and on the other hand, it introduces CI-based algorithms able to improve the current status of quantum computers.

In the former research line, Quantum Genetic Algorithms (QGAs) and Quantum Fuzzy Inference Engines (QFIEs) are proposed. In detail, the QGAs introduced in this thesis are composed of quantum operators able to be run on actual quantum machines that can exploit quantum noise to enhance the exploration capability of these classical optimization algorithms. The QIFEs developed in this work are the very first fuzzy engines able to be run on quantum devices, both annealers and digital quantum computers, and in this latter case, the proposed QFIE is exponentially faster than a classical fuzzy inference engine in computing fuzzy rules, opening up new application scenarios for these linguistic rule-based inference systems.

In the latter research line, where CI algorithms are used to improve the current status of quantum computers, this thesis proposes an innovative compiling method for quantum algorithms based on deep neural networks. This method aims to solve the important issue of mapping quantum algorithms to quantum processors, which is a crucial point in the current era of quantum computers.

In order to fully describe the dichotomic view of the proposed thesis, it is structured as follows: an introduction about the basic aspects of QC and CI is provided in Chapter 1; in Chapter 2 the quantum-enhanced CI algorithms proposed in this thesis will be shown; in Chapter 3 it will be discussed in general how CI can support the current status of quantum computers and the neural network-based mapping strategy will be analyzed.

Chapter 1

Preliminars

This chapter introduces the basic concepts of Quantum Computing (QC) and Computational Intelligence (CI). In detail, Section 1.1 discusses the innovative quantum computational model, highlighting the two most popular quantum computing paradigms to date, namely the quantum circuit-based and the quantum annealing-based computing paradigms. On the other hand, Section 1.2 aims to briefly describe the three main blocks composing the CI research area, such as Fuzzy Logic, Evolutionary Algorithms, and Neural Networks.

1.1 Quantum Computing

Quantum computing can be described as the exploration of how the characteristics of quantum systems, such as superposition, entanglement, and interference, can be leveraged to enhance specific computational tasks [130, 57]. These attributes are not evident in our everyday macroscopic world, and while they exist at the fundamental level within our computing devices, they are not actively utilized in the conventional computing models we employ for developing microprocessors and designing algorithms. Consequently, quantum computers exhibit a drastically distinct behavior compared to classical computers, enabling the possibility to develop better algorithms than those that can be computed on traditional computing devices. For instance, the most famous problem for which quantum algorithms offer a huge advantage over classical methods is finding prime factors of big integers. Indeed, this problem is addressed by the groundbreaking quantum algorithm proposed by Peter Shor [159] with a computational complexity of $O((\log n)^2(\log\log n)(\log\log\log n))$, where n is the number to factorize, whereas the computational complexity of the best classical algorithm for integer factorization is $O(\exp\Big(c(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}\Big))$, where c is a constant. Shor's algorithm is not the only known quantum speed-up example, several other fundamental algorithms have been discovered over the years such as Grover's algorithm [80] and HHL algorithm (Harrow-Hassidim-Lloyd) [82] among the (few) others. In detail, Grover's algorithm is a quantum algorithm for searching an unsorted database with N entries in $O(\sqrt{N})$ time and using $O(\log N)$ storage space, whereas the complexity of the best classical algorithm that performs the same operation in $O(N)$ steps. On the other hand, HHL algorithm is used to find a scalar measurement on the solution vector of a sparse linear system with a low condition number k with a runtime of $O(\log Nk^2)$, where N is the number of variables in the linear system, whereas the fastest classical algorithm is exponentially slower running in $O(Nk)$.

Wonderful as the properties of these quantum algorithms are, they require quantum computers that are fault-tolerant and more powerful than those available today. Indeed, current quantum computers denoted as *Noisy Intermediate-Scale Quantum* (NISQ) computers [144] are characterized by a high level of noise and a low number of qubits available for computation and thus a practical quantum advantage for real applications remains something still far off. Only in 2019 a team of researchers from Google in a landmark paper published in the prestigious Nature Journal [29] reported that a quantum computer had solved, in just a few minutes, a properly designed problem that would have taken the most powerful classical supercomputer in the world thousands of years.

To bridge the gap there is for practical applications of quantum computing, researchers are studying several paradigms of quantum computation, including *quantum Turing machines*, *measurement-based quantum computing*, *adiabatic quantum computing* or the *quantum circuit model.* All of them are equivalent in power, but the most popular one is for sure the *quantum circuit model.* In the following section, this model will be discussed since it is the computing model used to develop most of the algorithms proposed in this thesis work. However, the remaining part of the algorithms is developed by using the adiabatic quantum computing paradigm, which will be discussed instead in Section 1.1.2.

1.1.1 Circuital Model

Every computation has three elements: **data, operations,** and **output**. In the quantum circuit model, these correspond to: **qubits, quantum gates,** and **measurements**. This paradigm uses the qubit as the basic unit to store and manage information. Informally speaking, while a classical binary digit (bit) can be in a classical state either 0 or 1, a qubit can be in a quantum state that is a quantum superposition of 0 and 1, before being measured. In a sense, before performing a quantum measurement, a qubit may have simultaneously the values 0 and 1 and, only when it is measured, does it collapse to one of these two values, corresponding to classical bits. When a qubit is in a superposition of states, it can be said that it has an amplitude associated with each state. Two key aspects are related to the concept of amplitude, two knobs to adjust the configuration of a qubits superposition:

- The *magnitude* associated with each basic state of a qubit (0 or 1), which is related to the probability that a qubit will collapse to the state 0 or 1 after a quantum measurement;
- The *relative phase* between the different states in the qubits superposition. Note that though the probability of measuring a certain state in the superpo-

sition is related only to the magnitude associated with it, the relative phase takes a key role in several quantum algorithms such as *amplitude amplification*, *quantum Fourier transform* or *phase estimation* [93] in modeling desired magnitude distributions and enabling the design of efficient quantum algorithms in different applications domains.

The magnitude and relative phase are values available for being exploited when computing, and it is worth thinking of them as being encoded in a single qubit.

As classical computation does with classical gates, quantum circuit-based computing uses logic gates (*quantum gates*) to change the state of qubits and transform input information into a desired output. These quantum gates are *reversible.* This means that, given an output, the corresponding input can be retrieved by using the reverse of the original operation. Some examples of quantum gates are listed in Table 1.1.

As in classical computation where single bits can be aggregated together to form a classical register, one or more qubits can be aggregated together to form *quantum registers.* A classical register can contain any arbitrary number of bits, say n. A quantum register can hold any superposition of n-qubit quantum states. Therefore, while an n-bit classical register can embody any one of 2^n possible numbers, it can store just one at a time. An n-qubit register, on the other hand, can store any combination of 2^n numbers. Moreover, some of the qubits belonging to a single register or multiple registers can be entangled among them; the entanglement is a non-local property of two or more qubits that allows a set of qubits to express higher correlation than is possible in classical systems.

Formally speaking, a qubit is represented by a unit vector, namely $|\psi\rangle$, of a two-dimensional Hilbert space:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.1}$$

where $\alpha, \beta \in \mathbb{C}$, $|\alpha|^2 + |\beta|^2 = 1$, and $|0\rangle$ and $|1\rangle$ are the basis states of the Hilbert space:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1.2}$$

The value $|\alpha|^2$ is to be interpreted as the probability that, after measuring the qubit, it will be found in state $|0\rangle$, whereas $|\beta|^2$ is to be interpreted as the probability that, after measuring the qubit, it will be found in state $|1\rangle$. The notation adopted to represent a qubit ($|\cdot\rangle$) is named *ket notation* and it is used in quantum mechanics to model quantum state vectors [130].

There is an alternative representation of a qubit enabling its visualization in a three-dimensional reference system. This representation is named *Bloch sphere*[1], and it uses the following representation of a qubit to work, derived from the polar form of complex numbers [96]:

$$\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{1.3}$$

By using this representation only two real numbers, namely θ and ϕ, are necessary to identify a qubit, and consequently, it can be represented as an arrow from the origin to the surface of a three-dimensional sphere of $\mathbb{R}^3$ of radius 1, as shown in Fig. 1.1. According to the general notation presented in (1.1), θ refers to the magnitude associated to each basis state and ϕ is the relative phase between them.

The evolution of a closed quantum system is described by special linear operators, *unitary operators*[2] U which operates on qubits as follows:

$$U|\psi\rangle = U[\alpha|0\rangle + \beta|1\rangle] = \alpha U|0\rangle + \beta U|1\rangle \tag{1.4}$$

[1]Named after the German physicist Felix Bloch.

[2]A linear operator is said to be unitary if $UU^\dagger = U^\dagger U = I$, where $U^\dagger$ denotes the adjoint of the operator U and I the identity matrix.

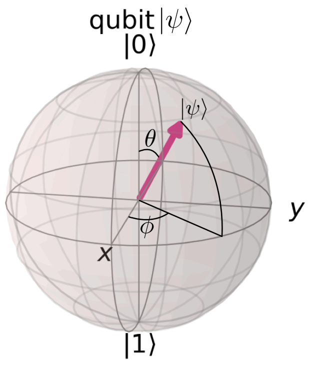


Figure 1.1: Bloch sphere

Therefore, for each of the above quantum gates, there will be a unitary operator capable of formalizing its behavior. In general, the unit operators perform rotations of the vectors corresponding to the quantum states in a two-dimensional Hilbert space. As an example, let us consider the Pauli-X gate acting on a single qubit. It is the quantum equivalent of the NOT gate for classical computers and, for this reason, it is sometimes called bit-flip. The unitary matrix associated with the Pauli-X gate is the one reported in Table 1.1. Let us suppose to have a qubit in a state $|\psi\rangle = 1 \cdot |0\rangle + 0 \cdot |1\rangle$, where $\alpha = 1$ and $\beta = 0$, $|\alpha|^2 = 1$ and $|\beta|^2 = 0$, and compute $|\psi'\rangle = X|\psi\rangle$ (see Fig. 1.2):

$$|\psi'\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1.5}$$

In addition to the X gate, other quantum single-qubit gates can be used to change the state of a qubit. Among these, the R_y and R_z gates are of particular interest, because they allow a simple and direct modification of the aforementioned magnitude and phase knobs. As an example, the R_y gate performs a single-qubit rotation through angle θ radians around the y-axis. The R_y rotation mainly acts on the magnitude knob of the qubit. Thus, this gate modifies the probability that a qubit in a state $|\psi\rangle$ will collapse to 1 or 0, after measuring it.

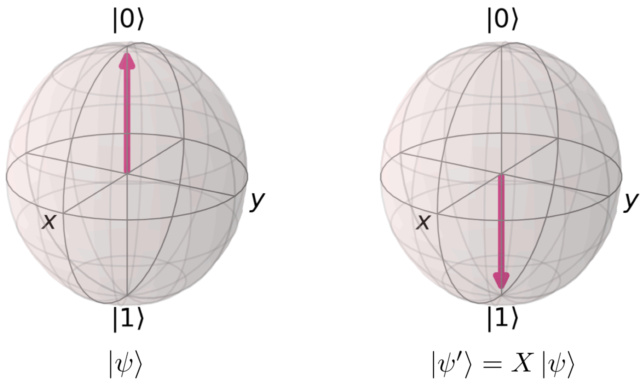


Figure 1.2: The X gate applied to a qubit in state $|0\rangle$: $X\,|0\rangle = |1\rangle$

As useful as single qubits can be, they are much more powerful in groups. Indeed, when a quantum device has access to more than one qubit, it can make use of another powerful quantum phenomenon, entanglement. Formally speaking, a size n quantum register is a quantum system comprising n individual qubits, where each qubit q_i with $i \in \{0, \dots, n-1\}$ is represented by a unit vector of two-dimensional Hilbert space $\mathcal{H}_i$ with $i \in \{0, \dots, n-1\}$. Then, the resulting quantum register is represented by a unit vector of n-dimensional Hilbert space:

$$\mathcal{H} = \mathcal{H}_{n-1} \otimes \mathcal{H}_{n-2} \otimes \ldots \otimes \mathcal{H}_0$$

where the symbol $\otimes$ computes the tensor product of two vector spaces. Therefore, the generic quantum state can be written as:

$$|\psi\rangle = \sum_{k=0}^{2^n-1} \alpha_k\,|k\rangle \qquad \alpha_k \in \mathbb{C} \tag{1.6}$$

where $\sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1$, k is the integer representation of the binary string composed of n bits and $|k\rangle$ refers to the k-th basis vector of $\mathcal{H}$.

When working with a multi-qubit quantum register, the computation is done by both single-qubit gates and multi-qubit gates, i.e., gates that act simultaneously on more than one qubit. An important example of a quantum gate acting on two qubits is the Controlled NOT (CX). Precisely, the CX gate operates on two qubits, a *control* qubit, and a *target* qubit, and it applies the logical quantum NOT operation to the target qubit, if and only if the control qubit is in the state $|1\rangle$. The matrix representation for the unitary operator related to the CX gate is shown in Table 1.1. In Dirac notation, the action of a CX gate on a two-qubit quantum state $|\psi\psi'\rangle$, where ψ is used as control and ψ' as target, is as follows:

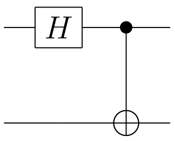
$$\begin{aligned} CX|\psi\psi'\rangle &= CX[\alpha\alpha'|00\rangle + \alpha\beta'|01\rangle + \beta\alpha'|10\rangle + \beta\beta'|11\rangle] \\ &= \alpha\alpha'|00\rangle + \alpha\beta'|01\rangle + \beta\beta'|10\rangle + \beta\alpha'|11\rangle \end{aligned}$$

The role of CX gate is particularly relevant when the control qubit is in a superposition state because, in this case, it enables the quantum entanglement and for this reason, it is called *entangling gate*. Together with superposition, entanglement is another quantum property useful for improving the performance of computing devices. Quantum entanglement often is seen as a key ingredient if quantum computers are to demonstrate an advantage over classical computers. In particular, if a quantum system is not highly entangled it can often be simulated efficiently on a classical computer [125].

For the sake of completeness, it is essential to highlight that any controlled gate can be generalized to its multi-controlled version, where there are more than one control qubits. For instance, the CX gate can be generalized to its multi-controlled version MCX. In the case of two control qubits, the MCX is also known as the *Toffoli gate* (see Table 1.1). Denoting with $|\psi_1\rangle = \sum_{i=0}^{1} \alpha_i^{c_1}|i\rangle$ and $|\psi_2\rangle = \sum_{j=0}^{1} \alpha_j^{c_2}|j\rangle$ the quantum states of the two control qubits, and with $|\psi_t\rangle = \sum_{k=0}^{1} \alpha_k^{t}|k\rangle$ the state of the target qubit, then the action of MCX on the system $|\psi_1\psi_2\psi_t\rangle$ is as follows:

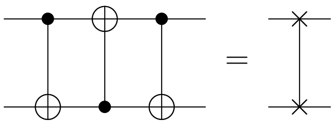
$$\begin{aligned} MCX\,|\psi_1\psi_2\psi_t\rangle &= \sum_{\substack{i,j=0\\ i\wedge j\neq 1}}^{1}\sum_{k=0}^{1}\alpha_i^{c_1}\alpha_j^{c_2}\alpha_k^{t}\,|i,j,k\rangle + \\ &+ \alpha_1^{c_1}\alpha_1^{c_2}\alpha_0^{t}\,|111\rangle + \alpha_1^{c_1}\alpha_1^{c_2}\alpha_1^{t}\,|110\rangle \end{aligned}$$

All the above gates enable the design of quantum algorithms by means of circuits composed of collection gates in the same way classical computation uses gates such as AND, OR, and NOT to develop algorithms on classical computers. An example of a quantum circuit is in (1.7) and it is used to generate the aforementioned Bell state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.



(1.7)

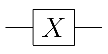
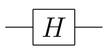
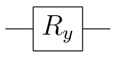
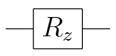
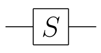
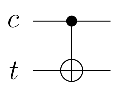
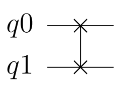
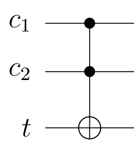
In particular, classical algorithms can be designed by using a subset of classical gates, such as $NAND$ and $FANOUT$, capable of reproducing the behavior of any function of the form: $f : \{0,1\}^n \to \{0,1\}^m$. In the same way, a fundamental theoretical result in quantum computing proves that $\{R_x, R_y, R_z, CNOT\}$ is a universal gate set[3] [130]. Any quantum gate can be unrolled in terms of this set of gates. As an example, the SWAP decomposition in terms of CX is the following [77]:



(1.8)

[3]This set is not the only universal gate sate for unitary matrices. Many others can be used.

Table 1.1: Examples of Quantum Gates

Symbol	**Name**	**Description**	**Matrix**
	Pauli X (also NOT)	Logical bitwise NOT	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
	Hadamard H	The Hadamard gate is a single-qubit operation creating an equal superposition of the two basis states, typically $\|0\rangle$ and $\|1\rangle$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
	Rotation R_y	The R_y gate is one of the Rotation operators. It is used to modify the magnitude knob of qubits.	$\begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$
	Rotation R_z	The R_z performs a rotation of ϕ around the Z-axis direction	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$
	S Gate[4]	This is an R_z-gate with $\phi = \pi/2$. It does a quarter-turn around the Bloch sphere.	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}$
	CX	Controlled NOT: if (c) then NOT(t). This gate is used to enable the quantum entanglement between two qubits	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
	SWAP	The SWAP gate performs the swap of two qubit states q0 and q1	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
	Toffoli CCX	The Toffoli gate: if (c_1 AND c_2) then NOT(t)	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

1.1.1.I Quantum Oracles

In the realm of quantum computing, a remarkable concept known as the *quantum oracle* holds a pivotal role. Quantum oracles are an essential tool that enables quantum algorithms to harness the full potential of quantum mechanics. They act as black boxes, providing an interface for quantum computers to interact with classical data and perform computations that surpass the capabilities of classical computing.

At its core, in theoretical computer science, an oracle is a *black box* capable of producing a solution for any instance of a given computational problem. This problem can be a so-called *decision problem* when a yes-no question of the input values is posed or a so-called *function problem* where the solution is the value $f(x)$ given an input x for a certain function f. The most typical form of an oracle is described by a Boolean function, and, for this reason, it is referred as to *Boolean oracle*. Formally, a Boolean function h is defined as follows:

$$h : \{0,1\}^n \to \{0,1\}. \tag{1.9}$$

In other words, the Boolean function h maps an n-bit string $\mathbf{x} = x_1, x_2, \ldots, x_n$ where $x_i \in \{0,1\}$ to a binary value $h(x) \in \{0,1\}$. In literature, also multi-output Boolean functions where the output is represented by a m-bit string are discussed. Formally, a *multi-output Boolean function* h is defined as follows:

$$h : \{0,1\}^n \to \{0,1\}^m \tag{1.10}$$

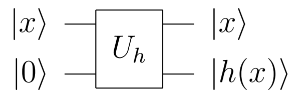
where $m > 1$.

However, in the quantum realm, oracles possess extraordinary properties that allow for the exploitation of quantum phenomena such as superposition and entanglement. These properties make quantum oracles instrumental in a multitude of famous quantum algorithms, paving the way for groundbreaking advancements in various fields. For instance, the aforementioned algorithms proposed by Shor and Grover are based on the concept of a quantum oracle, as are the algorithms of Deutsch and Josza [65], Bernstein-Vazirani [39], and Simon [161] among the others. Also, some of the algorithms proposed in this thesis are based on the same quantum oracle concept. All these quantum algorithms exploit a so-called *quantum oracle* U_h that represents a black box (composed of one or more quantum gates) taking in input $|x\rangle$ and giving in output $|h(x)\rangle$. Formally, considering two quantum registers:

$$U_h |x, y\rangle = |x, y \oplus h(x)\rangle . \tag{1.11}$$

where the output $|h(x)\rangle$ is encoded in the output register by initializing it to the state $|0\rangle$.

Graphically, the action of the quantum oracle U_h is displayed as follows:



In the case of multi-output Boolean functions where $h(x)$ represents an m-bit string, the application of a quantum oracle U_h is defined as follows:

$$U_h |x, \bar{0}\rangle = |x, h(x)\rangle \tag{1.12}$$

where $\bar{0}$ represents a quantum register whose qubits are initialized to the quantum state $|0\rangle$. It is worth noting that the unitarity of a quantum oracle is ensured if and only if every unique input x results in a unique output $h(x)$. To conclude, the advantage of implementing an oracle in quantum computing is that it can be used with superposition quantum states. Formally,

$$U_h \sum_x |x, \bar{0}\rangle = \sum_x |x, h(x)\rangle . \tag{1.13}$$

And that is where the magic happens! By means of quantum oracles, it is possible to evaluate the Boolean function h on all the possible n- dimensional inputs simultaneously, with just one query to the oracle thanks to the quantum superposition of the input state.

For the sake of completeness, it must be said that on quantum computers Boolean functions can be mapped also as *Phase Oracle*. Here, a phase oracle P_h implementing the function h acts as follows:

$$P_h \left|x\right\rangle = (-1)^{h(x)} \left|x\right\rangle \tag{1.14}$$

The equivalence between this phase oracle and the Boolean oracle in (1.11) can be easily shown by considering the output register in (1.11) initialized to the state $\left|-\right\rangle = \frac{\left|0\right\rangle - \left|1\right\rangle}{\sqrt{2}}$:

$$\begin{aligned} & U_h \tfrac{\left|x,0\right\rangle - \left|x,1\right\rangle}{\sqrt{2}} = \\ & \tfrac{\left|x,0\oplus h(x)\right\rangle - \left|x,1\oplus h(x)\right\rangle}{\sqrt{2}} = \\ & \begin{cases} \left|x,-\right\rangle & \text{if } h(x) = 0 \\ -\left|x,-\right\rangle & \text{if } h(x) = 1 \end{cases} \end{aligned} \tag{1.15}$$

which omitting the output register is equivalent to $(-1)^{h(x)} \left|x\right\rangle$.

1.1.1.II Digital Quantum Computers: The State of the Art Quantum Technologies

Digital, or gate-based, quantum computers represent the most widespread approach to quantum computing and have garnered significant attention and investment from numerous companies. These companies are investigating which is the best underlying technology to harness the power of quantum mechanics. These technologies vary in their implementation of qubits, methods of qubit manipulation, and approaches to correct errors. Hereafter it is proposed a brief summary of these promising technologies highlights the pros and cons of each.

1. ***Superconducting Qubits***: One prevalent technology is superconducting qubits, employed by companies such as IBM Quantum, Google, and Rigetti Computing. Superconducting qubits are implemented using superconducting circuits that exhibit quantum behavior at extremely low temperatures. These qubits rely on the controlled flow of electrical current and the manipulation of microwave signals to encode and manipulate quantum information. They offer scalability potential, and the recent advancements have improved qubit coherence times and gate fidelities. However, coherence times and error rates are relatively worse compared to other qubit technologies.

2. ***Trapped Ion Qubits***: Companies like IonQ utilize trapped ion qubits for their quantum computing systems. Trapped ion qubits involve the precise control and manipulation of individual ions trapped in electromagnetic fields. Laser beams are used to initialize, manipulate, and measure the quantum states of ions. Trapped ion qubits are known for their long coherence times and high gate fidelities, making them attractive for performing accurate quantum computations. On the other hand, they have limited scalability.

3. ***Topological Qubits***: Microsoft's approach to digital quantum computing involves the utilization of topological qubits. These qubits rely on the exotic properties of certain two-dimensional materials called topological superconductors. Topological qubits are more robust against environmental noise and errors, as the quantum information is encoded in non-local properties. However, achieving the conditions required for topological qubits is still an active area of research.

4. ***Quantum Dot Qubits***: Quantum dot qubits are employed by companies like Intel in their pursuit of digital quantum computers. Quantum dots are tiny semiconductor structures that can confine a small number of electrons. The quantum information is encoded in the charge or spin of these confined

electrons, and precise control of quantum dot parameters allows for qubit manipulation. Quantum dot qubits have shown potential for scalability and compatibility with existing semiconductor fabrication techniques.

5. ***Photonic Qubits***: Some companies, such as Xanadu, focus on using photonic qubits for quantum computing. Photonic qubits represent quantum information using the properties of light. They can be generated, manipulated, and measured using photonic components such as waveguides and beam splitters. Photonic qubits offer advantages in terms of long-distance communication and the potential for integration with existing optical technologies.

Among these technologies, superconductors and trapped ions are the ones that currently appear to be the most advanced and 'mature', while the others still face important technical complexities. Because of this, the experiments exploiting quantum gates proposed in the following chapters of this thesis work were performed by using mostly superconductive devices (provided by IBM) and sometimes trapped-ion devices (provided by IonQ).

1.1.1.III NISQ Device Limitations

Although in these years quantum devices are growing fast, some hardware limitations still affect the performance they can achieve. One of them is surely related to the short decoherence times of current systems. Quantum decoherence can be viewed as the loss of information from a system into the surrounding environment. Formally speaking, a quantum system is said to be coherent as long as there exists a definite phase relation between different states. However, the interaction of a quantum device with the surrounding environment causes the first to pass from a coherent state to a statistical mixture of states, which no longer contains the quantum information encoded in its states. This loss of information is mainly due to two types of decoherences: transverse relaxation and longitudinal relaxation. The first one is caused by the loss of coherence between the relative phases of the amplitudes of a quantum state. The resulting decoherence time is indicated by T_2. The second one is due to population decay: excited states tend to decay spontaneously at the ground state in a certain typical time T_1. Considering this, to ensure a proper execution of a quantum algorithm it has to be executed in a time that is shorter than the decoherence times of the system. In recent years incredible efforts have been made in this direction. For instance on superconductive devices, execution times for single-qubit gates are on the order of nanoseconds, while typical decoherence times vary from tens to hundreds of microseconds. This makes it possible to manipulate qubits using quantum gates.

A further limitation, indeed, is related to the high error rates in the current quantum devices. The main issue is due to the high error rate of the CNOTs and readout operations. On IBM devices each CNOT gate has a typical error rate order of 10^{-2}, while single-qubit gates have an error rate order of 10^{-4}. The reliability of a quantum circuit is, therefore, mainly influenced by the number of CNOTs in it. Similarly, also the measurement operation has an error rate not negligible, order 10^{-2}. Lastly, in current quantum hardware (at least on the superconductive one), the qubits are not entirely connected to each other, but a quantum processor is characterized by a coupling map that limits the interactions of qubits by means of CNOT gates. This limitation is what is faced by the research proposed in Section 3.1, where a new technique based on classical machine learning is proposed as a workaround to this issue.

1.1.2 Quantum Annealing

The primary emphasis of this chapter was on digital quantum computing, although it has been briefly acknowledged the existence of alternative quantum computing models. Among these models is Adiabatic Quantum Computing (AQC), which was introduced in 2000 by Farhi, Goldstone, Gutmann, and Sipser through a highly influential paper [73]. AQC is polynomially equivalent to other quantum computing models, as proved by Aharonov et al. [23], including the quantum circuit model. This means that anything that is efficiently computable in one of these models is also efficiently computable in AQC, and vice versa. However, in its practical incarnation, AQC is usually implemented in a restricted version called **quantum annealing**. In detail, quantum annealers work by solving QUadratic Binary Optimization (QUBO) or Ising problems. Therefore, to fully describe how a quantum annealer works, QUBO and Ising problems will be discussed in detail, AQC will be analyzed, and finally, these two ingredients will be combined in the analysis of quantum annealers.

1.1.2.I QUBO and Ising Problems

Binary Quadratic Model (BQM) problems are traditionally used in computer science, with applications ranging from machine learning [61] to biology [74]. They are defined as optimization problems formulated as follows: if $\mathcal{Q}$ is an upper-diagonal matrix, which is an NxN upper-triangular matrix of real weights, and X is a vector of binary variables, a BQM problem consists in minimizing the function:

$$f(X) = \sum_{i=1}^{n}(q_i x_i) + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(q_{ij} x_i x_j) \tag{1.16}$$

where q_i and q_{ij} are configurable (linear and quadratic) coefficients. BQM encompasses both Ising and QUBO problems, with the difference that in the former case, the solutions are spin solutions, i.e. $x_i \in \{-1, 1\}$ with $i \in [1, \dots n]$, whereas in the latter case, the solutions are binary solutions, i.e. $x_i \in \{0, 1\}$ with $i \in [1, \dots n]$. Let us focus for a moment on Ising problems: they aim to find the state of minimum energy of an instance of the Ising model, i.e. a mathematical model for the ferromagnetic interaction of particles with spin, usually arranged in a lattice [75]. The particle spins are represented by variables z_j that can take values 1 (spin up) or -1 (spin down). The total energy of the system is given by the Hamiltonian:

$$H = -\sum_{j,k} J_{j,k} z_j z_k - \sum_j h_j z_j \tag{1.17}$$

where the coefficients $J_{j,k}$ represent the interaction between particles j and k and the coefficients h_j represent the influence of an external magnetic field on particle j. From a quantum point of view, solving an instance of the Ising problem means finding the ground state of the Hamiltonian of the system, i.e. :

$$\text{Minimize } \langle\psi| H |\psi\rangle . \tag{1.18}$$

Although sometimes finding the exact ground state is very complex since the problem (1.18) is NP-Hard, a pseudo-optimal solution consists of a state $|\psi\rangle$ such that the amplitude $\alpha_{z_{min}} = \langle z_{min}|\psi\rangle$ is big in absolute value, then it will be high the probability of finding z_{min} by measuring $|\psi\rangle$.

This is what quantum annealers are used for!

Moreover, it is easy to see that each QUBO problem can be reformulated as an Ising problem by performing the transformation $x_j = (1 - z_j)/2$. In such a way, x_j will be 0 when z_j is 1 and 1 when z_j is -1.

At this point, to fully understand how quantum annealers can solve BQM problems, it is required to introduce the theoretical paradigm of computing on which they rely, i.e. the AQC.

1.1.2.II Adiabatic Quantum Computing

In the context of quantum circuits, operations are performed using discrete and sequential steps, employing quantum gates. In contrast, adiabatic quantum computing operates through continuous transformations. In this approach, it is employed a time-varying Hamiltonian[5] as the driving force to alter the state of the qubits according to the time-dependent Schrödinger equation:

$$H(t)\,|\psi(t)\rangle = i\hbar\frac{\partial}{\partial t}\,|\psi(t)\rangle \tag{1.19}$$

where $H(t)$ is the time-dependent Hamiltonian, $|\psi(t)\rangle$ is the state vector of the system, i is the imaginary unit and $\hbar$ is the Planck constant. Now, the idea behind the AQC is that if it is prepared a quantum state that is a ground state of a simple Hamiltonian (let's say H_0), it can be 'very gently' evolved to the ground state of another Hamiltonian (let's say H_1) which codifies the problem to solve. This idea exploits the ***adiabatic theorem*** introduced by Born and Fock in [44]. It says that the process to be adiabatic should be 'slow enough', i.e. its total time should be inversely proportional to the square of the spectral gap, which is the minimum difference in energy between the ground state and the first excited state of the time-dependent Hamiltonian during the whole evolution. From an informal point of view, this means that if there is always a big difference in energy between the ground state and the first excited state, then the process can be speeded up. Otherwise, if the difference is small, with a higher probability it is possible to jump to a higher energy state. More formally, the time-dependent Hamiltonian used in AQC is of the form:

$$H(T) = A(t)H_0 + B(t)H_1 \tag{1.20}$$

where $A(t)$ and $B(t)$ are real-valued functions that accept inputs over the interval $[0, T]$, with T total time of the adiabatic process. In detail, A and B are in such a way that $A(0) = B(T) = 1$ and $A(T) = B(0) = 0$, and therefore $H(0) = H_0$, while $H(T) = H_1$. Considering this, if the adiabatic evolution starts in a ground state of

[5]The Hamiltonian can be thought of as a mathematical object that can describe the energy of the system. In the time-independent Schrödinger equation, the Hamiltonian remains unchanged during the evolution of the quantum system. Instead, in the time-dependent counterpart, energy can vary with time. This is the case, for instance, if it is applied an electromagnetic pulse to the qubits and it is changed its intensity or frequency.

H_0, and the process is effectively adiabatic, it will end up in the ground state of H_1 which is the solution of the problem that must be solved. A common choice for the functions A and B is to set:

$$A(t) = 1 - \tfrac{t}{T}; \quad B(t) = \tfrac{t}{T} \tag{1.21}$$

1.1.2.III Quantum Annealers

Practically, AQC is implemented in a restricted version called **quantum annealing** [57]. It deviates from full AQC in two ways:

- The final Hamiltonian H_1 cannot be chosen completely at will;
- Evolution in real quantum annealers is not guaranteed to be adiabatic.

Let's exploit in detail these two points.

Firstly, H_1 in quantum annealers has to be selected from a certain, restricted class. A typical option is an Ising Hamiltonian of the form reported in (1.17), where the user can select the appropriate $J_{j,k}$ and h_j coefficients. Due to this restriction in the choice of the final Hamiltonian, quantum annealing, unlike AQC is not universal and can be only used to solve a specific type of problem. Moreover, the initial Hamiltonian in the quantum annealing setup is also fixed to $H_0 = -\sum_{j=0}^{n-1} X_j$, where n is the number of qubits, and X_j represents the X matrix acting on the $j-$th qubit. Therefore, the ground state of H_0 is $\otimes_{i=0}^{n-1} |+\rangle$ which is relatively easy to prepare because it is completely unentangled.

Secondly, in quantum annealing, the other significant departure from the AQC approach is that the evolution is no longer guaranteed to be adiabatic. This deviation arises due to two main reasons. Firstly, determining the spectral gap, which represents the minimum difference between the ground state and the first excited state of the time-dependent Hamiltonian, $H(t)$, over the interval $[0, T]$, can be highly challenging. In fact, computing this spectral gap can be even more difficult than finding the desired ground state itself, as demonstrated by Cubitt et al. [59]. Secondly, even if we manage to compute the required time for adiabaticity, it may be impractical or even infeasible to run the system evolution for such a long duration.

Hence, in the context of quantum annealing, the system evolution is conducted for a specific period of time that does not necessarily satisfy the conditions for adiabaticity. Nonetheless, the expectation is to still obtain reasonably accurate approximations of the optimal solution to our problem. Indeed, it is not strictly needed to remain in the ground state $H(t)$, because, in the end, the state will be measured, and it would be enough if the amplitude of an optimal (or sufficiently good) solution in the final state was big enough.

On the bright side, physical quantum devices based on quantum annealing are simpler to construct, making it possible to scale the size of these quantum annealers up to hundreds or even thousands of qubits. In 2011, the Canadian company D-Wave[6] was the first to ever commercialize a quantum annealer. That quantum annealer, called D-Wave One, had 128 qubits, while one of D-Wave's most recent quantum devices, 'Advantage', has more than 5000 qubits.

1.2 Computational Intelligence

Computational Intelligence (CI) is a multidisciplinary field that encompasses various artificial intelligence (AI) techniques aimed at solving complex real-world problems. It draws inspiration from biology, neuroscience, mathematics, and computer science to develop intelligent algorithms and systems capable of learning, adapting, and making decisions in dynamic environments. By combining the power of advanced computing technologies with the principles of human intelligence, CI offers innovative solutions to a wide range of applications [3, 31, 191].

The field encompasses several key techniques, including neural networks, evolu-

[6]https://www.dwavesys.com

tionary computation, and fuzzy logic. These techniques enable machines to learn from data, optimize their performance, and make intelligent decisions. In the following part of this section, these three main blocks of CI will be introduced and discussed in more detail.

1.2.1 Fuzzy Logic

Lotfi Zadeh introduced *fuzzy sets* in 1965 with the main goal of modeling classes of objects that do not have precisely defined criteria of membership [187]. This new vision has made possible the introduction of *fuzzy logic* as an approach to computing that implicitly takes into account the notion of uncertainty in a way that mimics human thinking [186]. Thanks to this capability, fuzzy logic has found great application in the field of automatic control and decision-making, explained by the fact that expert knowledge is easily introduced into fuzzy systems in a faster and easier way than conventional engineering approaches, by means of *fuzzy rules* [188].

In order to explain how a Fuzzy Rule-Based Control System (FRBS) works, it is important to introduce the concepts of fuzzy variables and fuzzy logic. By definition a fuzzy variable X is composed of a set of m linguistic terms $X = \{T_1, T_2, \ldots, T_m\}$, where each linguistic term T_i (with $i = 1, 2, \ldots, m$) is described by a fuzzy set. A fuzzy set S in a universe of discourse U is a collection of ordered pairs, where each pair consists of a generic element x and its grade of membership, as follows: $S = \{(x, \mu_S(x)) | x \in U\}$. Figure 1.3 reports an example of a fuzzy variable defined by four fuzzy sets.

On fuzzy sets are defined operations useful for the implementation of inference engines. Some examples of such operations are union ($\cup$), intersection ($\cap$), and height (hgt). Let us consider two fuzzy sets A and B defined in the same universe of discourse U, their union is a fuzzy set $C_\cup$ computed as follows:

$$C_\cup = A \cup B = \{(x, \mu_{A\cup B}(x)) | x \in U\} \tag{1.22}$$

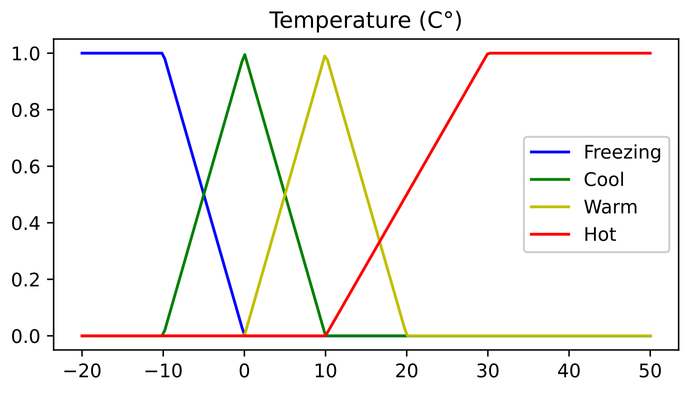


Figure 1.3: Temperature as fuzzy variable defined in the universe $U = [-20°, 50°]$ composed of the set of linguistic terms T=[Freezing, Cool, Warm and, Hot].

where $\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x))$; similarly, their intersection defines a fuzzy set $C_\cap$ computed as follows: $C_\cup$ computed as follows:

$$C_\cap = A \cap B = \{(x, \mu_{A\cap B}(x))|x \in U\} \tag{1.23}$$

where $\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$; finally, the height of the fuzzy set A is defined as below:

$$\text{hgt}(A) = \max_{x\in U} \mu_A(x) \tag{1.24}$$

By using this logic, it is possible to compute the linguistic rules modeling the behavior of a FRBS. In detail, a fuzzy rule is an *if-then* statement where the antecedent and the consequent consist of fuzzy propositions. A fuzzy proposition represents a statement "X is T", where X is a fuzzy variable and T is one of its linguistic terms. The antecedent of a rule can combine various propositions by means of logical connectives, such as *and* and *or*, which are implemented by *t-norm* and *s-norm* operators, respectively [188]. According to the type of the consequent in the rule base, FRBSs are divided into Mamdani's [117] and TSK (Takagi-Sugeno-Kang) [166] FRBSs. The former employs linguistic rules that associate fuzzy sets with output variables, while the latter uses fuzzy rules with crisp output values derived from linear or nonlinear functions. Because in this thesis a quantum version improving the Mamdani approach is proposed, the remaining chapter will focus on the description of such FRBS. For the sake of simplicity, let us consider a MISO (Multiple-Input Single-Output) fuzzy system composed of k rules, n input fuzzy variables $X_1, X_2, \ldots, X_n$, and a single output fuzzy variable Y. All fuzzy variables are composed of m linguistic terms $\{T_i\}_{i=1}^{m}$, and the logical connective used is *and* implemented by the product t-norm. In such a system, a generic fuzzy rule is defined as follows:

$$\text{IF } (X_1 \text{ is } T_{s_1}) \text{ and } (X_2 \text{ is } T_{s_2}) \text{ and } \ldots \text{ and } (X_n \text{ is } T_{s_n}) \text{ THEN } Y \text{ is } T_{s_Y} \tag{1.25}$$

where $s_1, s_2, \ldots, s_n, s_Y \in \{1, 2, \ldots, m\}$.

The fuzzy inference engine of such a system uses a database and a rule base to perform a nonlinear mapping from input and output fuzzy variables, through four sequential steps [92]:

1. the evaluation of each fuzzy proposition belonging to the antecedent part of each rule by considering the system inputs;

2. the computation of the degree of fulfillment of each rule obtained by aggregating the fuzzy propositions of the antecedent part;

3. the computation of the fuzzy output of each rule obtained by applying the implication operator;

4. the computation of the overall fuzzy output obtained by accumulating the fuzzy outputs of individual fuzzy rules.

The first step computes a value α_i^j for the j-th proposition belonging to the i-th rule of the system, with $j = 1, 2, \ldots, n$ and $i = 1, 2, \ldots, k$. This value is computed by intersecting the linguistic T_{s_j} belonging to the j-th proposition of the antecedent part of the i-th fuzzy rule, with a fuzzy set I^j that represents the j-th input of the MISO system, and using the height operator as follows:

$$\alpha_i^j = \mathrm{hgt}(T_{s_j} \cap I^j) \tag{1.26}$$

In the second step, the fuzzy inference engine aggregates fuzzy propositions in the antecedent part of the i-th rule by using the product t-norm operator to compute the degree of fulfillment of the rule:

$$F_i = \prod_{j=1}^{m} \alpha_i^j \tag{1.27}$$

This operation can be carried out also with the min operator.

The third step uses F_i to compute R_i, the output fuzzy set of the i-th rule:

$$\mu_{R_i}(y) = I(F_i, \mu_{Y^i}(y)) \tag{1.28}$$

where $\mu_{Y^i}(y)$ is the membership function of the output variable Y in the i-th rule, and I is an implication function [68]. Common implication functions are implemented by a conjunction operator (t-norm) [92], such as Mamdani's implication [117], based on the minimum operator, and Larsen's implication [104], based on the product operator.

In the fourth and last step, the inference engine computes the whole output fuzzy set Y by using a disjunction operator, implemented by the maximum operator:

$$\mu_Y(y) = \cup_{i=1}^{k} \mu_{R_i}(y) = \max_{k} \mu_{R_i}(y). \tag{1.29}$$

The above description shows that fuzzy inference engines work by dealing only with fuzzy sets. However, the knowledge about the problem at hand is typically expressed by continuous value variables (*crisp* values). Therefore, it is necessary to use so-called *fuzzification* and *defuzzification* interfaces to convert these continuous value variables

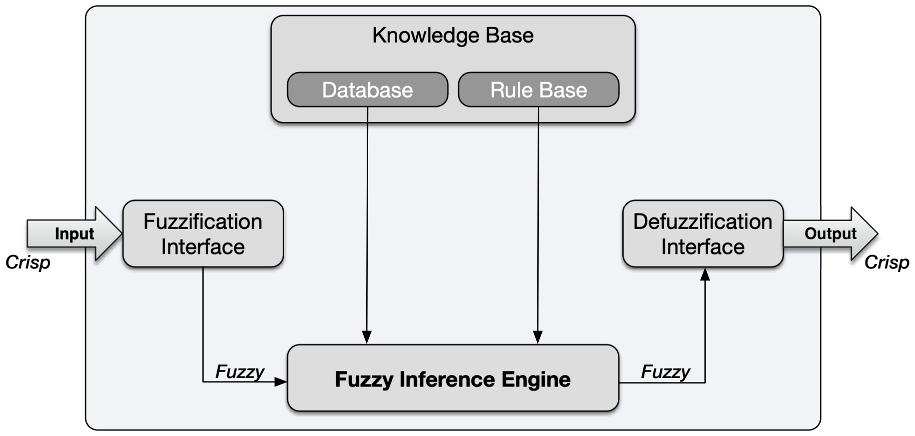


Figure 1.4: Workflow of a FRBS.

into fuzzy sets, and vice versa. [92]. Let us consider a fuzzy variable X defined on a universe of discourse U, and a continuous value input $x' \in U$, then the fuzzification interface computes a singleton fuzzy set T on the universe of discourse U to represent the continuous value input x' as follows:

$$\mu_T(x) = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases} \tag{1.30}$$

Among the defuzzification methods in the literature, the Center of Gravity method (CoG) [101] is widely used. According to this method, the output continuous value $\hat{y}$ of a FRBS is computed as follows:

$$\hat{y} = \frac{\int_{y_l}^{y_u} y \cdot \mu_Y(y)dy}{\int_{y_l}^{y_u} \mu_Y(y)dy} \tag{1.31}$$

where y_l and y_u are respectively the lower and the upper bounds of the universe of discourse of the output variable Y.

Fig. 1.4 summarizes all the steps described above graphically.

1.2.2 Evolutionary Algorithms

Optimization problems are ubiquitous in our lives. They are used, for example, by satellite navigation systems to suggest travel routes, by shipping companies delivering packages to our homes, by financial companies to manage client portfolios, by airline reservation systems to suggest customized and convenient travel proposals, and so on. These problems can be classified into continuous optimization problems and combinatorial optimization problems depending upon the structure and shape of the solutions space. Optimization algorithms are mathematical tools for solving optimization problems, and they can be classified into exact or heuristic algorithms, depending on whether they can guarantee exact or approximate optimal solutions to problems [194]. Hard optimization problems are usually addressed by means of heuristic optimization techniques [109, 143, 167, 2, 190, 142, 66]. One of the most widespread strategies is based on evolutionary optimization. These algorithms are inspired by Darwin's theory of evolution, with the idea of the survival of the fittest, i.e. nature lets evolve individuals that are adapted or fit to the environmental conditions best.

The history of evolutionary algorithms reveals the existence of various variants within the field. However, the fundamental idea underlying all these techniques remains the same. In a scenario where a population of individuals competes for limited resources within an environment, natural selection, or survival of the fittest, occurs. This leads to an increase in the overall fitness of the population. To maximize a given quality function, an initial set of candidate solutions is randomly generated, representing elements within the function's domain. The quality function is then applied to these candidates as a measure of their fitness, aiming for higher values. Based on these fitness values, some of the better candidates are selected to serve as parents for the next generation. Recombination, involving two or more parents, or mutation, applied to a single parent, is then employed to create new candidates, referred to

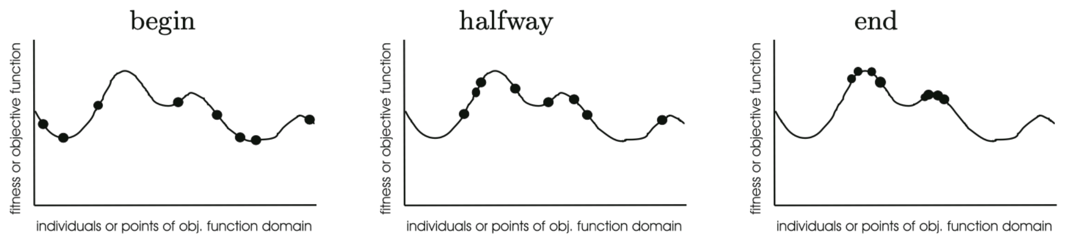


Figure 1.5: Typical progress of an EA illustrated in terms of population distribution. For each point x in the search space y shows the corresponding fitness value [71].

as offspring. The fitness of the offspring is evaluated, and they compete with the previous generation's candidates, considering their fitness and potentially their age, to secure a place in the next generation. This iterative process continues until a candidate with satisfactory quality (a solution) is found or a predefined computational limit is reached.

Overall, two primary forces shape evolutionary systems.

- ***Exploration***: variation operators, such as recombination and mutation, introduce diversity within the population, fostering novelty.
- ***Exploitation***: selection acts as a driving force to enhance the average quality of solutions within the population.

By reaching a good trade-off between exploitation and exploration in an evolutionary algorithm, the fitness values tend to improve across successive generations. This process can be perceived as evolution optimizing or approximating the fitness function, gradually approaching optimal values over time. Figure 1.5 shows the ideal result of the above evolutionary cycle.

It should be noted that many components of such an evolutionary process are stochastic. For example, during selection, the best individuals are not chosen deterministically, and typically even the weak individuals have some chance of becoming a parent or of surviving. During the recombination process, the choice of which pieces from the parents will be recombined is made at random. Similarly, for mutation, the

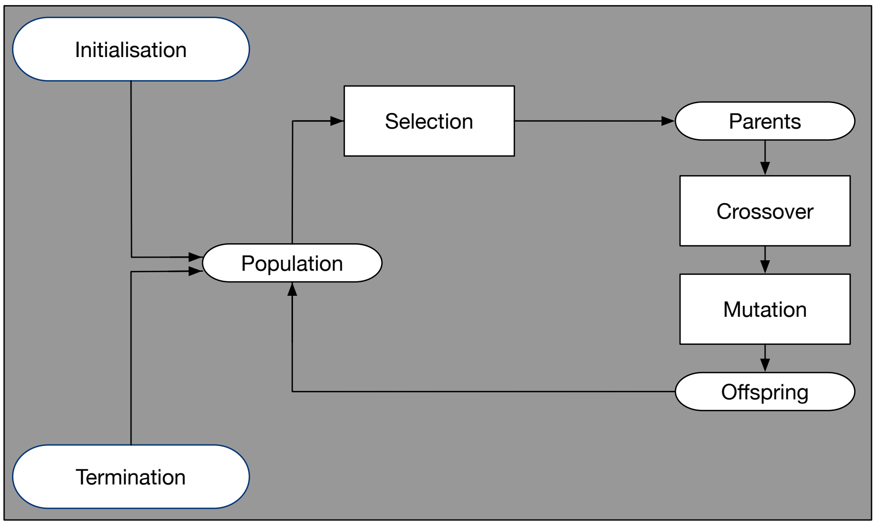


Figure 1.6: The general scheme of an evolutionary algorithm as a flowchart.

choice of which pieces will be changed within a candidate solution, and of the new pieces to replace them, is made randomly. The general scheme of an evolutionary algorithm is given as a flowchart in Fig. 1.6.

The various dialects of evolutionary computing we have mentioned previously all follow these general outlines, differing only in technical details. In particular, one notable distinction among these variants lies in the way candidate solutions are represented. This means that the data structures employed to encode candidates differ across different streams. For instance, genetic algorithms (GAs) typically utilize strings composed of a limited set of symbols, evolution strategies (ESs) employ real-valued vectors, classical evolutionary programming (EP) involves finite state machines, and genetic programming (GP) revolves around trees [71]. Over time, more and more evolutionary algorithms have been introduced, such as Particle Swarm Optimization (PSO) [97], Differential Evolution (DE) [164] among others. However, the focus of this thesis will be on GAs, as their quantum versions will be proposed in later chapters.

1.2.2.I Genetic Algorithms

GAs were proposed by Holland [84] and in its ’canonical’ implementation, they have a binary representation of solutions, fitness proportionate selection, a low probability of mutation, and an emphasis on genetically inspired recombination as a means of generating new candidate solutions. A more in-depth description of crossover and mutation operators will be carried out in Section2.1.2.I, when they are compared with a quantum counterpart. For the same reason, this analysis of selection operators will be carried out in Section 2.1.1.I. Over the years, the canonical GAs have been adapted to work with integer and real solution vectors, changing the operators acting on them accordingly. Moreover, some features have been added to this workflow to boost the performance of the algorithm. For instance, a common mechanism used at the end of each GA generation is the elitism mechanism. In other words, the best chromosome from the old population is carried over to the next one replacing the worst chromosome of the offspring.

Algorithm 1 Pseudo-code of a GA

```
Require: size of the population $pop\_size$, crossover probability $p_c$, mutation probability $p_m$, termination criterion
    $t$, parameters of the mating pool mechanism $m_p$.
Ensure: the best solution $best$.
 1: $gen \leftarrow 0$;
 2: $pop \leftarrow$ generateRandomPopulation($pop\_size$);
 3: evaluateFitness($pop$);
 4: $best \leftarrow$ getBestIndividual($pop$);
 5: while ($t$ is not satisfied) do
 6:     $offspring \leftarrow$ executeMatingPoolSetUp($pop$, $m_p$);
 7:     executeCrossover($offspring$, $p_c$);
 8:     executeMutation($offspring$, $p_m$);
 9:     evaluateFitness($offspring$);
10:     $pop \leftarrow offspring$;
11:     $pop \leftarrow$ elitism($pop$, $best$);
12:     $best \leftarrow$ getBestIndividual($pop$);
13:     $gen \leftarrow gen + 1$;
14: end while
15: return $best$;
```

Overall, the pseudo-code of the GA used in this thesis, is reported in Algorithm 1, where the termination criterion t usually uses the maximum number of generations for the algorithm or the maximum number of fitness evaluations.

1.2.3 Neural Networks

Neural networks (NN) have emerged as one of the most influential advancements in the field of artificial intelligence (AI). Inspired by the structure and functioning of the human brain, neural networks have revolutionized various industries and applications, ranging from computer vision [122] and natural language processing [25] to autonomous vehicles [53] and finance[168]. The roots of NNs can be traced back to the 1940s when the concept of an artificial neuron was introduced. Early pioneers like Warren McCulloch and Walter Pitts laid the foundation by developing models of artificial neurons and simple computational systems [120]. The term "neural network" gained prominence in the 1950s, with the development of the perceptron model by Frank Rosenblatt [150]. The perceptron consists of three main components: input values, weights, and an activation function. The input values represent the features of the input data, and each input value is associated with a weight. These weights determine the significance of each input value in the decision-making process. The activation function takes the weighted sum of the inputs, applies a threshold, and produces a binary output. Fig. 1.7 describes graphically this architecture.

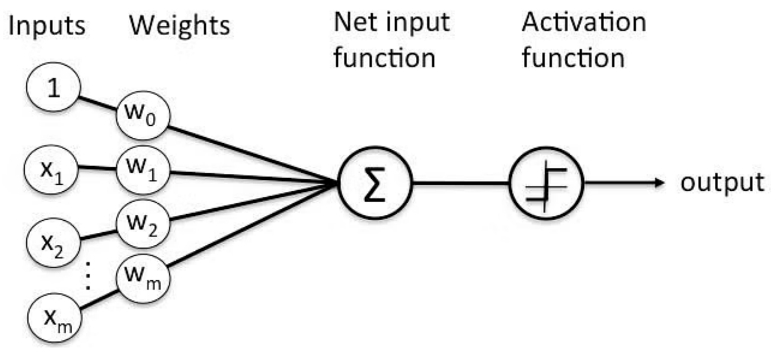


Figure 1.7: Perceptron architecture

With the above architecture it is possible to address binary classification problems in the context of supervised machine learning, if and only if the data to classify are binary separable. In these cases, the Perceptron can be trained adjusting the weights based on the errors made during classification. Initially, the weights are assigned random values. For each input instance, the Perceptron computes the output and compares it with the expected output. If the output matches the expected output, no adjustments are made. However, if the output differs from the expected output, the weights are updated to minimize the error. This adjustment is performed iteratively until the Perceptron achieves satisfactory accuracy.

However, if dataset are not linearly separable then such a linear classifier can never reach a satisfactory level of accuracy. Therefore, a more complex structure is required. Here comes the idea of NNs: the simplest structure of a NN is the 'Multi-layer Perceptron' (MLP) [150], i.e. interconnected artificial neurons, organized into layers. The three primary layers are the input layer, hidden layer(s), and output layer. Each neuron receives inputs, applies an activation function to the weighted sum of those inputs, and passes the output to the next layer, following a process that is known as *forward propagation*. The connections between neurons, known as *synapses*, have associated weights that are adjusted during the training process, usually by means of the *backpropagation* algorithm [151]. Figure 1.8 shows an example of MLP.

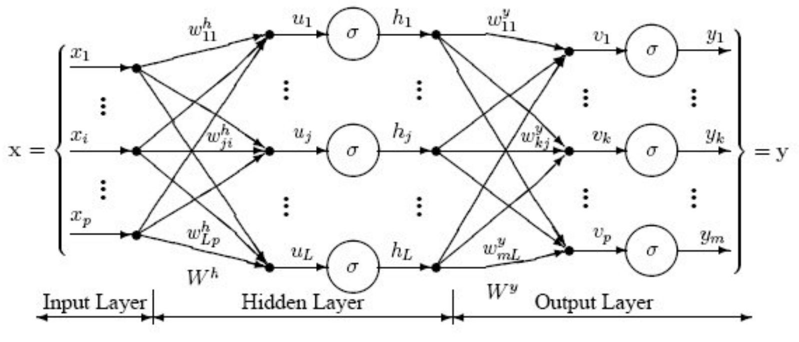


Figure 1.8: MLP architecture

In the face of increasingly complex and diverse data, the realm of artificial neural networks has witnessed a remarkable transition from traditional MLP to the power of deep neural networks (DNNs). The sheer complexity and richness of modern datasets demand models with the capacity to extract intricate and abstract representations.

DNNs with their ability to learn hierarchical features through multiple layers, have emerged as the go-to choice in tackling these challenges.

Chapter 2

Quantum-Enhanced Computational Intelligence

This section aims to introduce different quantum-enhanced computational intelligence algorithms. It is divided into two sections, the former devoted to introducing quantum genetic algorithms (Section 2.1) and the latter proposing different quantum fuzzy engines (Section 2.2). In detail, Section 2.1 is structured as follows: firstly, an introduction to quantum evolutionary algorithms is given (Section 2.1), then the Quantum Genetic Sampling (QGS) and the Quantum Mating Operator (QMO) are introduced and experimentally evaluated in Section 2.1.1 and Section 2.1.2 respectively. Finally, in Section 2.1.3 a study on the effect of quantum noise in quantum genetic algorithms equipped with QMO is carried out. On the other hand, the quantum fuzzy reasoning section is organized as follows: Section 2.2.1 analyzes the literature about quantum fuzzy inference engines; Section 2.2.2 introduces the very first fuzzy inference system able to be run on quantum annealers; Section 2.2.3 proposes and evaluates the QFIE to be run on digital quantum computers and that achieves an exponential speed-up over the classical counterpart.

2.1 Quantum Evolutionary Algorithms

As seen in Section 1.1, phenomena such as *superposition*, *entanglement* and *interference* can allow an enormous degree of parallelism in quantum computation and enable some peculiarities that cannot be replicated efficiently on digital machines. These interesting features, together with the fact that quantum computers are real random number generators, fit very well with the evolutionary optimization paradigm. Indeed, several researchers and practitioners have proposed lately different quantum genetic algorithms (QGAs) [116, 192, 103] or hybrid quantum genetic algorithms (HQGAs) [19, 35] that offer (in theory) improved performance in solving different types of optimization problems, from combinatorial to continuous. However, many of these proposed QGAs-HQGAs cannot be executed on current NISQ hardware that, as seen in Section 1.1.1.III, is strongly limited both in the number of qubits available and in the high level of quantum noise affecting the computation. Considering this, the proposed thesis work doesn't aim to implement wholistic QGAs, but rather to develop precise quantum operators that can exploit the current conditions of NISQ devices to boost the performance of classical GAs. To achieve this goal, it will be proposed a Quantum Genetic Sampling (QGS) algorithm and a Quantum Mating Operator (QMO) to be used in the context of binary-encoded GAs. The studies about the former operator have also been recently published in [14, 17], while the results obtained for QMO have been also reported in [16, 4]. Also, by the end of Section 2.1 it will be reported the study carried out in [12] investigating how the quantum noise affects the performance of QGAs boosting their exploration capability by introducing some randomness which can positively influence the search for the optimal solution.

2.1.1 Quantum Genetic Sampling

As seen in Section 1.2.2, the workflow of GAs is based on the application of an iterative scheme composed of three sequential operators: *selection*, *crossover*, and *mutation*. The joint usage of these operators allows a genetic algorithm to evolve a population of candidate solutions of a given problem towards near-optimal solutions, making these algorithms particularly well suited to efficiently solving hard problems. However, under certain circumstances, genetic algorithms could suffer from two different types of problems: *premature convergence*, occurring when high-rated individuals quickly attain to dominate the population, constraining the algorithm to converge to low-quality local optima ([136]); *random behavior*, occurring when completely random individuals attain to dominate the population, constraining the algorithm to converge towards random solutions ([183]).

Among the genetic operators that can generate the above problems is certainly selection, that is, the mechanism used in genetic processes to choose a set of individuals from the current population to form a pool of parents (*mating pool*) from which the next generation will originate. Indeed, selection provides a mechanism to affect the trade-off between exploration (i.e. exploring the new areas of search space) and exploitation (i.e. using already detected points to search the optimum) in genetic processes ([90]), so as to impact the performance of the whole optimization process. Based on this, the term *selective pressure* is used to classify selection operators by considering a strong (*high selective pressure*) respectively weaker (*smaller selective pressure*) emphasis of selection on the best individuals ([32]). Thus, a selection operator characterized by excessive selection pressure may cause the convergence of a genetic algorithm to a local optimum because of the loss of population diversity, whereas an operator based on low selection pressure may cause a genetic algorithm to provide random results that differ at each iteration of the algorithm ([32]). As a consequence of this strong dependence on the trade-off between exploration and exploitation on the selection mechanism used, several selection operators have been proposed over the years to try to address the effects of premature convergence and random behavior. However, at present there is no selection mechanism that can properly handle these problems. As a result, one could completely overhaul the way genetic algorithms form the mating pool and introduce a completely new sampling mechanism.

The main goal of this research is to fill this gap by replacing the genetic algorithm selection mechanism with a sampling operator based on quantum computation, named *Quantum Genetic Sampling (QGS)*. This novel approach exploits the stochastic nature of quantum mechanics to collect the individuals for the creation of the mating pool considering the following concept: the probability of choosing elements belonging to the current genetic population is proportional to their fitness value, as usual; also, the probability of choosing elements outside the current genetic population is greater than zero. The implementation of the proposed operator is based on a well-known quantum primitive, named *Quantum Amplitude Amplification* (QAA). QAA is a quantum algorithm used to change the probability distribution modeled by a quantum state by increasing the probability of measurement of so-called *marked items*. A genetic algorithm equipped with QGS uses a quantum state that embodies all possible solutions to a problem and exploits quantum amplitude amplification to opportunely change that quantum state. In particular, quantum amplitude amplification adjusts the probability of individuals in the current genetic population to be selected to enter the mating pool, according to their fitness values. The use of an appropriate quantum state to model the probability distribution used to sample individuals that will belong to the mating pool has an important side effect: there is a non-zero probability of sampling (that is, introducing in the mating pool) individuals do not belong to the current genetic population. This behavior leads to a key benefit because it allows, at each iteration, new genetic material to be created and added to the mating pool. Consequently, the use of QGS increases diversity in the genetic population and reduces the probability of premature convergence to local optima. QGS has been designed as an adaptive operator in that it can also adjust its ability to affect genetic population diversity by using appropriate hyperparameters. QGS has been developed on actual quantum backends from the IBM Q Experience® platform[1]. Experimental studies have been carried out to show the suitability of QGS in being embedded in a genetic optimization process and reaching a good trade-off between exploitation and exploration. This study involves a set of well-known benchmark functions used to test the performance of the proposed quantum operator against traditional selection methods in terms of the obtained solution quality and the genetic population diversity. Moreover, a statistical comparison procedure based on the Wilcoxon signed rank test ([179]) has been carried out to investigate about the impact of the proposed quantum genetic operator on the performance of genetic algorithms. The Section is structured as follows: firstly, it will provide a detailed discussion of the selection methods existing in the literature; secondly, it will describe with detail the QGS algorithm, including a discussion about hyperparameters; finally, it will highlight the setting of experiments carried out to evaluate QGS and their comparative results.

2.1.1.I Related Work

Selection methods represent the driving force of the evolution of genetic algorithms. Indeed, without it, the search would be no better than random ([33]). The design of a good selection method requires to balance between fast search (exploitation) and sustaining diversity (exploration). Over the years, several selection methods have been developed in order to try to satisfy these requirements. All approaches in literature can be classified into four categories ([78]): proportionate reproduction, ranking selection, tournament selection, and "steady-state" selection.

The first category, proportionate reproduction, includes methods that choose indi-

[1] `https://quantum-computing.ibm.com/`

viduals according to their fitness values. Formally, in these methods, the probability p to select the i-th individual at t-th generation is given by $p_{i,t} = \frac{f_{i,t}}{\sum_{j=1}^{N} f_{j,t}}$, where $f_{i,t}$ is the fitness value and N is the size of the population. The method par excellence belonging to this category is surely the *Roulette Wheel Selection* (RWS) ([33]), which chooses several solutions from the population by repeated random sampling. RWS is characterized by a high selective pressure in the first generations and a low one near the convergence. In detail, the initial high selective pressure leads to a population composed of individuals with similar fitness values, which, in turn, leads RWS to become quite a random search (and, hence, characterized by a low selective pressure). The negative consequence of this behavior is the achievement of a premature convergence. A variant of RWS aimed at reducing this risk is known as *Stochastic Universal Sampling* (SUS) ([34]). Differently from RWS, SUS uses a single random value to sample all of the solutions by choosing them at evenly spaced intervals ([37]). This feature allows SUS to improve on RWS by keeping a constant selective pressure. However, the risk of premature convergence is not completely deleted ([121]).

Regarding ranking selection methods, they sort the population on the basis of fitness values, and then allocate selection probabilities to individuals according to their rank, rather than according to their actual fitness values ([72]). The most popular among the methods belonging to this second category is the *Linear Rank Selection* (LRS) ([78]). In detail, this method uses a linear mapping from rank number to selection probability. Formally, the selection probability p of the i-th individual at the t-th generation is given by $p_{i,t} = \frac{r_{i,t}}{N(N-1)}$, where $r_{i,t}$ is the rank and N is the population size. LRS is characterized by a constant but limited selection pressure that leads to a slow convergence.

The previous two categories of selection methods perform sampling by relying on the knowledge of the entire population. The third category of selection methods, the *Tournament Selection* (TS) ([78]), is based on an ordering relation that compares only k individuals, where k is called tournament size. The general scheme of the methods in this category consists of repeating N times (where N is the population size) the following steps: 1) randomly selecting a set of k individuals; 2) choosing the fittest individual among them to be inserted in the parent mating pool. Hence, the methods belonging to this category differ among them for the value associated with k. The most popular tournament selection method is the binary Tournament Selection (TS) characterized by $k = 2$, but general $k - ary$ tournaments are often applied. Currently, the tournament selection methods are the most widely used selection operators thanks to the fact that the selection pressure is easy to control by means of hyperparameter k. Indeed, the larger the tournament size, the greater the chance that it will contain members of above-average fitness. Thus the probability of selecting a high-fitness member increases, as k is increased. Hence, increasing k increases the selection pressure ([72]).

Finally, the "steady-state" selection represents a completely different approach with respect to previous ones since it selects only a few individuals to be reproduced, whereas, most part of the population survives to the next generation. This method was popularized by Darrell Whitley and Joan Kauths GENITOR system ([178]), i.e., an alternative to the traditional generational approach in genetic algorithms. Precisely, the idea is to iteratively breed a new child or two, assess their fitness values, and then reintroduce them directly into the population itself, killing off some preexisting individuals (typically the worst ones) to make room for them ([114]). The "steady-state" selection is fairly exploitative compared to a generational approach: the parents stay around in the population, potentially for a very long time, and thus, this runs the risk of causing the system to prematurely converge to large copies of a few highly fit individuals ([114]).

Starting from this analysis, this research aims to develop a genetic operator able to perform the filling of the parent mating pool in genetic algorithms by achieving a good balance between exploration and exploitation. The goal of this research is to enhance the literature by introducing a new genetic operator, named QGS, capable of achieving this task better than state-of-the-art selection approaches, as described in the experimental session, by using the stochastic nature of quantum mechanics. This quantum view allows QGS to operate in a completely different way than the four mentioned categories of selection operators. In fact, unlike conventional approaches, QGS can insert in the mating pool genetic material not present in the current genetic population.

2.1.1.II Quantum Amplitude Amplification

To fully understand QGS, it is required to describe its core, i.e. the Amplitude Amplification routine [46, 87, 126]. This technique in quantum computing is the basis of many well-known algorithms, such as Grovers algorithm among others.

The main goal of amplitude amplification is to maximize the probability of measuring one or more desired basis states among all those present in a given quantum superposition. As described in [94], this task can be achieved by means of an iterative scheme where, at each iteration, the following two steps are carried out:

- *flip step*: this step allows the algorithm to target the desired M basis states and distinguish their phases from the others thanks to a phase rotation of π.
- *mirror step*: This step consists of an inversion about the average of the amplitudes of all basis states. In other words, this operation finds a different sequence of amplitudes such that (i) the new amplitudes have the same distance as before from the mean, but inverted around the mean, (ii) the new sequence has the same mean.

The quantum amplitude amplification algorithm consists of iterating these two steps (flip and mirror) multiple times to further increase the probability value to measure the marked states. However, it is important to note that after a certain number of iterations, the probability of the marked states will start decreasing. This is related to the fact that, at a certain iteration, after the mirror step, the marked states will be out of phase with the others, so the next flip subroutine will cause all phases to be in alignment. At that point, there will be no marked states, so further iterations will start diminishing the probability of the initially marked states until ending up with the original state [94]. Therefore, it is very important to set opportunely the number of iterations to achieve the maximum probability for the marked states. In literature, it is found that the optimal number of iterations N_{AA} for a quantum register of n qubits is [94]:

$$N_{AA} = \lfloor \frac{\pi}{4}\sqrt{\frac{2^n}{M}} \rfloor \tag{2.1}$$

where M is the number of marked states.

Mathematically, let $\mathcal{H}$ be a Hilbert space of a n-dimensional quantum system and $\chi : \{0, 1, \ldots, 2^n-1\} \rightarrow \{0, 1\}$ be a boolean function that induces a partition of $\mathcal{H}$ into a marked subspace (spanned by the set of basis states $|j\rangle \in \mathcal{H}$ for which $\chi(j) = 1$, with $j \in \{0, 1, \ldots, 2^n - 1\}$) and its complementary subspace (spanned by the set of basis states $|j\rangle \in \mathcal{H}$ for which $\chi(j) = 0$, with $j \in \{0, 1, \ldots, 2^n - 1\}$). Moreover, let $\mathcal{A}$ be any quantum algorithm that acts on $\mathcal{H}$ that does not use measurements and $|\Psi\rangle = \mathcal{A}|0\rangle$ denote the state obtained by applying $\mathcal{A}$ to the initial zero state. Thus, the QAA algorithm consists of applying the operator $\mathbf{Q}$, defined in (2.2), repeatedly on the state $|\Psi\rangle$ [46].

$$\mathbf{Q} = \mathbf{Q}(\mathcal{A}, \chi) = -\mathcal{A}\mathbf{S}_0\mathcal{A}^{-1}\mathbf{S}_\chi \tag{2.2}$$

Here, the operator $\mathbf{S}_\chi$ conditionally changes the sign of the amplitudes of the marked states as follows:

$$|x\rangle \rightarrow \begin{cases} -|x\rangle & \text{if } \chi(x) = 1 \\ |x\rangle & \text{if } \chi(x) = 0 \end{cases} \tag{2.3}$$

where i is the imaginary unit, whereas, the operator $\mathbf{S}_0$ changes the sign of the amplitudes if and only if the state is the zero state $|0\rangle$. In other words, the $\mathbf{S}_\chi$ is a Phase Oracle of the form reported in (1.14). In Grover's algorithm $\mathcal{A} = H^{\otimes n}$, leading the state $|\Psi\rangle$ in a uniform superposition of the basis states, and the phase oracle inverts the phase of the states that need to be found.

Graphically, the circuit representing the amplitude amplification process is reported in Fig. 2.1 where the $\mathcal{FP}$ operation is implemented by the operator $\mathbf{S}_\chi$, the $\mathcal{MP}$ operation is implemented by the product $\mathcal{A}\mathbf{S}_0\mathcal{A}^{-1}$ and $\mathcal{A} = H^{\otimes n}$ like the standard Grover's algorithm.

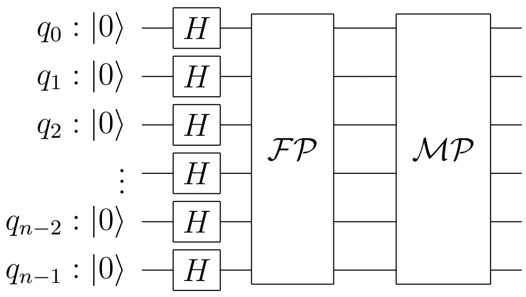


Figure 2.1: Circuital representation of amplitude amplification process on an n-qubit register.

In literature, there are some extensions of the conventional amplitude amplification that use arbitrary phase rotations of the marked states [81, 189]. In these approaches, the operator $\mathbf{S}_\chi$ is defined as follows:

$$|x\rangle \rightarrow \begin{cases} e^{i\phi}|x\rangle & \text{if } \chi(x) = 1 \\ |x\rangle & \text{if } \chi(x) = 0 \end{cases} \tag{2.4}$$

where i is the imaginary unit, whereas, the operator $\mathbf{S}_0$ multiplies the amplitude by a factor of $e^{i\phi}$ if and only if the state is the zero state $|0\rangle$.

In our context, QGS uses this variant of the QAA process by relating ϕ to the fitness value of individuals. In this way, as described in the next section, each marked state will be characterized by a different probability of being measured according to its own fitness value (i.e. the higher the quality of the solution, the higher the probability of being measured).

2.1.1.III Quantum Genetic Sampling Workflow

In this section, a quantum-based genetic operator, aimed at replacing the selection mechanism in the conventional workflow of genetic algorithms, is introduced. Due to the embryonic nature of quantum computers, which are equipped with a limited number of qubits, the proposed operator will be embedded in a binary-encoded genetic algorithm. In such a context, QGS aims to create a mating pool through two sequential steps:

1. Represent the search space of a specific problem by preparing a quantum state $|\psi\rangle$, which models a probability distribution;
2. Use quantum amplitude amplification to adjust the quantum state $|\psi\rangle$, according to the fitness values of the best individuals belonging to the genetic population.

Once the quantum amplitude amplification has properly adjusted the quantum state, the quantum measurement is used to extract individuals for inclusion in the mating pool, considering the relative probability distribution. The behavior of QGS and its ability to adjust the trade-off between exploration and exploitation depends on a set of hyperparameters. These hyperparameters act both on the definition of the quantum state $|\psi\rangle$ and on the execution of the quantum amplitude amplification.

The first step of QGS is the preparation of the quantum state $|\psi\rangle$ that models the search space of a problem. Let $\mathcal{U} = \{s_0, s_1, \ldots, s_{2^n-1}\}$ be the set of all possible binary strings composed of n binary digits, and let $\mathcal{P}$ and $\mathcal{B}$ be two sets[2] such that

[2]For sake of simplicity, a genetic population is considered as a set rather than a multi-set without loss of generality.

$|\mathcal{P}| = k$, $|\mathcal{B}| = 2^n - k$, $\mathcal{P} \cup \mathcal{B} = \mathcal{U}$ and $\mathcal{P} \cap \mathcal{B} = \emptyset$:

$$\begin{aligned} \mathcal{P} &= \{s_{z_0}, s_{z_1}, \dots, s_{z_{k-1}} | z_j \in \{0, 1, \dots, 2^n - 1\}\} \\ \mathcal{B} &= \{s_{l_0}, s_{l_1}, \dots, s_{l_{2^n-k-1}} | l_j \in \{0, 1, \dots, 2^n - 1\}\} \end{aligned} \tag{2.5}$$

where $\mathcal{P}$ represents a set of binary strings belonging to a genetic population, whereas $\mathcal{B}$ represents a set of binary strings that do not belong to the genetic population $\mathcal{P}$. Thus, a superposed quantum state $|\psi\rangle$ modelling a genetic population can be defined as:

$$|\psi\rangle = \sum_{j=0}^{k-1} c_{s_{z_j}} \left|s_{z_j}\right\rangle + \sum_{j=0}^{2^n-k+1} c_{s_{l_j}} \left|s_{l_j}\right\rangle \tag{2.6}$$

where $c_{s_{z_j}}, c_{s_{l_j}} \in \mathbb{C}$. In this scenario, a given individual belonging to the genetic population, namely $s_{z_j} \in \mathcal{P}$, has a probability $|c_{s_{z_j}}|^2$ to be selected when the quantum state $|\psi\rangle$ is measured. At the same way, an item belonging to the set $\mathcal{B}$, namely $s_{l_j} \in \mathcal{B}$, has a probability $|c_{s_{l_j}}|^2$ to be selected when the quantum state $|\psi\rangle$ is measured. It means that QGS is able to select both individuals belonging to and not belonging to the current genetic population, increasing the level of diversity in the generation of the next population. However, so as to enable QGS to work properly, the value of above probabilities $|c_{s_{z_j}}|^2$ and $|c_{s_{l_j}}|^2$ need to be set up to maximize the likelihood that best individuals in $\mathcal{P}$ are selected, and minimize the likelihood that binary strings in $\mathcal{B}$ are selected. In order to achieve this goal, the third step of QGS uses a modified version of the quantum amplitude amplification described in (2.4).

In detail, the workflow of QGS can be summarized as follows:

1. Let $\mathcal{P}' = \{s_{z_0}, s_{z_1}, \dots, s_{z_{t-1}} | z_j \in \{0, 1, \dots, 2^n - 1\}\}$ be the set of t best individuals of $\mathcal{P}$ ordered such that the fitness value of the individual s_{z_j} is better than that of the individual $s_{z_{j+1}}$, where $j = 0, 1, \dots, t-1$. Let $\mathcal{Q} = \{s_{l_0}, s_{l_1}, \dots, s_{l_{2^n-t-1}} | l_j \in \{0, 1, \dots, 2^n - 1\}\}$ be the complementary set of $\mathcal{P}'$ composed of all the remaining possible binary strings of n binary digits.

2. Initialize a quantum register $\mathcal{R}$ of n qubits to the zero state, $|\psi_0\rangle = |0\rangle^{\otimes n}$. Then, apply n Hadamard gates to create a quantum uniform superposition of elements belonging to $\mathcal{P}'$ and $\mathcal{Q}$ as follows:

$$\begin{aligned} |\psi\rangle =& \mathcal{A}|\psi_0\rangle = \mathbf{H}^{\otimes n}|\psi_0\rangle = \sum_{s=0}^{2^n-1} \frac{1}{\sqrt{2^n}}|s\rangle = \\ =& \sum_{j=0}^{t-1} \frac{1}{\sqrt{2^n}}|s_{z_j}\rangle + \sum_{j=0}^{2^n-t-1} \frac{1}{\sqrt{2^n}}|s_{l_j}\rangle \end{aligned} \tag{2.7}$$

3. Define a quantum circuit using the register $\mathcal{R}$ and perform the amplitude amplification for the states related to individuals in $\mathcal{P}'$ as follows:

   - Define a Boolean function $\chi : \{0, \ldots, 2^n - 1\} \to \{0, 1\}$ similar to the one used in (2.3):

$$\begin{cases} \chi(s) = 1 & \text{if } s \in \mathcal{P}' \\ \chi(s) = 0 & \text{if } s \notin \mathcal{P}' \end{cases} \tag{2.8}$$

   - Execute the flip phase operation $\mathcal{FP}$ as described in (2.4), marking the quantum states $|s_{z_j}\rangle$ as follows:

$$|s_{z_j}\rangle \to \begin{cases} e^{i(\pi - j\beta)}|s_{z_j}\rangle & \text{if } \chi(s_{z_j}) = 1 \\ |s_{z_j}\rangle & \text{if } \chi(s_{z_j}) = 0 \end{cases} \tag{2.9}$$

   where i is the imaginary unit, $\beta \in \left[0, \frac{\pi}{t+1}\right]$ and $j = 0, 1, \ldots, t-1$. This is a key step in QGS workflow because it ensures that the t marked basis states belonging to $\mathcal{P}'$ have a relative phase different from the basis states belonging to $\mathcal{Q}$. Hence, the probability of measuring the t marked basis states will be increased in the next mirror step with respect to the other basis states. Moreover, the choice of setting ϕ to $\pi - j\beta$ in (2.3) permits to obtain a different probability of measurement also among the t marked states. In particular, the higher the value of ϕ, the higher the probability of measurement. Therefore, the individual s_{z_j} will be characterized by a higher probability of being measured with respect to the individual $s_{z_{j+1}}$ (because ϕ increases when j decreases). Hence, among the t marked states, the individuals with a better fitness value will have a higher probability of being inserted in the mating pool.

- Execute the mirror operation $\mathcal{MP}$ and iterate the operations $\mathcal{FP}$ and $\mathcal{MP}$ for a chosen number of iterations N_G, obtaining the final quantum state $|\psi_f\rangle$:

$$|\psi_f\rangle = \sum_{j=0}^{t-1} c_{s_{z_j}} \left|s_{z_j}\right\rangle + \bar{c} \sum_{j=0}^{2^n-t+1} \left|s_{l_j}\right\rangle \tag{2.10}$$

such that

$$|c_{s_{z_0}}|^2 > |c_{s_{z_1}}|^2 > \cdots > |c_{s_{z_{t-1}}}|^2 > |\bar{c}|^2 \tag{2.11}$$

where $\sum_{j=0}^{t-1} |c_{s_{z_j}}|^2 + |\bar{c}|^2 \times (2^n - t) = 1$.

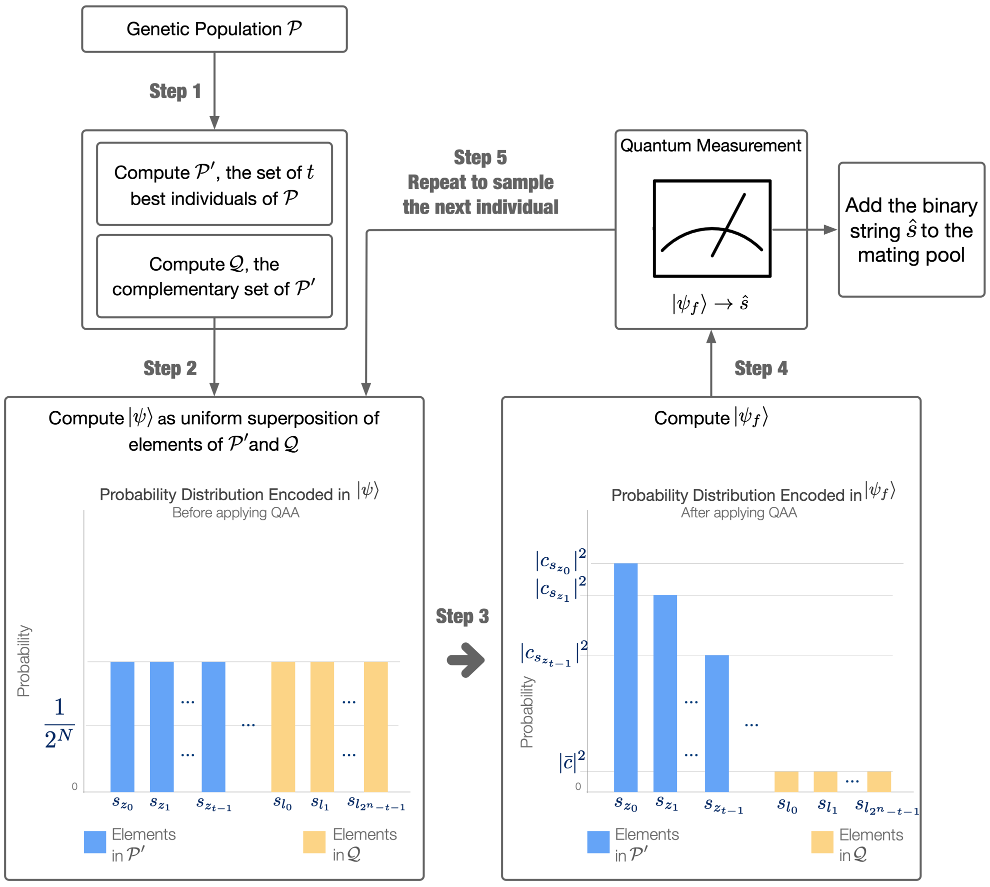
4. Measure $|\psi_f\rangle$ to obtain one individual $\hat{s}$, representing one of the 2^n binary strings belonging to $\mathcal{U}$, to be inserted in the mating pool.

5. Repeat k times the steps from 2. to 4. to obtain the whole mating pool from which the next generation will originate.

To enhance the understanding of QGS steps, Fig. 2.2 shows a graphical workflow.

According to the QGS workflow, the measurement probability of an individual (that is, the probability of introducing the individual in the mating pool) can be tuned by means of three hyperparameters, namely t, β, N_G, where:

- The hyperparameter t sets the size of the sub-population $\mathcal{P}'$ of individuals with the best fitness values in $\mathcal{P}$, and, it corresponds to the number of marked states in the quantum state $|\psi\rangle$. Hence, in order to ensure the good behavior of the amplitude amplification algorithm $t << 2^n$. However, it is worth noting that



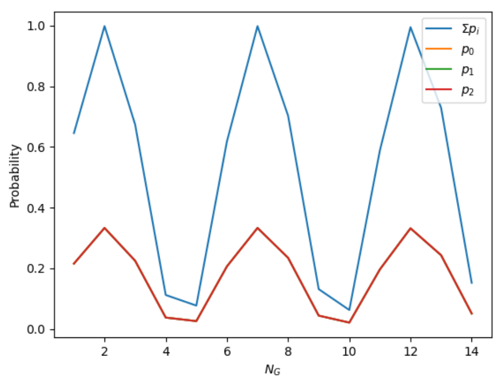
Genetic Population $\mathcal{P}$
Step 1
Compute $\mathcal{P}'$, the set of t best individuals of $\mathcal{P}$
Compute $\mathcal{Q}$, the complementary set of $\mathcal{P}'$
Step 5
Repeat to sample the next individual
Quantum Measurement
$|\psi_f\rangle \rightarrow \hat{s}$
Add the binary string $\hat{s}$ to the mating pool
Step 2
Step 4
Compute $|\psi\rangle$ as uniform superposition of elements of $\mathcal{P}'$and $\mathcal{Q}$
Probability Distribution Encoded in $|\psi\rangle$
Before applying QAA
Probability
$\frac{1}{2^N}$
0
s_{z_0} s_{z_1} $s_{z_{t-1}}$ s_{l_0} s_{l_1} $s_{l_{2^n-t-1}}$
Elements in $\mathcal{P}'$
Elements in $\mathcal{Q}$
Step 3
Compute $|\psi_f\rangle$
Probability Distribution Encoded in $|\psi_f\rangle$
After applying QAA
$|c_{s_{z_0}}|^2$
$|c_{s_{z_1}}|^2$
$|c_{s_{z_{t-1}}}|^2$
$|\bar{c}|^2$

Figure 2.2: Graphical representation of QGS.

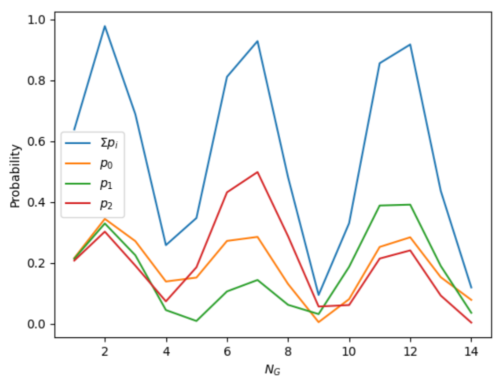
a value for t too small (e.g. $t = 1$ or $t = 2$) leads QGS will introduce in the mating pool the same few individuals with a very high probability by reducing population diversity.

- The hyperparameter β sets the angle for the progressive rotation of the phase for the states $\left|s_{z_j}\right\rangle_{j\in[0,t-1]}$. The upper limit for β is defined by the number of states to mark t. If β is equal to zero, QGS works by following the standard amplitude amplification algorithm: all the phases of the marked states $\left|s_{z_j}\right\rangle_{j\in[0,t-1]}$ are rotated with the same angle π and the probability to measure one of them is uniform: $|c_{s_{z_0}}|^2 = |c_{s_{z_1}}|^2 = \cdots = |c_{s_{z_{t-1}}}|^2 > |\bar{c}|^2$. Otherwise, the closer β to its upper limit, the greater the inequality between the probabilities shown in (2.11), according to the number of iterations N_G.

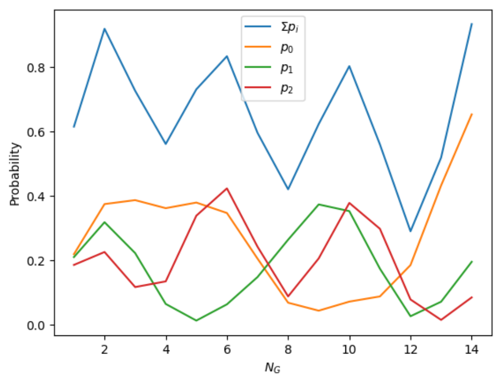
- The hyperparameter N_G represents the number of iterations of the amplitude amplification algorithm. If the value of β is 0, then QGS works similarly to the standard amplitude amplification algorithm thus the optimal value for N_G is surely given by (2.1). Otherwise, indicating with $p_j = |c_{s_{z_j}}|^2$ the probability to measure $\left|s_{z_j}\right\rangle$, if $\beta \neq 0$ the number of iterations N_G could be tuned in order to find a more opportune trade-off among the values $p_0, \ldots, p_{t-1}$ related to the marked t states.



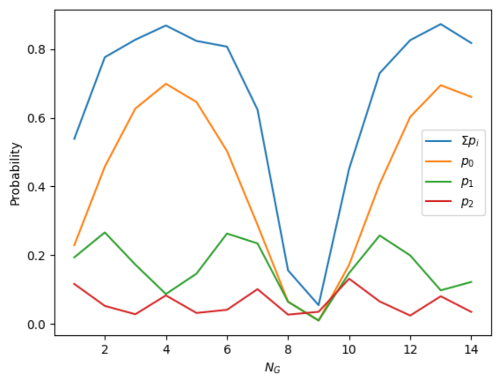
(a) $\beta = 0$



(b) $\beta = \pi/16$



(c) $\beta = \frac{\pi}{8}$



(d) $\beta = \frac{\pi}{4}$

Figure 2.3: Graphical representation of how the probability values of measuring $t = 3$ marked states change against N_G by considering in (a) $\beta = 0$, (b) $\beta = \frac{\pi}{16}$, (c) $\beta = \frac{\pi}{8}$ and (d) $\beta = \frac{\pi}{4}$.

An example of how the hyperparameters affect the probabilities of the marked states is reported in Fig. 2.3. In detail, the reported plots show the probabilities of $t = 3$ marked states and their sum (the blue line) by considering a quantum register with $n = 5$ qubits and, respectively, $\beta = 0$, $\beta = \frac{\pi}{16}$, $\beta = \frac{\pi}{8}$ and $\beta = \frac{\pi}{4}$. By seeing these plots, it is possible to highlight some of the aforementioned observations:

- The probability of each one of the $t = 3$ marked states is characterized by periodicity, even if it is imperfect. Hence, it is not necessary to consider values for N_G too high. It is sufficient to take into account a value for N_G that shows the first period of all marked states;
- if $\beta = 0$ (see Fig. 2.3a) the probabilities p_0, p_1 and p_2 of the $t = 3$ marked states are overlapped, i.e., the same;
- if $\beta \neq 0$ (see Figs. 2.3b, 2.3c and 2.3d), the higher the β value, the higher is the difference among the probabilities of the $t = 3$ marked states.

2.1.1.IV Experiments and results

Let's see now the results of our experimental study carried out to highlight the suitability of QGS to be used instead of a selection operator in genetic algorithms and study its impact on the performance yielded by these optimization algorithms. To achieve this goal, a traditional binary-encoded genetic algorithm with the generational approach has been implemented and equipped with QGS or with one of the state-of-the-art selection methods, namely RWS, SUS, LRS, and TS (see Fig. 2.4). For the sake of completeness, also a random selection denoted as RS, is considered in the study.

The performance of the different genetic algorithms, each one equipped with a different mechanism for creating the mating pool (QGS, RWS, SUS, LRS, TS, RS) has been assessed in terms of two metrics, namely the *average fitness value* and the *average genetic population diversity*, useful to evaluate genetic algorithms' exploration and exploitation capability. Moreover, similar to other works ([56, 54]), a statistical procedure has been conducted to evaluate the significance of the impact of the proposed quantum genetic operator on the performance of genetic algorithms.

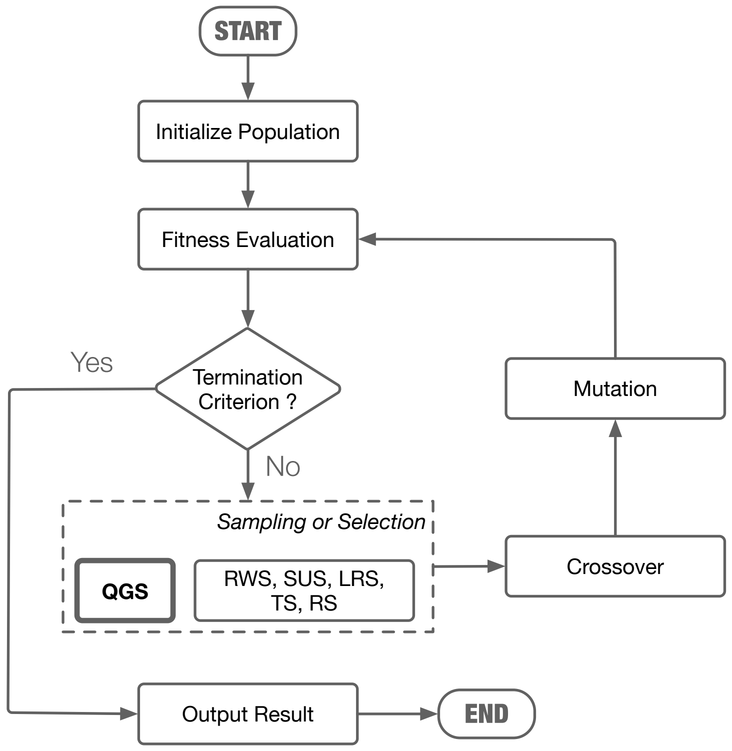


Figure 2.4: A genetic algorithm equipped with QGS or conventional selection operators.

The whole experimentation involves the application of different genetic algorithms to solve a set of typical benchmark functions, both binary and real functions. For the latter, a discretization procedure has been used to map real solutions into binary ones. The experimental study has been organized to test part of the benchmark functions in a simulated quantum environment, and another part in a real quantum environment so that the robustness of the proposed operator to be executed in both computational scenarios could be evaluated. Moreover, in both contexts, the experiments have been performed using the maximum number of qubits such that QGS was not affected by quantum noise. Hereafter, details about the used benchmark functions, the configuration setup of experiments, the outcomes of the study, and the results of the statistical comparison for binary and real optimization problems are given.

Benchmark functions

Our experimental study involves a set of benchmark functions already used in the literature related to both binary and real optimization problems in order to investigate our proposal in different application scenarios. Firstly, Table 2.1 reports the used n-dimensional binary optimization problems characterized by one or more global optima (reported in Table 2.2). In detail, f_1, f_2 and f_3 are used in ([67]), whereas f_4, f_5, f_6 and f_7 are introduced in ([185]). Note that with the notation $u_i^j(x)$ used in the functions f_4, f_5, f_6 and f_7, it is intended as the number of ones in the solution sub-string that goes from the i-th bit to the j-th bit where $j > i$ computed as follows:

$$u_i^j(x) = \sum_{k=i}^{j} x_k. \tag{2.12}$$

Table 2.1: Binary benchmark functions

Function definition
$f_1(x) = \sum_{i=0}^{n-1} x_i$
$f_2(x) = \sum_{i=0}^{n-1} i \cdot x_i$
$f_3(x) = \sum_{i=0}^{n-1} \prod_{j=0}^{i} x_j$
$f_4(x) = \begin{cases} 1 & \text{if } u_0^{n-1}(x) = 0 \text{ or } 6 \\ 0 & \text{if } u_0^{n-1}(x) = 1 \text{ or } 5 \\ 0.360384 & \text{if } u_0^{n-1}(x) = 2 \text{ or } 4 \\ 0.640576 & \text{if } u_0^{n-1}(x) = 3 \end{cases}$
$f_5(x) = \sum_{i=0}^{1} g_i(x) \text{ where } g_i(x) = \begin{cases} 1 & \text{if } u_{i\frac{n}{2}}^{\frac{n}{2}-1+i\frac{n}{2}} = 0 \\ 0 & \text{if } u_{i\frac{n}{2}}^{\frac{n}{2}-1+i\frac{n}{2}} = 1 \\ 0.360384 & \text{if } u_{i\frac{n}{2}}^{\frac{n}{2}-1+i\frac{n}{2}} = 2 \\ 0.640576 & \text{if } u_{i\frac{n}{2}}^{\frac{n}{2}-1+i\frac{n}{2}} = 3 \end{cases}$
$f_6(x) = \begin{cases} 1 - 0.2 \cdot u_0^{n-1}(x) & \text{if } u_0^{n-1}(x) \leq n-1 \\ n & \text{if } u_0^{n-1}(x) = n \end{cases}$
$f_7(x) = \sum_{i=1}^{2} g_i(x) \text{ where } g_i(x) = \begin{cases} 1 - 0.2 \cdot u_{(i-1)\cdot\frac{n}{2}+(i-1)}^{i\cdot\frac{n}{2}}(x) & \text{if } u_{(i-1)\cdot\frac{n}{2}+(i-1)}^{i\cdot\frac{n}{2}}(x) \leq 3 \\ 3 & \text{if } u_{(i-1)\cdot\frac{n}{2}+(i-1)}^{i\cdot\frac{n}{2}}(x) = 3 \end{cases}$

Secondly, our experimentation involves real-parametrized optimization problems already used in ([19]). Table 2.3 reports the considered functions and the upper and lower bounds for their variables. All of them represent mono-dimensional continuous optimization problems with one or more global optima (see Table 2.2). Due to the binary encoding used by the conventional designed GA, a discretization procedure is

Table 2.2: Absolute optimum values for all considered benchmark functions.

Function	**Optimum**	**Function**	**Optimum**
f_1	6	g_1	10.88
f_2	15	g_2	10.46
f_3	6	g_3	7.04
f_4	1	g_4	21.04
f_5	2	g_5	8.99
f_6	6	g_6	11.86
f_7	6		

applied to map real values in binary ones. To achieve this goal, the problem search space is discretized in a uniform way and each value of the problem variables is coded in a n fixed-length binary string ([62, 63]). In detail, as described in ([19]), the set ξ of the discrete values for each variable is calculated as follows:

$$\xi = \{x_l + \lambda * \frac{x_u - x_l}{2^n - 1} : \lambda \in \{0, 1, \ldots, 2^n - 1\}\} \tag{2.13}$$

where x_l and x_u are the lower and upper bounds of the variable, respectively, and n is the length of the binary string. Hence, precisely, the fixed-length binary string encodes the integer value λ. Moreover, Gray code binary representations have been used in order to guarantee the distance-preserving, i.e., small changes in a binary string result in small changes in the number that is represented ([62]).

Table 2.3: Properties of the discretized benchmark functions

Test function	**Bounds**
$g_1(x) = \sin(x) + \sin\left(\frac{10}{3}x\right)$	$x_l = 2.7, x_u = 7.5$
$g_2(x) = -e^{-x}\sin(2\pi x)$	$x_l = 0.0, x_u = 4.0$
$g_3(x) = \frac{x^2-5x+6}{x^2+1}$	$x_l = -5.0, x_u = 5.0$
$g_4(x) = -x\sin(x)$	$x_l = 0.0, x_u = 13$
$g_5(x) = -x^{2/3} - (1-x^2)^{1/3}$	$x_l = 0.001, x_u = 0.99$
$g_6(x) = \sin(x) + \sin\left(\frac{2}{3}x\right)$	$x_l = 3.1, x_u = 20.4$

To ensure the correct behavior of RWS, all benchmarks have been considered as maximization problems. Due to the limited number of qubits and their noisy nature, our experimentation considers as problem dimension $n = 5$ for real-optimization functions and $n = 6$ for binary functions (since the binary functions f_4, f_5, f_6 and f_7 require that n is even). It is worth noting that the small dimensionality is not a limitation of our approach that will be able to be applied to multi-dimensional benchmark functions when IBM quantum computers characterized by a large number of qubits and low noise will be made available via cloud [3].

Experimental Setup

The experimental study consists of running a traditional binary-encoded genetic algorithm with the generational approach equipped with a different mechanism for creating the mating pool (QGS, RWS, SUS, LRS, TS, and RS) at a time. The pseudo-code of the implemented GA is the one reported in Algorithm 1. Roughly, a population of chromosomes is generated randomly and evaluated by using the fitness function. Successively, the algorithm evolves by means of a set of iterations until some termination criteria are satisfied. In our experimentation, the algorithm ends when a maximum number of iterations is reached. In each iteration, the following three steps are executed: 1) filling of the mating pool by means of one of the compared operators; 2) recombination to generate an offspring; 3) eventual mutations. Finally, the offspring is evaluated and replaces the current population. The implemented genetic algorithm is also equipped with an elitism mechanism. In other words, the best chromosome from the old population is carried over to the next one replacing the worst chromosome of the offspring.

In our experimentation, typical values for genetic algorithm hyperparameters have been set without performing a tuning procedure since the goal of the designed genetic algorithm is not to be the best optimization solver, but, to be a casing for the compared operators. Table 2.4 reports the genetic algorithm hyperparameter setting. In particular, the hyperparameters *pop_size* and *num_iters* have been set

[3]https://research.ibm.com/blog/ibm-quantum-roadmap

in combination in order not to exceed the number of fitness function evaluations for exhaustive research (32 for $n = 5$ and 64 for $n = 6$).

Table 2.4: Hyperparameters of the designed genetic algorithm

Hyperparameter	**Value**
Population size *pop_size*	7 for $n = 6$ and 5 for $n = 5$
Crossover	Two-point crossover with probability $p_c = 0.8$
Mutation	Bit flip mutation with probability $p_m = 0.005$
Number of iterations *num_iters*	5 for $n = 6$ and 5 for $n = 5$

As for the compared operators, only TS and QGS are characterized by hyperparameters and, for these, a tuning procedure has been carried out. In particular, binary TS resulted in the best configuration after investigating three values 2, 3, and 5 for the tournament size. As for QGS, the tuning procedure and its results are discussed in detail in the next subsection.

To evaluate the balance between exploitation and exploration capabilities and perform a comparison, two metrics have been used, namely the average fitness value and the average genetic population diversity. Moreover, a convergence study is carried out in terms of the average convergence rate as described in ([83]). In detail, the first metric, the average fitness value, is defined at the g-th iteration as:

$$\text{Avg\_fitness}(g) = \frac{\sum_{r=0}^{R-1} f_r^*(g)}{R} \tag{2.14}$$

where $f_r^*(g)$ is the best fitness value at the g-th iteration (in our experimentation it is the maximum value) of the r-th run and R is the number of the runs executed. By using this metric, the average convergence rate R, defined in ([83]), is computed as follows:

$$R = 1 - \left(\left| \frac{opt - \text{Avg\_fitness}(num\_iters)}{opt - \text{Avg\_fitness}(0)} \right| \right)^{\frac{1}{num\_iters}} \tag{2.15}$$

where opt is the absolute optimum of the test function and num_iters is the last generation. The rate represents a normalized geometric mean of the reduction ratio of the fitness difference per generation. The larger the convergence rate, the faster the convergence.

Finally, to evaluate the exploration capabilities of the GAs, it has been considered the average genetic population diversity defined at the g-th iteration as follows:

$$\text{Avg\_diversity}(g) = \frac{\sum_{r=0}^{R-1} \sum_{i=0}^{K-1} \sum_{j=0 \wedge i \neq j}^{K-1} H(C_i^r(g), C_j^r(g))}{R \cdot K \cdot (K-1) \cdot n} \tag{2.16}$$

where H is the Hamming distance, $C_i^r(g)$ is the i-th chromosome of the population at the g-th iteration in the r-th run, K is the population size, n is the size of the chromosome, R is the number of the runs executed. In our experimentation, R is set to 20. Considering the execution of a set of runs is necessary due to the non-deterministic nature of genetic algorithms. Moreover, in order to carry out a fair comparison, the same initial random population is used for the different genetic algorithms in the r-th run. However, different initial random populations are used in the different runs.

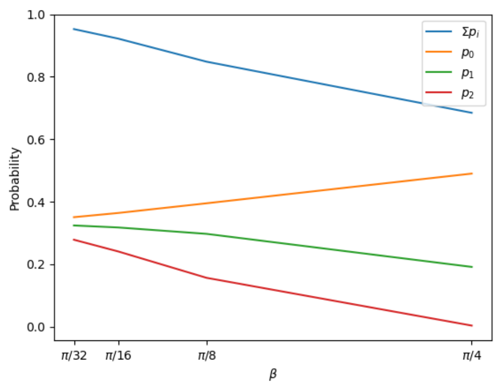
As for the statistical comparison carried out to investigate the significance of the impact of the proposed quantum genetic operator on the genetic algorithms' performance, it is conducted similarly to the work in ([76]) by applying the non-parametric statistical test known as Wilcoxon signed-rank test. In general, this test aims to detect significant differences between two sample means, where the two sample data represent the behavior of two algorithms. The underlying idea of this test is not just making a count of the wins of each compared algorithm but ranking the differences between the performance and developing the statistic over them ([58]). In our statistical comparison, the Wilcoxon test is used to conduct pairwise comparisons among the genetic algorithm equipped with QGS and genetic algorithms equipped with traditional selection operators. Each sample used in the statistical comparison is composed of the best fitness values obtained at the last generation in the executed runs (for this reason, the size of the sample is 20). Finally, in our experimentation, the Wilcoxon test is adopted by considering the most typical level of significance which is $\alpha = 0.05$.

All compared genetic algorithms are implemented in Python and run on a classical computer equipped with an Intel i9 processor and 128 GB of RAM. To investigate the robustness of QGS, it has been tested both on a noisy simulator and on a real quantum computer. In detail, genetic algorithms equipped with QGS for solving binary problems reported in Table 2.1 have been run by using a noisy simulator, namely *Fake Montreal*, made available by Qiskit™. On the other hand, genetic algorithms equipped with QGS for optimizing continuous functions reported in Table 2.3 have been tested by using real quantum computers made available by the IBM Q Experience platform via the cloud, namely *ibmq jakarta* (equipped with 7 qubits).

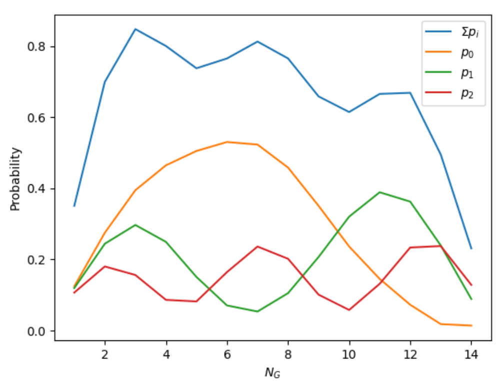
Discussion on binary optimization problems

In this paragraph, the results obtained by testing the performance of a genetic algorithm equipped with QGS for the considered binary optimization problems are reported. The paragraph is structured as follows: firstly the procedure of tuning of QGS is analyzed; secondly, genetic algorithms equipped with different selector operators are compared in terms of the metrics presented in Eqs. (2.14) and (2.16); finally, a statistical comparison is carried out to prove the significance of the obtained results.

As introduced in Section 2.1.1.III, QGS depends on three hyperparameters, namely t, β and N_G. As aforementioned, t should be chosen much smaller than 2^n, where n represents the size of the considered binary strings. At the same time, it should not be too small, because this leads to the loss of population diversity. Since, our experiments involve $n = 6$, setting $t = 3$ is a reasonable choice. Indeed, in this way, t is much smaller than 2^n, but, at the same time, it is not too small. The hyperparameter N_G is set according to (2.1). Therefore, $N_G = 3$. Finally, to set the value of β it has been carried out an evaluation of how to change the distribution of probability encoded in the quantum state by means of quantum amplitude amplification varying β. As aforementioned, $\beta \in \left[0, \frac{\pi}{t+1}\right]$. Hence, the tuning investigates the following set of values: $\frac{\pi}{32}$, $\frac{\pi}{16}$, $\frac{\pi}{8}$ and $\frac{\pi}{4}$. Fig. 2.5a shows the probabilities of the t marked states (together with their sum displayed in blue) against the investigated β values for the cases $N_G = 3$. By analyzing this plot, $\frac{\pi}{8}$ was chosen as the value for β since this value leads to a high probability (higher than 80%) to measure a marked state with respect to the other ones, and, at the same time, a strong difference among the measurement probabilities of each marked states. To check the validity of using $N_G = 3$, we compute the probability to measure the t marked states as N_G varies when $\beta = \frac{\pi}{8}$ (see Fig. 2.5b). As shown in Fig. 2.5b, the value $N_G = 3$ corresponds to the maximum value for the sum of the probabilities of the t marked states (blue line). Hence, $N_G = 3$ corresponds to the maximum probability of measuring one of the desired states and, as a consequence, it results correctly set.



(a) Probability vs β for $N_G = 3$



(b) Probability vs N_G for $\beta = \frac{\pi}{8}$

Figure 2.5: Graphical representation of how the probability values of measuring $t = 3$ marked states change against β values in (a) and against N_G values in (b).

Let's move to the analysis of the results of the comparison between QGS and the

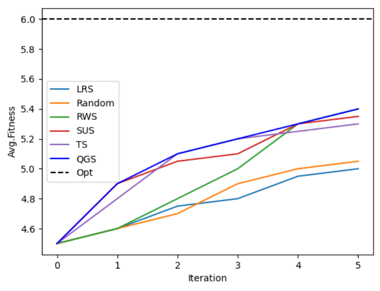
Table 2.5: Average convergence rate for each tested binary problem. The best R values are reported in bold.

	RWS	**SUS**	**TS**	**LRS**	**RS**	**QGS**
f_1	**0.14**	0.13	0.12	0.07	0.07	**0.14**
f_2	0.15	0.16	0.08	0.09	0.08	**0.26**
f_3	0.02	0.03	0.07	0.04	0.05	**0.13**
f_4	0.02	0.02	0.02	0.05	0.02	**0.27**
f_5	0.05	0.04	0.06	0.05	0.05	**0.17**
f_6	0.005	0.01	0.005	0.01	0.003	**0.12**
f_7	0.06	0.05	0.05	0.05	0.04	**0.13**

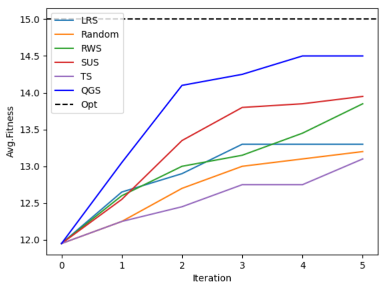
traditional selector operators after the tuning procedures are reported. The plots in Fig. 2.6 show the average fitness values metric against the number of generations for all considered benchmark functions. The dotted line in each plot reports the real optimum of the related function. As it is possible to see, the average fitness value for the genetic algorithm equipped with QGS is better than those obtained with a genetic algorithm equipped with classical selector operators for all considered benchmark functions except for f_1 where the average fitness value is similar to RWS. Moreover, an analysis of the convergence of the GAs in terms of the average convergence rate R described in (2.15) is reported in Table 2.5. As shown, GAs equipped with QGS converge faster than GAs equipped with classical selection operators for all the benchmark functions.

To complete the investigation of the QGS performance, the average diversity in the genetic population during the genetic algorithm generations has been analyzed and displayed in Fig. 2.7 for all the considered benchmark functions. As shown, different from the traditional selector operators, the designed genetic algorithm equipped with QGS maintains a constant degree of diversity within the genetic population for all the considered benchmark functions.

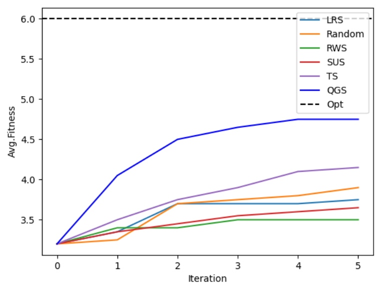
Finally, Table 2.6 shows the statistical significance of the Wilcoxon test expressed by p-values for the pairwise comparisons involving QGS and another approach. In detail, the one-sided version of the test is considered where the null hypothesis is the equivalence of the compared algorithms and the alternative one states, instead, that QGS is better than the compared approach. The null hypothesis is rejected when



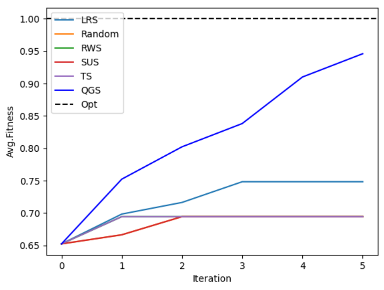
(a) f_1



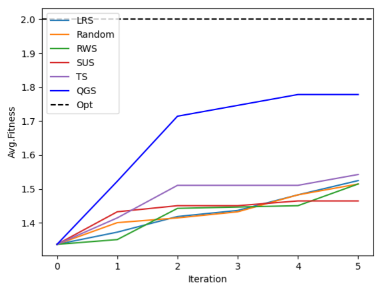
(b) f_2



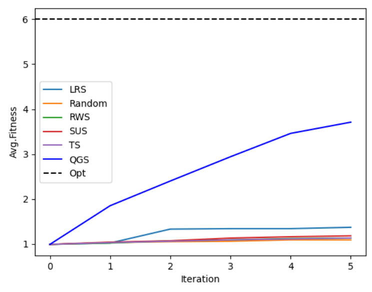
(c) f_3



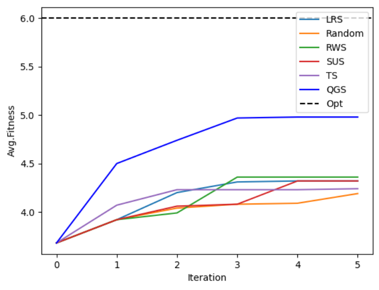
(d) f_4



(e) f_5

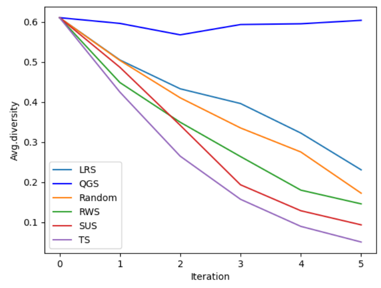


(f) f_6

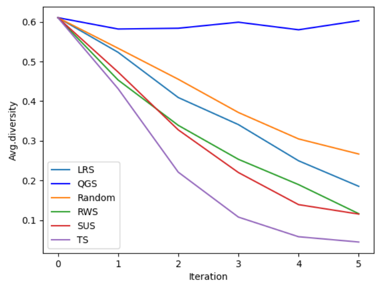


(g) f_7

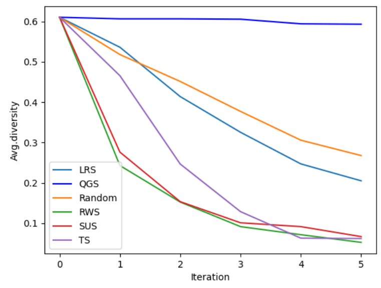
Figure 2.6: Average fitness values against the number of iterations for all the compared approaches for the function (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5, (f) f_6, (g) f_7.



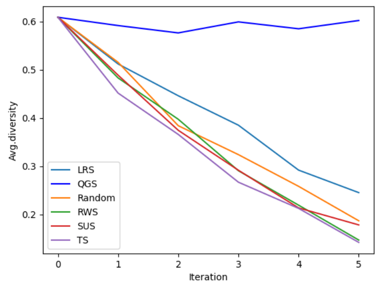
(a) f_1



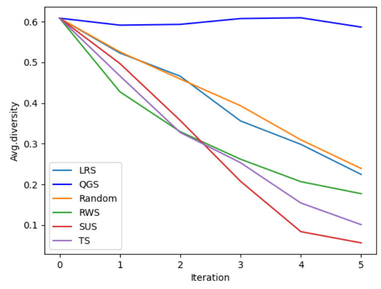
(b) f_2



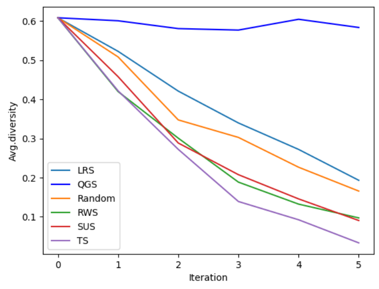
(c) f_3



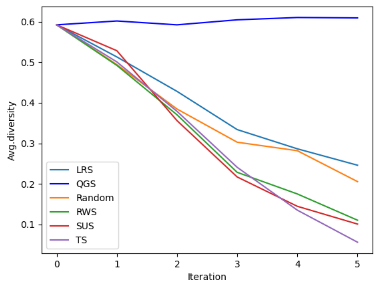
(d) f_4



(e) f_5



(f) f_6



(g) f_7

Figure 2.7: Diversity values against number of iterations for all the compared approaches for the function (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5, (f) f_6 and (g) f_7.

the reported p-value is less than the typical significance level $\alpha = 0.05$. Therefore, rejecting the null hypothesis in the pairwise comparisons leads to the state that QGS statistically outperforms the compared approach at a 95% confidence level. As shown

Table 2.6: One-sided Wilcoxons Statistical Significance (p-values) for QGS versus the other compared selectors for binary optimization problems

	RWS	**SUS**	**TS**	**LRS**	**RS**
f_1	0.6128	0.5000	0.4033	0.0469	0.0715
f_2	0.0356	0.0486	0.0057	0.0020	0.0100
f_3	0.0030	0.0045	0.0332	0.0129	0.0271
f_4	0.0001	0.0001	0.0001	0.0017	0.0001
f_5	0.0044	0.0002	0.0064	0.0022	0.0022
f_6	0.0012	0.0012	0.0008	0.0001	0.0004
f_7	0.0878	0.0400	0.0209	0.0283	0.0108

in Table 2.6, the null hypothesis is always rejected except for the pairwise comparisons involving RWS, SUS, TS, and RS for the function f_1 and RWS for the function f_7. Hence, QGS statistically outperforms at 95% confidence level all compared selectors for the functions f_2, f_3, f_4, f_5 and f_6. Moreover, QGS statistically outperforms at 95% confidence level all compared selectors except for RWS for the functions f_7. Finally, QGS statistically outperforms at 95% confidence level LRS for the function f_1, and, it is characterized by similar performance with respect to the other selectors.

Discussion on continuous optimization problems

This paragraph is devoted to presenting the results of the experimentation carried out in solving continuous optimization problems by using genetic algorithms equipped with QGS and comparing their performance to genetic algorithms equipped with classical selection operators already used in the experimentation carried out on binary optimization problems. The structure of this paragraph is the same as the previous one.

As for the binary optimization, also in this case QGS needs to be tuned. Also in this case, the hyperparameter t has been set to the value $t = 3$, for the reasons explained in the tuning procedure of previous experimentation. Following the same steps once fixed t, the procedure selects the most opportune β value among different values by considering $t = 3$ and N_G set by following the formula in (2.1). Therefore,

$N_G = 2$ for $n = 5$. As aforementioned, $\beta \in \left[0, \frac{\pi}{t+1}\right]$. Hence, the tuning investigates the following set of values: 0, $\frac{\pi}{16}$, $\frac{\pi}{8}$ and $\frac{\pi}{4}$. Fig. 2.8 shows the probabilities of the t marked states (together with their sum displayed in blue) against the investigated β value. By analyzing these plots, the values $\beta = \frac{\pi}{4}$ was selected. In detail, $\beta = \frac{\pi}{4}$ leads to a high probability (about 80%) to measure a marked state with respect to the other ones, and, at the same time, a strong difference among the measurement probabilities of marked states.

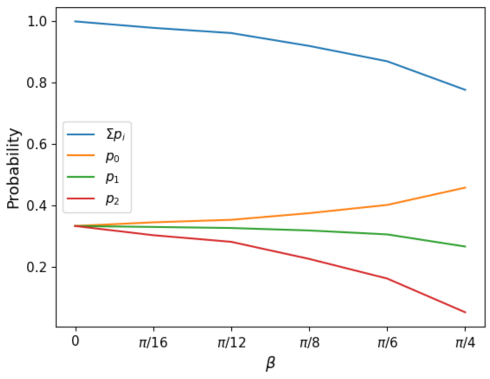


Figure 2.8: Graphical representation of how the probability values of measuring $t = 3$ marked states change against β values.

As for the previous tuning, once the values for t and β were fixed, the probabilities to measure the t marked states as N_G varies have been computed. Fig. 2.9 shows these probabilities. Differently from the previous case in which the value of N_G used to compute β corresponded to the maximum probability to measure one of the desired states, this does not occur. Therefore, a further tuning step was considered: different values for N_G were investigated in order to understand if there is a more opportune trade-off for the probabilities of the marked states. As aforementioned, it is not necessary to investigate too high values for N_G due to the periodic, even if imperfect, nature of the measurement probabilities of marked states. It is sufficient to take into account as an upper limit a value for N_G that shows a period for the measurement probability distributions of all marked states. For this reason, the investigated values of N_G are 2, 3, and 4. By considering these values, three configurations for QGS are tested as reported in Tables 2.7. In particular, the vertical lines in the plots in Fig. 2.9 highlight the probabilities of measuring each marked state for the investigated configurations of QGS.

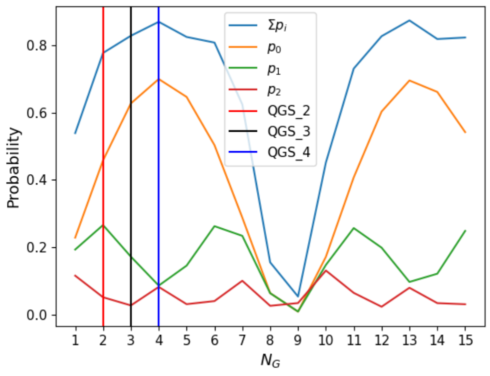


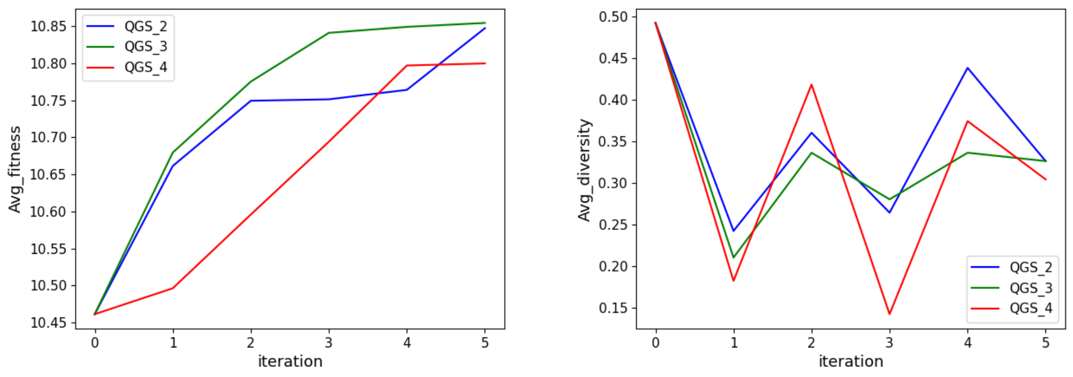
Figure 2.9: Graphical representation of how the probability values of measuring $t = 3$ marked states change against N_G values. The vertical lines represent the configurations of QGS tested.

Table 2.7: QGS hyperparameters for $n = 5$.

Name	**Hyperparameters**
QGS_2	$t = 3$, $\beta = \frac{\pi}{4}$, $N_G = 2$
QGS_3	$t = 3$, $\beta = \frac{\pi}{4}$, $N_G = 3$
QGS_4	$t = 3$, $\beta = \frac{\pi}{4}$, $N_G = 4$

In order to choose the best configuration for QGS, the tuning procedure has tested all the considered configurations of QGS, by including them, one at a time, in the genetic algorithm described in Table 1 instead of a traditional selection operator. The tests were conducted for all the benchmark functions reported in Table 2.3. The analysis has been carried out by taking into account both the average fitness metric and the genetic population diversity formalized in (2.14) and (2.16), respectively.

Figs. 2.10a and 2.10b show the average fitness values and the population diversity, respectively, against the number of iterations of the genetic algorithm applied to solve the function g_1. As shown, the three configurations QGS_2, QGS_3 and QGS_4, are characterized by a similar behavior in terms of the average fitness, whereas,



(a) Average fitness (b) Average diversity

Figure 2.10: Average fitness values in (a) and population diversity values in (b) against the number of iterations of the genetic algorithm applied to solve the benchmark function g_1.

QGS_2 is characterized by a better value for population diversity metric. Similar results are achieved for the other benchmark functions. Therefore, QGS_2 has been set as the best configuration for the case $n = 5$. This choice results also in a shallow quantum circuit with respect to the other QGS configurations.

Let us now move to discuss the results of the comparison between QGS and the traditional selector operators. The plots in Fig. 2.11 show the values of the average fitness value metric against the number of generations for all considered benchmark functions. The dotted line in each plot reports the function's real optimum. As it is possible to see, the average fitness value for the genetic algorithm equipped with QGS is generally greater than the one obtained with genetic algorithms equipped with classical selector operators for all considered benchmark functions. As for the binary problems, the average convergence rate R (see (2.15)) has been computed to evaluate the convergence velocity of the GAs equipped with QGS. Table 2.8 reports the obtained results. Also for the tested continuous problems, GAs equipped with QGS show a better convergence rate than GAs equipped with classical counterparts.

To complete the investigation of the QGS performance, the average diversity in the genetic population during the generations of the genetic algorithm has been an-

Table 2.8: Average convergence rate for each tested continuous problem. The best R values are reported in bold.

	RWS	**SUS**	**TS**	**LRS**	**RS**	**QGS**
g_1	0.06	0.04	0.07	0.07	0.03	**0.36**
g_2	0.09	0.02	0.04	0.04	0.06	**0.23**
g_3	0.10	0.07	0.14	0.07	0.05	**0.23**
g_4	0.04	0.02	0.02	0.05	0.08	**0.30**
g_5	0.09	0.10	0.04	0.09	0.05	**0.13**
g_6	0.05	0.04	0.02	0.06	0.05	**0.28**

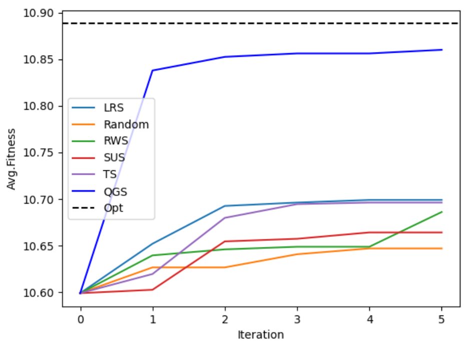
Table 2.9: One-sided Wilcoxons Statistical Significances (p-values) for QGS versus the other compared selectors for real optimization problems

	RWS	**SUS**	**TS**	**LRS**	**RS**
g_1	0.003222	0.001129	0.001196	0.002502	0.000349
g_2	0.007324	0.000122	0.000244	0.000244	0.008911
g_3	0.043384	0.003434	0.008606	0.003518	0.005575
g_4	0.000122	0.000238	6.10E-05	6.10E-05	0.002563
g_5	0.148682	0.188721	0.02045	0.189248	0.039063
g_6	0.000244	0.000122	0.000122	0.000244	0.000244

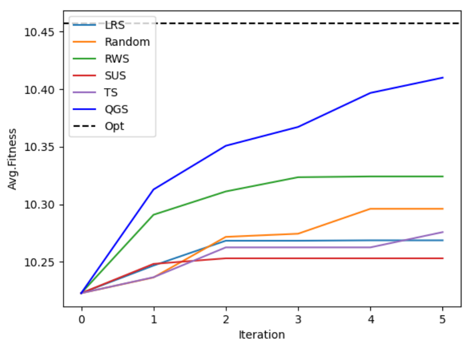
alyzed and displayed in Fig. 2.12 for all the considered benchmark functions. As shown, different from the traditional selector operators, the designed genetic algorithm equipped with QGS maintains a constant degree of diversity within the genetic population for all the considered benchmark functions.

Finally, Table 2.9 shows the statistical significances for QGS expressed by p-values computed by the Wilcoxon test. In detail, as for the tests carried out for the binary optimization problems, the one-sided version of the test is considered where the null hypothesis is the equivalence of the algorithms and the alternative one states, instead, that QGS is better than the compared approach. Rejecting the null hypothesis in Table 2.9 leads to the state that QGS statistically outperforms the compared approach at a 95% confidence level.

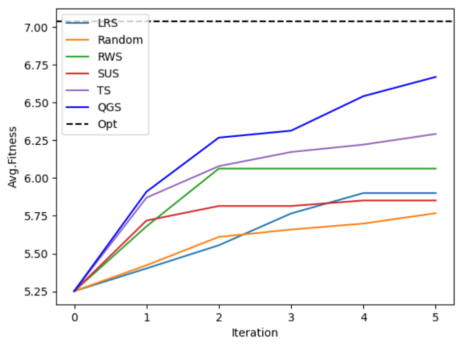
The genetic algorithm equipped with QGS outperforms all genetic algorithms equipped with traditional selection operators in all benchmark functions except for the benchmark function g_5. To support experimental reproducibility, data on which the statistical comparison has been carried out are available in the supplementary



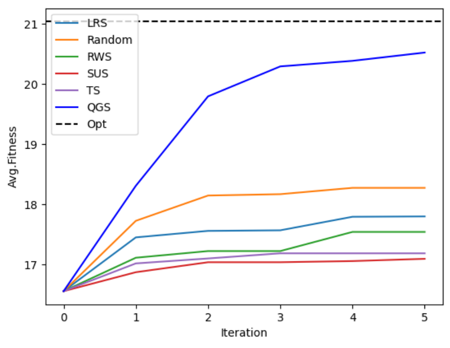
(a) g_1



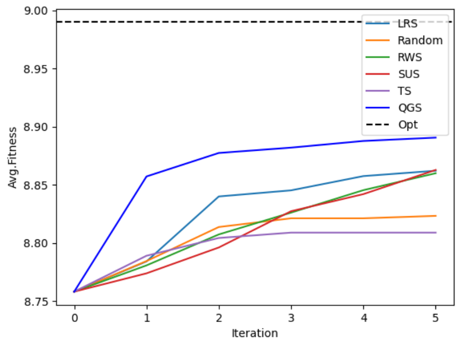
(b) g_2



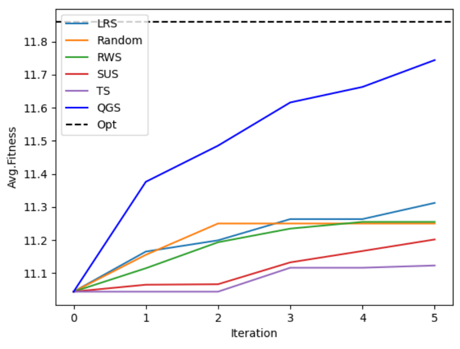
(c) g_3



(d) g_4



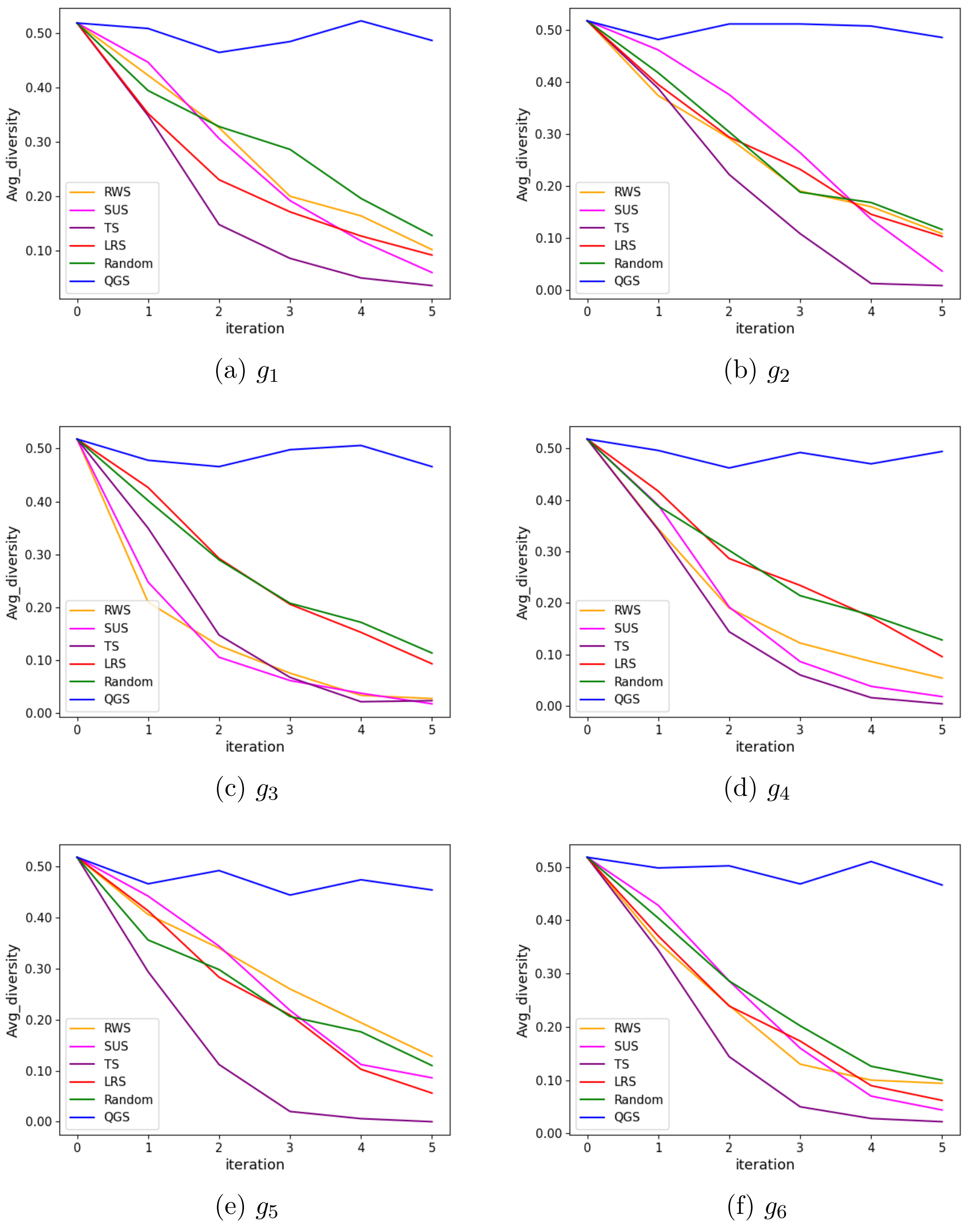
(e) g_5



(f) g_6

Figure 2.11: Average fitness values against the number of iterations for all the compared approaches in the case $n = 5$ for the function (a) g_1, (b) g_2, (c) g_3, (d) g_4, (e) g_5 and (f) g_6.

material of this paper.



(a) g_1 (b) g_2 (c) g_3 (d) g_4 (e) g_5 (f) g_6

Figure 2.12: Diversity values against number of iterations for all the compared approaches in the case $n = 5$ for the function (a) g_1, (b) g_2, (c) g_3, (d) g_4, (e) g_5 and (f) g_6.

2.1.1.V Summary

In this section, an innovative genetic operator, namely QGS, has been proposed. It exploits a modified version of the quantum amplitude amplification algorithm to define a quantum state capable of encoding genetic individuals and adjusting their probability of being included in the mating pool according to their fitness values. It is possible to summarize the main features of QGS as follows:

- QGS allows filling the mating pool by adding not only the best individuals of the current genetic population but also individuals that are not present in it with a non-zero probability. This leads to adding new genetic material into the population, reducing the likelihood of converging in local optima, and reaching a better trade-off between exploitation and exploration;
- QGS is able to control the degree of population diversity thanks to the opportune choice of its hyperparameters;
- QGS can be natively used for both minimization or maximization problems, and also when fitness functions are characterized by negative values. In other words, QGS can be directly used to solve different kinds of problems without performing specific conversions such as the scaling often used for RWS ([169]).

As shown in the experimental results involving several benchmark functions, the behavior of genetic algorithms based on QGS is statistically better than genetic algorithms equipped with well-known selection methods such as RWS, TS, SUS, LRS, and RS in the major part of the considered test cases. Moreover, the good trade-off between exploitation and exploration has been shown by reaching good solutions, and at the same time, maintaining a constant population genetic diversity. Although QGS has been tested using embryonic quantum computers characterized by a limited number of qubits and non-negligible noise, the rapid evolution of quantum technologies gives hope that QGS can soon be used on reliable, larger quantum machines. In future works, further investigations will be conducted to use QGS in no binary-coded genetic algorithms, use QGS in conjunction with quantum mating operators, and test the performance of QGS in solving more complex problems belonging to different real-world applications.

2.1.2 Quantum Mating Operator

If on the one hand, as seen in Section 2.1.1, a selection operator aims to choose the 'best' solutions in the population, on the other hand, crossover and mutation drive the GA in mating and modifying these solutions for creating new ones. The better choice of these processes and the better selection of their parameters enhance the performance of the GA [69, 170]. In particular, crossover and mutation operators have the crucial role of balancing exploration and exploitation of the GA, preventing its premature convergence to local optima. Over the years, many types of these operators have been proposed according to the characteristics of the problem to face [165, 162, 79]. The aim of this research is to enrich the literature with a quantum algorithm implementing a new mating operator that combines the crossover and the mutation steps, namely the Quantum Mating Operator (QMO). The proposed operator is designed specifically for binary-encoded problems, and it acts by considering the following steps: it identifies allele frequency patterns from individuals collected by using conventional selection operators and encodes these patterns through a quantum state; this state is mutated by means of rotational quantum gates; finally, a quantum measurement collapses the mutated quantum state to a binary-encoded chromosome, which will become part of the forthcoming genetic population. QMO can be classified as a multi-parent mating operator [70] because it performs the mating process by taking into account the entire set of selected parents. It is important to highlight that the proposed operator differs from other crossover operators proposed for mating individuals in quantum-inspired genetic algorithms (QIGAs) [128], a new class of algorithms that uses quantum computing concepts to manipulate solutions in a different way from classical GA. Indeed, QIGAs are classical algorithms, which simply are 'inspired' by certain principles of quantum mechanics, such as standing waves, interference, and coherence [128] but they are computed classically. On the contrary, QMO is a quantum operator designed to be executed on real noisy quantum computers and to be embedded in classical GAs in order to fully exploit the computational benefits of quantum computation in evolutionary optimization. Experimental studies have been carried out to show the suitability of QMO in being embedded in a genetic optimization process. This study involves a set of well-known benchmark functions used to test the performance of the proposed quantum operator against traditional mating methods in terms of the obtained solution quality. Moreover, a statistical comparison procedure based on the Wilcoxon signed rank test [179] has been carried out to investigate the impact of the proposed quantum genetic operator on the performance of genetic algorithms. The section is structured as follows. Firstly, preliminary concepts about the classical mating operators used for the experimental comparison with QMO are described; then the QMO algorithm is introduced and discussed; finally, the experimental results are shown and statistically evaluated.

2.1.2.I Related Work

The following digression will focus on crossover operators that operate on binary-encoded individuals of length n. In this context, one of the most used crossover operators is the *1-point crossover*. Starting from two parents, it randomly selects any crossover point p_i (with i from 0 to $n-1$), then two offspring are created by combining the parents at the crossover point. In (2.17) it is reported an example of 1-point crossover carried out on two parents of length $n = 6$ and using the point between the 2-th and the 3-th gene as crossover point.

$$\begin{array}{rl} \text{parent 1} & 100|110 \\ \text{parent 2} & 001|010 \\ \text{offspring 1} & 100|010 \\ \text{offspring 2} & 001|110 \end{array} \tag{2.17}$$

Another widely used operator is the *2-point crossover*. It selects two crossover points to create the offspring similarly to the 1-point crossover. As an example, (2.18) reports the creation of two offspring using as the two crossover points the one between the 1-th and the 2-th gene and the one between 3-th and 4-th gene of parents.

$$\begin{array}{rl} \text{parent 1} & 10|01|10 \\ \text{parent 2} & 00|10|10 \\ \text{offspring 1} & 10|10|10 \\ \text{offspring 2} & 00|01|10 \end{array} \tag{2.18}$$

A different crossover operator is the *uniform crossover*, firstly introduced in [165]. It selects two parents from the parent set and then it creates a bit string mask. The bits of offspring are duplicated from one of the parents according to the bits of their mask. If it is a 0 in the binary crossover mask, the bit is copied from the second parent otherwise from the first parent. The second offspring is obtained by using the complementary of the original mask. In (2.19) it is reported an application of the uniform crossover.

$$\begin{array}{rl} \text{parent 1} & 100110 \\ \text{parent 2} & 001010 \\ & \\ \text{mask} & 100010 \\ & \\ \text{offspring 1} & 101010 \\ \text{offspring 2} & 000110 \end{array} \tag{2.19}$$

If on the one hand, the crossover is a genetic operator used to combine the genetic information of two parents to generate new offspring, on the other hand, mutation is a genetic operator used to increase genetic diversity in the offspring from one generation to the next. In the case of chromosomes represented as binary strings, the most common mutation strategy is to use the *bit-flip mutation* operator. It flips genes of a chromosome according to a given probability.

Let us move on to our proposal to evolve population solutions by exploiting a properly defined quantum algorithm.

2.1.2.II QMO Workflow

Firstly, let us analyze how the workflow of a classical genetic algorithm (see Fig. 1.6) changes using QMO. As shown in Fig. 2.13, it replaces the crossover and mutation operators, combining them in a quantum algorithm.

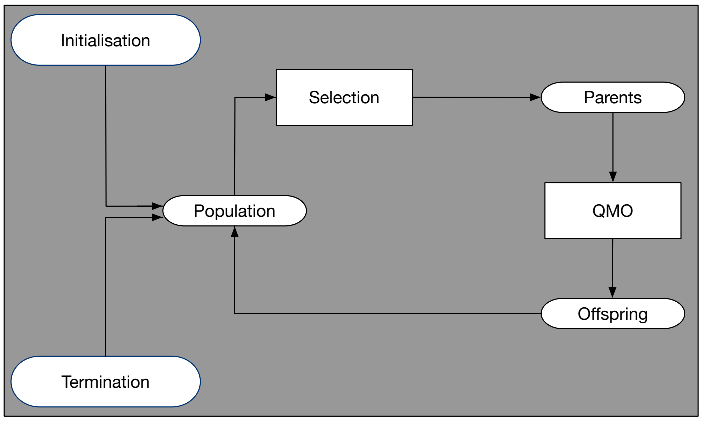


Figure 2.13: Workflow of a GA equipped with QMO

QMO pseudo-code is reported in Algorithm 2 and it is hereafter described:

1 Let $\mathcal{P} = \{I_0, I_1, \ldots, I_{n-1}\}$ be the set of parents in a given generation of a GA. Each I_j with $j \in [0, n-1]$ is a binary encoded individual of length m. The first step carried out by QMO is to randomly select an ensemble $\mathcal{S}_\mathcal{P}$ of parents from $\mathcal{P}$ as follows: for each $I_j \in \mathcal{P}$ a random number $p \in [0, 1]$ is generated if $p < p_c$ then I_j is added to $\mathcal{S}_\mathcal{P}$. The individuals in $\mathcal{S}_\mathcal{P}$ are mated using QMO, while the remaining are inserted directly into the offspring set.

2 Then, QMO computes the frequencies of '1' bit by bit for the elements in $\mathcal{S}_\mathcal{P}$. For instance, if $m = 4$ and $\mathcal{S}_\mathcal{P} = \{0110, 1010\}$ then the four frequencies computed are $[0.5, 0.5, 1, 0]$. Let us denote the frequency of '1' in the i-th gene of the chromosomes in $\mathcal{S}_\mathcal{P}$ as f_i, where $i \in [0, m-1]$.

Algorithm 2 Pseudo-code of the proposed QMO

```
Require: set of parents $\mathcal{P}$, crossover probability $p_c$, mutation probability $p_m$.
 1: $\mathcal{S}_{\mathcal{P}} \leftarrow$ RandomSampling($\mathcal{P}$, $p_c$);
 2: $offspring \leftarrow \mathcal{P} - \mathcal{S}_{\mathcal{P}}$;
 3: $\mathcal{O} \leftarrow$ ComputeOneFrequencies($\mathcal{S}_{\mathcal{P}}$);
 4: while ( $size\ offspring \neq size\ \mathcal{P}$) do
 5:     $qc \leftarrow$ CreateQuantumCircuit();
 6:     ApplyCrossoverRy($qc$, $\mathcal{O}$);
 7:     ApplyMutationRy($qc$, $p_m$);
 8:     $ind \leftarrow$ MeasureQuantumCircuit($qc$);
 9:     AddIndToOffspring($ind$, $offspring$);
10: end while
11: return $offspring$;
```

3 At this point, a quantum circuit composed of m qubits is initialized with all the qubits in the state $|0\rangle$. For each qubit, a $R_y(\theta)$ gate is performed. In detail, the angle used in the R_y gate carried out on the i-th qubit is $\theta_i = \pi f_i$, with $i \in [0, m-1]$. An implementation of this step indicated as R_y *crossover step*, is reported in Fig. 2.14, where the case with $m = 4$ has been considered and the frequencies computed in step 2 were used.

4 Successively, m random numbers in $[0, 1]$ are generated. Denoting with q_i the i-th number obtained, if $q_i < p_m$, then a second $R_y(\theta)$ gate is applied on the i-th qubit of the circuit. This procedure, belonging to the R_y *mutation step*, is iterated for all the values of $i \in [0, m-1]$. The angle for each applied rotation is random in the range $[0, \pi]$. For instance, in the $m = 4$ example, let us suppose that q_1 and q_3 are lower than the threshold p_m, then this step is implemented as reported in Fig. 2.14 in the R_y Mutation block, where θ_{r1} and θ_{r3} are the two randomly computed angles.

5 Finally, the quantum circuit is measured to obtain a new chromosome that is added to the offspring.

6 Steps 3-5 are repeated until the desired size of the offspring is reached.

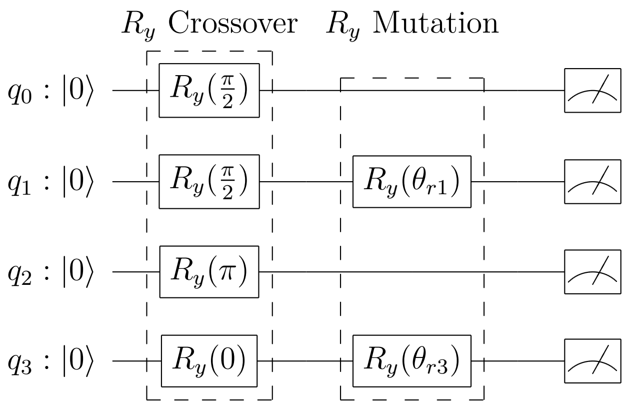


Figure 2.14: Circuital representation of QMO for the generation of a 4-dimensional chromosome.

2.1.2.III Experiments and Results

This paragraph is devoted to showing the results of an experimental study carried out to highlight the suitability of QMO to be used as a mating operator in GAs and study its impact on the performance yielded by the algorithm. To achieve this goal, QMO is compared with the well-known crossover and mutation methods presented in sec. 2.1.2.I.

The considered classical mating operators consist of the following pairs of operators: One-Point Crossover (OPC) and bit flip mutation, Two-Point Crossover (TPC) and bit flip mutation, Uniform Crossover (UC) and bit flip mutation. Because bit flip is used in all the classical mating strategies from now on, each pair of operators will be indicated only by the crossover operator. Both quantum mating and classical operators are embedded one at a time in a conventional binary-encoded GA applied to solve a set of typical benchmark functions. Firstly, a comparison in terms of solution quality provided by each genetic algorithm has been carried out. Secondly, a statistical procedure has been conducted to evaluate the significance of the impact of the proposed QMO on the performance of GAs.

Benchmark Functions

The experimental study involves a set of benchmark functions well-known in the literature. They are n-dimensional binary optimization problems with one or more global optima. The functions are reported in Table 2.10. In detail, f_1, f_2 and f_3 are used in [67], whereas f_4 and f_5 are introduced in [185].

Table 2.10: Benchmark functions

Function definition
$f_1(x) = \sum_{i=1}^{n} x_i$
$f_2(x) = \sum_{i=1}^{n} i \cdot x_i$
$f_3(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$
$f_4(x) = \sum_{i=0}^{4} g_{i+i\cdot 4}^{4+i+i\cdot 4};\ g_i^k = \begin{cases} 1 & \text{if } u_i^k(x) = 0 \text{ or } 5 \\ 0 & \text{if } u_i^k(x) = 1 \text{ or } 4 \\ 0.360384 & \text{if } u_i^k(x) = 2 \text{ or } 4 \\ 0.640576 & \text{if } u_i^k(x) = 3 \end{cases}$
$f_5(x) = \sum_{i=0}^{4} g_{i+i\cdot 4}^{4+i+i\cdot 4};\ g_i^k = \begin{cases} 1 - 0.2 \cdot u_i^k(x) & \text{if } u_i^k(x) \leq 4 \\ 5 & \text{if } u_i^k(x) = 5 \end{cases}$

The notation $u_i^j(x)$ used in the functions f_4 and f_5, represents the number of ones in the solution sub-string that goes from the i-th bit to the j-th bit where $j > i$. Formally:

$$u_i^j(x) = \sum_{k=i}^{j} x_k. \tag{2.20}$$

All benchmarks have been considered as maximization problems. Due to the limited number of qubits characterizing the current quantum hardware, and the impossibility of simulating quantum noise circuits with a high number of qubits, our experimentation considers as problem dimension the case $n = 25$. However, this is not a limitation of our approach that will be able to be applied to benchmark functions characterized by a higher dimension as soon as IBM quantum computers characterized by a larger number of qubits (e.g. IBM Q Brooklyn with 65 qubits) will be made available via the cloud.

Experimental setup

The experimental study consists of running a traditional binary-encoded GA with the generational approach equipped with a different mechanism for mating individuals (QMO, OPC, TPC, and UC) at a time. For the sake of reproducibility, the pseudo-code of the implemented GA is reported in Algorithm 1.

Regarding the hyper-parameters of each GA, two choices are made. Since the goal of each designed GA is not to be the best optimization solver, but, to be a casing for the compared operators, for hyper-parameters that are not involved in the mating step, the tuning procedure wasn't carried out. Table 2.11 reports these values. In particular, the hyper-parameters *pop_size* and *num_gens* have been set in combination in order not to perform a number of fitness evaluations that is much less than of an exhaustive research of the search space (composed of 2^{25} possible solutions for $n = 25$). On the other hand, hyperparameters of classical mating operators

Table 2.11: Hyper-parameters of the designed GA

Hyper-parameter	**Value**
Population size *pop_size*	30
Number of iterations *num_iters*	10

were tuned with the aim of comparing QMO with the best possible configuration of classical operators. As the best configuration, it is intended the combination of hyper-parameters of the operator embedded in the GA which maximizes the metric in (2.21). In particular, for each benchmark function, different combinations of crossover and mutation probability have been tested in order to select the best two probabilities among the following set of crossover probability $\{0.9, 0.8, 0.7, 0.5\}$ and the following set of bit flip probability $\{0.05, 0.1, 0.2, 0.3, 0.5, 0.7\}$. Table 2.12 reports for each benchmark function the best values of crossover probability and mutation probability for each classical mating operator.

Table 2.12: Hyper-parameters of classical mating operators

	OPC		**TPC**		**UC**	
Function	p_c	p_m	p_c	p_m	p_c	p_m
f_1	0.8	0.05	0.8	0.1	0.9	0.1
f_2	0.8	0.05	0.9	0.1	0.9	0.05
f_3	0.9	0.2	0.9	0.05	0.8	0.05
f_4	0.7	0.7	0.8	0.2	0.9	0.7
f_5	0.8	0.05	0.8	0.05	0.9	0.1

On the other hand, the probability p_c and p_m selected for QMO are 0.8 and 0.25 respectively. Due to the long time required to simulate GAs equipped with QMO, no tuning procedure was conducted for these hyper-parameters. Therefore, the comparison shown in the next section has to be intended as a comparison between the optimal configurations of classical mating operators and a non-optimal configuration of QMO. Finally, in such a comparison the metric adopted was the average fitness value. In detail, it is defined at the g-th iteration as:

$$Avg\_fitness(g) = \frac{\sum_{r=0}^{R-1} f_r^*(g)}{R} \tag{2.21}$$

where $f_r^*(g)$ is the best fitness value at the g-th iteration (in our experimentation it is the maximum value) of the r-th run and R is the number of the runs executed. In our experimentation, R is set to 20. The execution of a set of runs is necessary due to the non-deterministic nature of GAs. Moreover, in order to carry out a fair comparison, the same initial random population is used for the different GAs in the r-th run. However, different initial random populations are used in the different runs.

As for the statistical comparison carried out to investigate the same significance of the impact of the proposed quantum genetic operator the same Wilcoxon signed rank test performed in Sec. 2.1.1.IV is reported. Each sample used in the statistical comparison is composed of the best fitness values obtained at the last generation in the executed runs (for this reason, the size of the sample is 20). Finally, in our experimentation, the Wilcoxon test is adopted by considering the most typical level of significance which is $\alpha = 0.05$.

All compared GAs are implemented in Python by means of the DEAP™ library[4]. The GA equipped with QMO is run by using a real quantum computer simulator, namely *Fake Sydney*, made available by Qiskit™. All the different GAs are run on a classical computer equipped with an Intel i9 processor and 128 Gb of RAM.

Results of the study

The results of the comparison between QMO and the other classical mating operators are hereafter described. Firstly, the plots in Fig. 2.15 report the values of the average fitness metric (see (2.14)) against the number of generations for all considered benchmark functions. As shown, the average fitness value for the genetic algorithms equipped with QMO is generally better than the one obtained by GAs equipped with classical mating operators, except for function f_4, where the performance of QMO is slightly lower than OPC. Secondly, the set of the 20 best fitness values reached by each GA at the last generation of each execution has been reported as a box plot in Fig. 2.16 for each benchmark function tested. The boxes are created from the first quartile to the third quartile of each set of solutions. The yellow lines refer to the median value of the related set, while the whiskers extend from the boxes to the maximum and minimum values of the set. Finally, outliers (represented as empty circles) are solutions that differ significantly from the rest. As shown in Fig. 2.16 the third quartile of the set of solutions found using QMO is always higher than the first quartile of the solutions found using the classical mating operators, for each of the functions tested except for f_4, where the box plots related to QMO and OPC are comparable. This means that for all the functions tested, except for f_4, 75% of solutions found by using QMO are better than the 75% of solutions found by using the other mating operators. The same reasoning applies to the function f_4 except for the box plot related to OPC, which is very similar to the one related to

[4]https://deap.readthedocs.io/en/master/

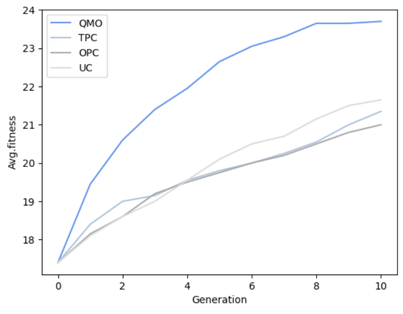
Table 2.13: One-sided Wilcoxons Statistical Significance (p-values) for QMO versus the other compared mating operators

	TPC	**OPC**	**UC**
f_1	0.0002	0.0001	0.0004
f_2	0.0002	0.0001	0.0003
f_3	0.0004	0.0003	0.0007
f_4	0.0708	0.6287	0.0007
f_5	0.0173	0.0087	0.0005

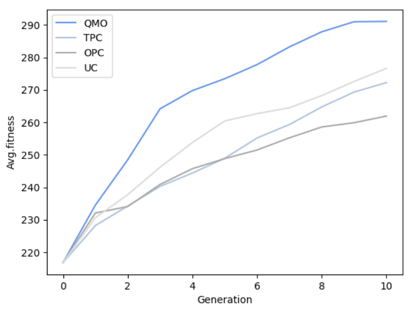
QMO. As the last evaluation of the achieved results, Table 2.13 shows the statistical significance of the Wilcoxon test expressed by p-values for the pairwise comparisons involving QMO and another mating operator. In detail, the one-sided version of the test is considered where the null hypothesis is the equivalence of the compared algorithms and the alternative one states, instead, that QMO is better than the compared approach. The null hypothesis is rejected when the reported p-value is less than the typical significance level $\alpha = 0.05$. Therefore, rejecting the null hypothesis in the pairwise comparisons leads to the state that QMO statistically outperforms the compared approach at a 95% confidence level. As shown in Table 2.13, the null hypothesis is always rejected except for the pairwise comparisons involving OPC and TPC for the function f_4. Hence, QMO statistically outperforms at 95% confidence level all compared mating operators for the functions f_1, f_2, f_3 and f_5, while it shows similar performance with respect to OPC and TPC for the function f_4.

2.1.2.IV Summary

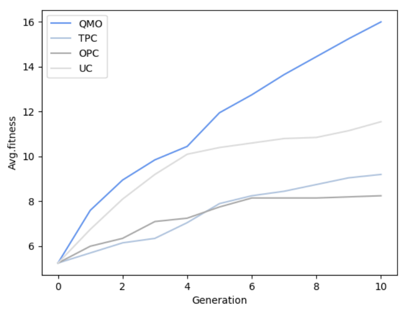
In this study, an innovative genetic operator, namely QMO, has been proposed. It exploits quantum rotational gates to define a quantum state capable of encoding the main characteristics of a parent set, from which new solutions can be generated. QMO comprises two steps: in the first, a set of R_y gates is used to perform quantum multi-parents recombination, while in the second, another set of R_y gates is used to insert mutations in the mating operation. The main goal achieved by this work is proof that using the proposed quantum algorithm to perform recombination and



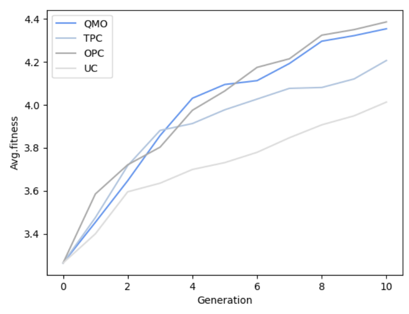
(a) f_1



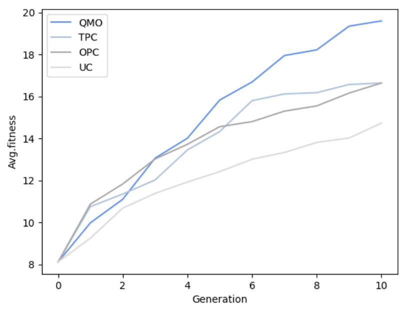
(b) f_2



(c) f_3

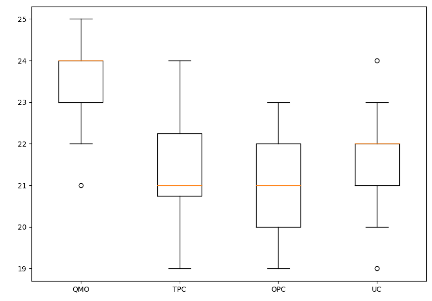


(d) f_4

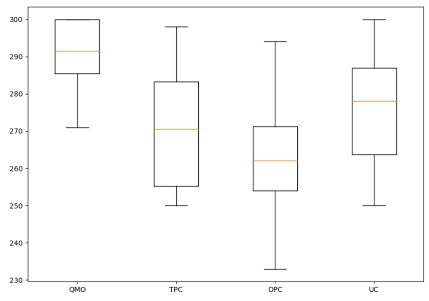


(e) f_5

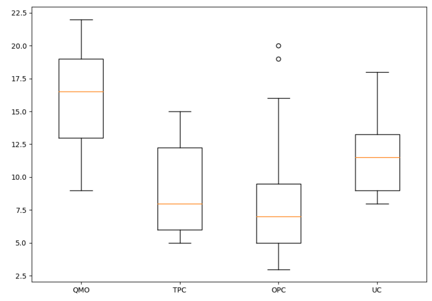
Figure 2.15: Average fitness values against number of iterations for all the compared approaches for the function (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5



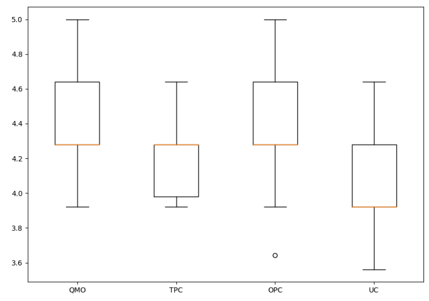
(a) f_1



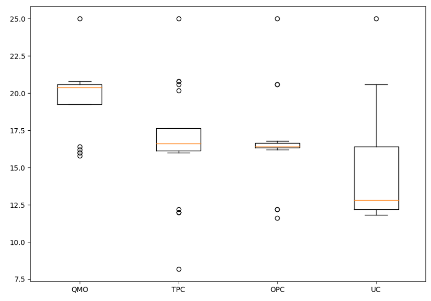
(b) f_2



(c) f_3



(d) f_4



(e) f_5

Figure 2.16: Box plots containing the 20 best solutions found by each GA for the function (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5. Each box is related to a GA equipped with a different mating operator.

genetic mutation in a classical GA improves the quality of the solutions found. Indeed, as shown in the experimental results involving several benchmark functions, the behavior of genetic algorithms based on QMO is statistically better than genetic algorithms equipped with well-known mating methods such as OPC, TPC, UC in the major part of the considered test cases. Also, the result achieved is important because it proves that genetic algorithms can be improved by means of quantum operators that can be executed even with current quantum hardware, which is very sensitive to noise [144] and on which it is currently impossible to compute entire quantum genetic algorithms reliably. Indeed, the structure of the quantum circuit implementing QMO is composed just of single-qubit quantum gates, which are much less prone to noise than multi-qubit quantum gates. In future works, further investigations will be conducted to use QMO in no binary-coded genetic algorithms and in conjunction with quantum selection operators and to test the performance of QMO in solving more complex problems belonging to different real-world applications.

2.1.3 On The Effect of Quantum Noise in Genetic Algorithms

Both QGS and QMO show excellent performance when run or simulated on NISQ devices that are affected by high noise levels. Considering this, a more formal study was conducted to investigate how quantum noise levels affect the performance of GAs equipped with quantum operators [12]. This study has been limited to GAs equipped with QMO, mainly for computational reasons. In fact, this operator can be easily simulated even taking quantum noise into account, whereas since the quantum circuit implementing QGS is much more complex, such an analysis would have required much more computational effort. In this scenario, the proposed analysis takes into account two different sources of quantum noise, namely *readout error* and *single qubit quantum gate error*: the first is related to the probability of flipping the state of a qubit during the measurement operation; the second is related to the probability of incorrectly performing operations based on single-qubit quantum gates. The effect of these quantum noises on QMO-based genetic algorithms has been assessed by solving a well-known combinatorial optimization problem, the *0-1 knapsack problem* [154], as the problem size and quantum noise levels increased. In order to be able to use QMO-based genetic algorithms to solve the 0-1 knapsack problem on quantum computers characterized by a limited number of qubits, our study was carried out on a distributed quantum architecture of the D-NISQ type [5]. Experimental results show that for highly constrained problem instances, the effect of quantum noise can increase GA performance, while for weakly constrained problem instances, quantum noise excessively disturbs algorithm evolution, resulting in worse performance than the equivalent without noise. In the remaining of the section, it will be better discussed how quantum noise can be formalized from a mathematical point of view, then it will be shown how is it possible to distribute the QMO algorithm to enable faster simulations, and finally, it will be presented the experimental results obtained in this study.

2.1.3.I Quantum Noise

Quantum operations are not perfect and noise heavily affects their precision and accuracy [11]. Hereafter, a more detailed and formal description of different kinds of quantum noises is carried out. Firstly, to formally describe quantum noise, a different and richer representation of quantum states than just state vectors is required. Indeed, it is necessary to introduce the so-called *mixed quantum states*, which are used to describe, for example, a physical system that is subject to noise and decoherence or is entangled with other quantum systems in the environment outside of our control. Mixed quantum states are different from the *pure state* (see (1.1)) because they represent the state of a quantum system that is not with certainty in a given state but that can be with different probabilities in different states, which is what happens considering noise. Mathematically, the more general description of quantum states uses *density matrices* to describe both pure and mixed quantum states. For pure quantum states, the state-vector formalism and the density matrix formalism coincide: for each pure state $|\psi\rangle$ such as in (1.1), there is a density matrix that is an equivalent representation of the same state. In this case, the density matrix ρ is defined as the outer product of the state $|\psi\rangle$ and its conjugate transpose:

$$\rho = |\psi\rangle \langle\psi| \tag{2.22}$$

As an example, the density matrix ρ corresponding to the two-dimensional state reported in (1.1) can be written as:

$$\rho = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \begin{bmatrix} \alpha^\dagger & \beta^\dagger \end{bmatrix} = \begin{bmatrix} \alpha\alpha^\dagger & \alpha\beta^\dagger \\ \beta\alpha^\dagger & \beta\beta^\dagger \end{bmatrix} \tag{2.23}$$

On the other hand, a general mixed state ρ consisting of an ensemble of pure states $\{|\psi_j\rangle\}_{j=1}^n$, each with probability of occurrence $\{p_j\}_{j=1}^n$ is defined as:

$$\rho = \sum_j p_j |\psi_j\rangle \langle\psi_j| \tag{2.24}$$

For instance, suppose that a qubit in the state $|0\rangle$ undergoes a bit flip error with probability s, then the quantum state evolves from the pure state $\rho = |0\rangle\langle 0|$ to the following mixed state:

$$\rho' = (1-s)|0\rangle\langle 0| + s|1\rangle\langle 1|. \tag{2.25}$$

This final density matrix is a mixed state representing a qubit that is either in $|0\rangle$ or $|1\rangle$ with a probability distribution $\{(1-s), s\}$.

With this density matrix formalism, the representation of the action of unitary gates on quantum states changes. Let us consider a unitary gate U acting on a state

$|\psi\rangle$, with the statevector formalism the new state $|\phi\rangle$ is obtained as $|\phi\rangle = U|\psi\rangle$. Therefore the new density matrix ρ' can be retrieved as:

$$\rho' = |\phi\rangle\langle\phi| = U|\psi\rangle\langle\psi|U^\dagger = U\rho U^\dagger \tag{2.26}$$

Considering the density matrix formalism, the action of quantum noise can be easily introduced from a mathematical point of view. First of all, let's define as *quantum channel* a general operation on a quantum state. To represent quantum channels mathematically, Kraus decomposition can be used. A quantum channel $\mathcal{N}$ acting on a density matrix ρ can be decomposed as

$$\mathcal{N}(\rho) = \sum_i s_i K_i \rho_i K_i^\dagger \tag{2.27}$$

where the K_i are Kraus operators describing the quantum channel, and the s_i is the probability that the operator represented by the Kraus operator K_i occurs. For instance, considering the action of a unitary gate U on ρ reported in (2.26), the corresponding quantum channel is given by $\rho' = \mathcal{N}_U(\rho) = U\rho U^\dagger$, i.e. there is only one Kraus operator $K_1 = U$ and it is applied with probability $s_1 = 1$. As for the density matrix representing the action of a bit flip error reported in (2.25), it can be obtained starting from $\rho = |0\rangle\langle 0|$ and applying the quantum channel $\mathcal{N}_{bitflip} = (1-s)\rho + sX\rho X$, where X is the quantum not gate. Therefore the Kraus operators for the bit flip quantum channel are $K_1 = I$ (the identity), and $K_2 = X$ (a bit flip), which are applied with probabilities (1-s) and s, respectively.

At this point, some quantum channels representing different kinds of quantum errors acting on single qubits are reported:

- **Bit Flip:** this error represents the flipping of a qubit flips from $|0\rangle$ to $|1\rangle$ or vice versa with some specified probability s. Bit flip errors are modeled by applying an X gate. Its Kraus representation is $\mathcal{N}(\rho) = (1-s)\rho + sX\rho X$.

- **Depolarizing:** This channel describes the process in which a 1-qubit quantum state loses its quantum information and decays into an incoherent mixture of the computational basis states, $|0\rangle\langle 0|$ or $|1\rangle\langle 1|$. The quantum state loses the phase as well as the superposition between basis states. The Kraus representation is $\mathcal{N}(\rho) = (1-s)\rho + s/3(X\rho X + Y\rho Y + Z\rho Z)$, where s represents the strength of the noise.

- **Phase Flip:** This error rotates with probability s the qubit around the Z-axis, affecting the phase of the qubit. Phase flips are modeled by applying a Z gate, and they map $|1\rangle$ to $-|1\rangle$. The Kraus representation is $\mathcal{N}(\rho) = (1-s)\rho + sZ\rho Z$.

- **Amplitude damping:** This channel represents the process through which the qubit state of higher energy $|1\rangle$ decays to the lower energy state $|0\rangle$. The Kraus representation is $\mathcal{N}(\rho) = K_1\rho K_1^\dagger + K_2\rho K_2^\dagger +$, where

$$K_1 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-s} \end{pmatrix} \quad K_2 = \begin{pmatrix} 0 & \sqrt{s} \\ 0 & 0 \end{pmatrix} \tag{2.28}$$

Other quantum channels representing errors affecting processes involving more than 1 qubit can be considered too, but as discussed in the following Section, they aren't relevant for the QMO algorithm.

2.1.3.II Noise Model in QMO

Current quantum computing libraries, such as Qiskit [1] or Pennyline [38], enable the simulation of quantum circuits with custom models of quantum noise. In detail, the offspring generation carried out by means of QMO depends on single R_y gates and quantum measurement operations. Therefore, the kinds of noise considered for building our quantum noise model consist of just *bit flip* and *depolarizing* quantum channels. This choice is justified by the fact that the bit flip channel models the noise that occurs during the quantum measurement operation (*readout error*), while the depolarizing channel models the noise in performing single qubit gate operations. Other noisy quantum channels such as *phase flip* and *amplitude damping* have been neglected, according to the following considerations: R_y gates performed in the QMO circuit don't affect the qubits phase and therefore a degeneration in quantum phase information is not relevant for QMO performance; typical decoherence times are on the order of hundreds of microseconds, while quantum rotations are executed in hundreds of nanoseconds, therefore with the short quantum circuit implementing QMO, amplitude dumping can be neglected too. According to the above analysis, QMO has been simulated considering different quantum noise models with different levels of bit flip and depolarizing strengths. In detail, denoting with s_{bf} the former probability and with s_d the latter, the following ranges have been investigated: $s_{bf} \in [0, 0.05, 0.1, 0.15, 0.2, 0.25]$ and $s_d \in [0, 0.001, 0.005, 0.007, 0.01, 0.02]$. Overall, 36 different noise models have been considered with different levels of the strength of the two quantum channels. The choice of these ranges for s_{bf} and s_d is motivated by actual quantum device data. In particular, as a reference IBM quantum devices were considered, which readout errors are on the order of $[10^{-3}, 10^{-1}]$ and which single qubit gate errors are on the order of $[10^{-3}, 10^{-2}]$.

2.1.3.III Experimental Analysis

Let us now describe the analysis carried out to assess how quantum noise can affect the performance of QMO. To perform such an analysis, QMO has been embedded in binary encoded GAs to solve the *0-1 knapsack problem* by considering different levels of quantum noise and different sizes of the problem solution space.

The knapsack problem is a constrained combinatorial optimization problem. This is a classical NP problem in operation research [36] and it has been used as a benchmark in several works [95, 171]. In the 01 knapsack problem, items with varying weights and respective values are given. A knapsack of a given capacity is given. The goal is to select objects in such a way that maximum values should be gained with available capacity. Formally, given N objects, where the $i-th$ object has weight w_i and value v_i, denoting with C the knapsack capacity, the objective function is defined as:

$$\begin{aligned} &\text{Maximize: } \sum_{i=1}^{N} v_i x_i \qquad x_i \in \{0,1\} \\ &\text{Subject to: } \sum_{i=1}^{N} w_i x_i \leq C \end{aligned} \tag{2.29}$$

Here, x_i has a value of 1 if it is selected else 0.

In our experiments the value and the weight of each object i were selected randomly in a range $[1, 100]$, while the capacity of the knapsack C has been defined as a certain percentage of the sum of the N weights:

$$C = \sum_{i=1}^{N} w_i \cdot c \quad c \in [0,1]\,. \tag{2.30}$$

Solutions out of the constraint are penalized with a fitness value of -1.

The experimental study consists of running a traditional binary-encoded GA (see (2.13)) with the generational approach equipped with QMO computed for different levels of quantum noise. The proposed GA was used to solve different instances of the 01-knapsack problem considering different dimensions of the region of the acceptable solutions space. In detail, for each configuration of quantum noise affecting the simulation of QMO, three values of N were considered, 10, 30, and 100, as well as three different values of C, obtained by setting c equal to 0.25, 0.5, and 0.75.

This analysis aims to investigate how quantum noise can affect the search of a GA equipped with QMO depending on whether the problem to be solved is more or less constrained.

To avoid the noisy simulation of 30-dimensional and 100-dimensional quantum circuits which would be quite hard even for the best supercomputer on the earth, the QMO algorithm has been distributed over sub-quantum circuits according to the D-NISQ architecture proposed in [5]. Indeed, the D-NISQ architecture offers a reference model for distributing quantum computation in smaller quantum circuits that can then be executed on different quantum processors. In detail, considering a mating pool composed of M individuals to be mated, each of them with length m. Then, individuals can be divided into N sub-strings, and QMO on each set of sub-strings can be run on different quantum devices of smaller sizes than the original m. Each QMO produces a sub-part of the new individual generated. Finally, the integration of the different outputs is in the end classically performed by means of the *information fusion* layer, which returns the generated offspring solution. Fig. 2.17 reports a schematic view of how QMO is distributed according to the D-NISQ reference model.

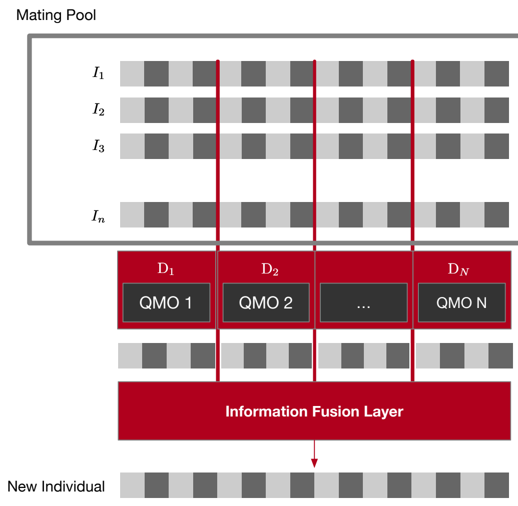


Figure 2.17: QMO distribution according to the D-NISQ reference model: mating pool individuals are divided into groups of genes; each group is then mated according to QMO algorithm on a different quantum processor $D_1, D_2, \ldots, D_N$; finally, the generated sub-groups of bits are merged by the *information fusion layer* and the new individual is therefore obtained.

In the first study, for each problem instance and noise model, the average value of the best solutions found by the GA over $R = 60$ independent runs of the algorithm was analyzed.

In the second step, the solutions found by GAs were grouped in clusters of quantum noise rather than being analyzed for each noise model. This is motivated by the fact that also the calibrations of quantum devices are affected by noise as all the physical operations of measurements, and therefore it may be convenient to study any patterns in the results grouped by quantum noise ranges rather than for specific noise models. The clusters considered are labeled as *High* (H), *Medium* (M), *Low* (L), and *Zero* (Z). These groups correspond to different intervals of bit flip and depolarizing probabilities: in detail, in the group labeled as *High* noise, it was inserted all the results obtained considering s_{bf} in $[0.2, 0.25]$ or s_d in $[0.01, 0.02]$; in the group labeled as *Medium* noise the levels of noise considered are for s_bf in the range $[0.1, 0.15]$ and for $s_d = 0.007$; for *Low* noise the s_{bf} and s_d have been considered in the ranges $[0, 0.05]$ and $[0, 0.001]$ respectively; finally, the group labeled *Zero* noise refers to the results obtained in the case of noiseless simulations. Fig. 2.18 graphically shows the clusters of quantum noise on the heatmap discussed above.

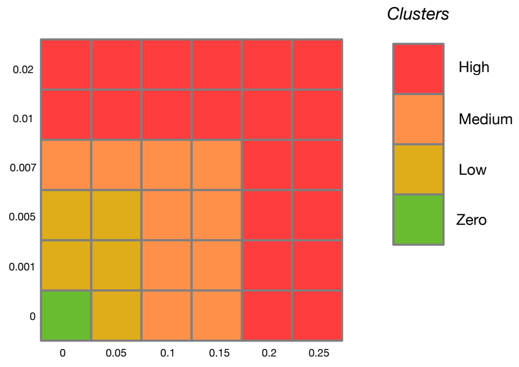


Figure 2.18: Graphical representation of the different clusters of quantum noise.

2.1.3.IV Results

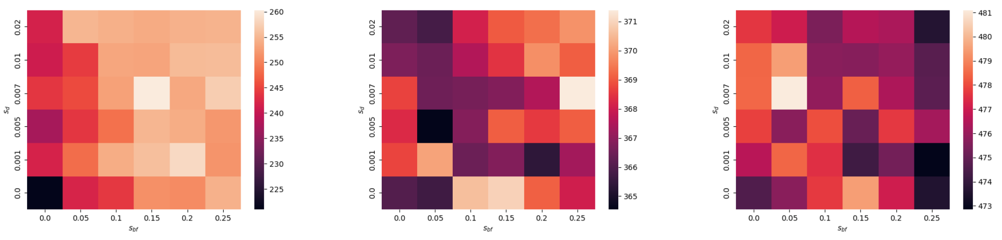
This section reports the results of our experimentation. For the sake of reproducibility, the hyperparameters of GAs used for the different knapsack problem sizes are reported in Table 2.14.

Moreover, for the 30-dimensional and the 100-dimensional Knapsack problem, the D-NISQ architecture for QMO distribution is built in such a way that each sub-circuit uses a maximum of 6 qubits. Firstly, the heatmaps in Fig. 2.19, Fig. 2.20, and Fig.

Table 2.14: Hyperparameters used for the experiments.

n	**pop_size**	p_c	p_m	**max_eval**	**max_gen**	**tourn_size**
10	20	0.8	0.3	524	40	3
30	20	0.8	0.3	5000	300	3
100	20	0.8	0.5	5000	300	3

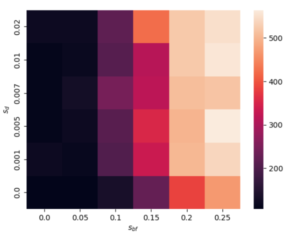
2.21 report the average best fitness values found at the end of the R independent and randomly initialized runs of the GAs for the knapsack problem with size 10, 30 and 100, considering the different combinations of noises affecting QMO computation. These plots show a recurrent behavior: for highly constrained problems ($c = 0.25$) the best solutions are found with a high level of noise affecting the computation of QMO. This is particularly evident in the 30-dimensional knapsack problem. On the other hand, when the constraint of the problem is lighter ($c = 0.75$) and therefore the space of acceptable solutions is bigger, the best solutions are found when low levels of quantum noise affect QMO. Finally, for the medium-constrained problem ($c = 0.50$), no clear effect of noise is highlighted by the proposed analysis. Furthermore, for the sake of completeness, the 100-dimensional case with $c = 0.25$ has been reported, but the very big region of unacceptable solutions doesn't permit a suitable evolution considering the same configuration of the GA used for the other cases.

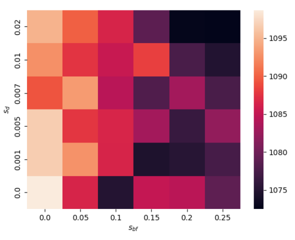


(a) $c = 0.25$ (b) $c = 0.50$ (c) $c = 0.75$

Figure 2.19: Average maximum fitness values against the different levels of quantum noise, for the 10-dimensional 01 Knapsack problem with different capacities.

Secondly, Fig. 2.22, Fig. 2.23, and Fig. 2.24 report the bar plots representing the maximum fitness values obtained in the GAs executed with the quantum noise affecting QMO belonging to the different clusters, for the knapsack size of 10, 30 and


(a) $c = 0.25$


(b) $c = 0.50$

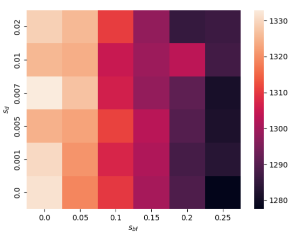
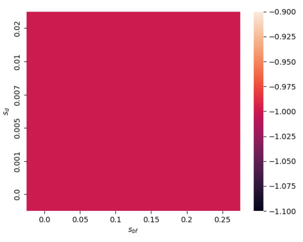
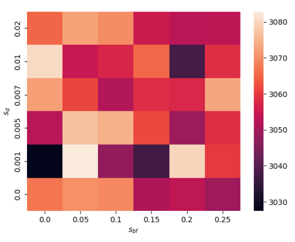

(c) $c = 0.75$

Figure 2.20: Average maximum fitness values against the different levels of quantum noise, for the 30-dimensional 01 Knapsack problem with different capacities.


(a) $c = 0.25$


(b) $c = 0.50$

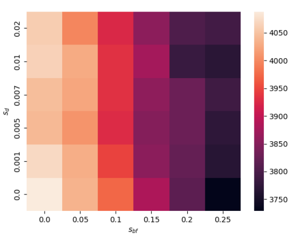
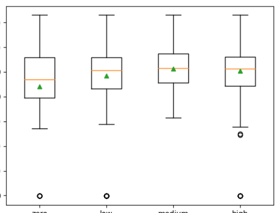

(c) $c = 0.75$

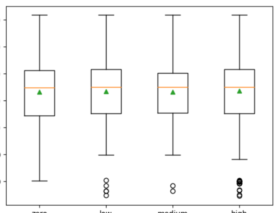
Figure 2.21: Average maximum fitness values against the different levels of quantum noise, for the 100-dimensional 01 Knapsack problem with different capacities.

100 respectively. In the box plots the median of the results is represented as a yellow line in the box, while the mean is drawn as a green triangle. The higher these two quantities, the better the effect of the related cluster of noise on QMO. These bar plots confirm the behavior highlighted in the first analysis: for the highly constrained 01 knapsack problem a medium-high level of noise boosts the performance of the GAs equipped with QMO, on the other hand, when the space of acceptable solutions is big, such as when the capacity of the knapsack is the 75% of the sum of the weights, then the best results are obtained when no noise or a low level of noise affects QMO. No particular dependence of the results from the quantum noise is highlighted in the cases with $c = 0.5$.

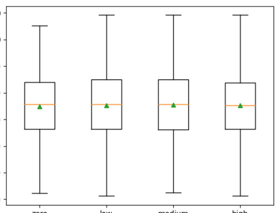
The obtained results are reported also numerically in Table 2.15. Here, for the different knapsack sizes and for the different clusters of noise affecting QMO, it is reported for all the different knapsack capacities, the mean of the best fitness μ obtained at the end of the GAs belonging to such a class, as well as the standard



(a) $c = 0.25$

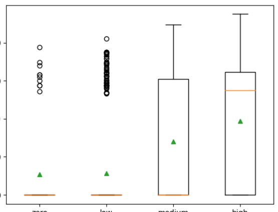


(b) $c = 0.50$

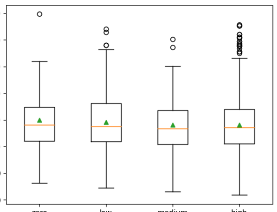


(c) $c = 0.75$

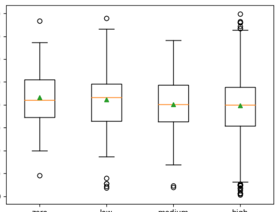
Figure 2.22: Box plots containing the set of best fitness values found solving the independent instances of the 10-dimensional 01-Knapsack for the different clusters of quantum noise affecting QMO. Each plot refers to a different knapsack capacity.



(a) $c = 0.25$

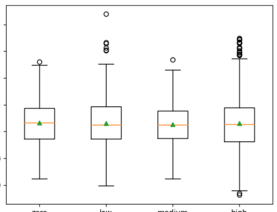


(b) $c = 0.50$

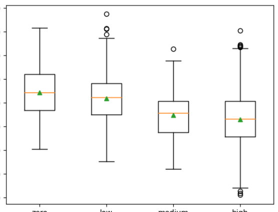


(c) $c = 0.75$

Figure 2.23: Box plots containing the set of best fitness values found solving the independent instances of the 30-dimensional 01-Knapsack for the different clusters of quantum noise affecting QMO. Each plot refers to a different knapsack capacity.



(a) $c = 0.50$



(b) $c = 0.75$

Figure 2.24: Box plots containing the set of best fitness values found solving the independent instances of the 100-dimensional 01-Knapsack for the different clusters of quantum noise affecting QMO. Each plot refers to a different knapsack capacity.

deviation of this set of results and the percentage gain of the μ with respect the noiseless cluster[5]. As highlighted in Table 2.15, important percentage gains of the mean of the best results are recorded when the capacity of the knapsack is very

[5]The $\%_{gain}$ for a certain cluster of noise with respect the Zero noise cluster is computed as $(\frac{\mu_i}{\mu_0} \cdot 100) - 100$, where μ_i is the mean value of the best fitness obtained using the $i-th$ cluster of noise and μ_0 is the mean of the best fitness found in the noiseless simulations.

Table 2.15: Experimental results of the GA equipped with QMO and a quantum noise belonging to the different clusters defined.

Knapsack Size	**Cluster**	**c = 0.25**			**c = 0.50**			**c = 0.75**		
		μ	σ	$\%_{gain}$	μ	σ	$\%_{gain}$	μ	σ	$\%_{gain}$
10	Z	220.9	95	-	366.0	59.9	-	474.6	69.5	-
	L	242.6	68.7	9.8	367.5	59.8	0.41	476.8	72.8	0.46
	M	**256.6**	45.9	**16.2**	366.7	58.1	0.19	**477.1**	73.4	**0.53**
	H	252.4	50.6	14.2	**368.1**	59.1	**0.57**	476.0	71.7	0.29
30	Z	106.5	241.5	-	**1098.7**	118.1	-	**1331.6**	131.1	-
	L	114.7	249.5	7.7	1092.0	109.1	-0.60	1323.4	133.5	-0.61
	M	281.5	318.9	199	1081.1	102.4	-1.6	1302.1	129.5	-2.2
	H	**388.8**	315.0	**265**	1081.8	102.6	-1.5	1298.0	129.2	-2.5
100	Z	-			**3066.9**	180.9	-	**4088.3**	216.6	-
	L				3061.9	198.2	-0.16	4037.8	210.9	-1.2
	M				3055.1	177.5	-0.38	3894.9	211.7	-4.7
	H				3060.1	190.4	-0.22	3861.6	230.1	-5.5

limited. On the other hand, important percentage losses are registered when the capacity of the knapsack is big. No significant gains or losses are registered for the case of $c = 0.5$.

2.1.3.V Summary

The results discussed above highlight a certain dependence between the performance of the GAs equipped with QMO and quantum noise affecting the computation of the related quantum circuit. This dependence seems to be strictly related to the size of the problem-acceptable solutions space ($\mathcal{A}$). For highly constrained 01 knapsack problems (small $\mathcal{A}$), the best results are obtained when a high level of quantum noise affects the computation of QMO, while on the other hand, when the constraint of the problem is small (high knapsack capacity and therefore large $\mathcal{A}$) the best solutions are found when a low level of noise affects QMO. From this, it appears that quantum noise increases the randomness of the matching process performed by QMO in the genetic algorithm workflow. Therefore, in the former case when the space of feasible solutions is highly constrained, this randomness increases the probability of generating solutions that respect the constraint. On the other hand, when the size of the feasible solution space is big (so the constraint is small), randomness brought about by high levels of quantum noise penalizes the ‘natural’ evolution of a GA, generating solutions that are far away from the parent ones not allowing an appropriate degree of exploitation of the algorithm. In the future, this analysis can be carried out on real quantum hardware and on different quantum operators from QMO as well as on different problems from the 01 knapsack problem. The dependency between acceptable solutions space of an optimization problem and quantum noise can be in this way further analyzed.

2.2 Quantum Fuzzy Reasoning

In the realm of classical logic, the principles of true and false stand as distinct, unwavering states. However, when exploring the intricacies of the quantum world, the comforting certainties of classical logic begin to waver amidst the haze of uncertainty. It is within this realm that the marriage of quantum mechanics and fuzzy logic gives rise to innovative algorithms potentially able to change the fuzzy reasoning we know today.

This section describes two approaches able to perform fuzzy reasoning on quantum annealers and on quantum digital computers respectively. The discussion reported in Sec. 2.2.2 collects the works published in [139, 140, 141], while the quantum algorithm proposed in Sec. 2.2.3, which is able to obtain an exponential speed-up in computing fuzzy rules on quantum digital computers with respect the classical counterpart, has been published with an important paper in [15], further developed in [10, 7] and, it has been successfully used for the first time in a real-world application in [6].

The two research lines hereafter reported differ not only for the different backends on which the algorithms are designed for, but also for the ideological approach used: in the first scenario, the one related to quantum annealers, the idea was to start from modeling basic fuzzy logic operations as optimization problems to be solved by means of quantum annealers and then merge together the various steps to achieve a fuzzy inference engine able to compute the mapping input-output in an FRBS. On the other hand, in the latter approach, it has been developed a wholistic Quantum Fuzzy Inference Engine (QFIE) able to speed up the fuzzy inference process, which is classically impractical in contexts characterized by a high number of variables to take into account. Indeed, FRBSs suffer from an implementation problem known as *fuzzy rule explosion*: the number of rules in an FRBS grows exponentially with the number of variables that make up the system. The proposed QFIE is able to tackle this problem by exploiting the massive parallelism in quantum computation and enabling simultaneous evaluation of all the rules of an FRBS, thanks to quantum mechanics peculiarities, such as quantum superposition and entanglement.

As proven by the literature analysis in Sec. 2.2.1, the proposed approaches are the very first quantum algorithms able to run fuzzy inference on quantum devices. Moreover as a secondary effect, the ability of fuzzy logic to represent computation using a linguistic approach, and the ability of fuzzy systems to act as universal approximators of functions [177][176], can lead to a strong simplification in the design of quantum circuits by making quantum computing accessible to researchers and practitioners who do not have basic knowledge in the mathematics of complex numbers and Hilbert vector spaces.

2.2.1 Related Work

Both quantum mechanics and fuzzy sets theory are related to the concept of uncertainty. This affinity has led the scientific community to develop approaches that mutually bind these two disciplines, as also demonstrated by recent research activities [193, 88, 108, 85, 91], where quantum approaches and fuzzy methods interact to each other's advantage. More generally, two lines of research have emerged in recent years: on the one hand, fuzzy sets theory has been used for simulating and modeling physical quantum systems [105, 21]; on the other hand, quantum computing approaches have been investigated to improve the performance of conventional fuzzy systems. Our research belongs to the second category, so the main research activities carried out in this area are discussed below.

An early example of the application of quantum computing to improve the computational performance of fuzzy systems is presented by Rigatos et al. in [149]. In this work, one-step quantum addition and subtraction replace operations between large matrices to speed up the inference process of fuzzy switching control. In more detail, this research shows that some operations performed in that specific fuzzy inference engine, namely *increase* and *decrease*, can be replaced by quantum operations. However, at the time when that research was carried out, no suitable quantum devices or simulators were available to allow experimental validation on the field. Successively, different research focused on the quantum implementation of fuzzy logic operators. In [174], Visintin et al. use quantum gates to implement t-norm and t-conorm operations so as to enable a quantum version of fuzzy union, intersection, and so on. Similarly, in [30], Avila et al. introduced an approach for the implementation and validation of fuzzy $X(N)or$ operations using quantum computing. In this research, the description of these operations is based on compositions of controlled and unitary quantum transformations, and the corresponding interpretation of fuzzy operations is obtained by applying operators of projective measurements. In the same vein as these works, Reiser et al. introduced a quantum circuit-based approach for the modeling and interpretation of Atanassovs intuitionistic fuzzy logic [148].

Although all of the above research efforts show how quantum circuits can be useful for performing basic operations between fuzzy sets, none of them implements an efficient quantum algorithm for executing fuzzy rules on actual quantum devices. Instead, this is what strongly characterizes the proposed research, which enhances the embryonic work presented in [9]. Indeed, this research introduced a quantum approach to implementing a fuzzy system based on a lookup table. In particular, Grover's algorithm has been used to perform the search for relationships between input and output variables of a system using a quantum computer. Although this paper represents the first attempt to use a well-known quantum algorithm to implement fuzzy systems, its use is very limited since the input-output relationships present in the lookup must be generated using classical computation. This limitation does not allow us to obtain a computational advantage from the quantum implementation.

Following the analysis of the state of the art, it can be said that no quantum approaches for efficient execution of fuzzy rules have yet been studied and tested on current quantum devices, as is done in this research work by introducing innovative

QFIEs able to be run both on quantum annealers and on quantum digital computers.

2.2.2 Fuzzy Inference on Quantum Annealers

The idea of representing fuzzy sets with quantum states comes from the similarity between the degrees of membership (a value between 0 and 1), and the probability of collapsing a superpositioned qubit into a binary state. Here we focus on the adiabatic model representation of fuzzy sets in order to exploit the large number of qubits in the quantum computers developed for the adiabatic model (e.g., D-Wave Systems).

2.2.2.I A Quantum Representation of Fuzzy Sets

Representing a fuzzy set with qubits, may not initially seem to be any similar to an optimization problem. As will be shown here, they can be formulated in QUBO so that a quantum computer can treat them as optimization problems. We will review how to represent a fuzzy set by qubit states in this subsection. In the next subsection, it will be explained how implementing fuzzy set operations (such as union and intersection) in the quantum model is formulated as a QUBO optimization problem.

Let A be a fuzzy set with n members (x_i) having membership grades $\mu_A(x_i)$. A can be represented by n qubits (q_i) in superposition state with $p_i(1) = \mu_A(x_i)$; where $p_i(1)$ is the probability of q_i being collapsed to the state 1.

The above definition is applicable to both circuitry and adiabatic models. In the adiabatic model, setting up a qubit system to represent a fuzzy set is equivalent to giving linear biases to the n qubits corresponding to the values of the membership function. There will be no quadratic term necessary for the representation. This is trivial since the members of a fuzzy set are independent. This practically means that a fuzzy set is simply represented by a system of biased (stimulated) qubits called *qfuzzy* system.

The corresponding BQM objective function of a qfuzzy system is thus defined as:

$$f(X) = \sum_{i=1}^{n} \big(\mu_A(x_i) \cdot x_i\big); \quad X = \{x_1, \ldots, x_n\} \tag{2.31}$$

Obviously, a qfuzzy system does not represent an optimization problem, thus it is not the subject of annealing. It is actually needed in its stimulated state, not in its collapsed state.

The next step towards implementing a quantum-fuzzy inference engine is to implement the basic fuzzy set operations in quantum algorithms. This will be explained in the next subsection.

2.2.2.II A Quantum Implementation of Basic Fuzzy Set Operations

Having defined how to represent fuzzy sets as qfuzzy systems, the aim of this section is to show how applying basic fuzzy set operations (Union, Intersection, Maximum, Alpha-cut, and Centroid defuzzification) on fuzzy sets can be translated to solving QUBO problems over the underlying qfuzzy systems, hence can be implemented on quantum computers.

- **Intersection Operator** (*Q.Intersection*): Let fuzzy sets A and B, with discrete membership functions $\mu_A(x_i)$ and $\mu_B(x_i)$ over the same universe of discourse $X = \{x_1, x_2, \ldots, x_n\}$ are represented as two qfuzzy systems as follows:

  $$\begin{aligned} A &: \textstyle\sum_{i=1}^{n} \big(\mu_A(x_i) \cdot x_i\big) \\ B &: \textstyle\sum_{i=1}^{n} \big(\mu_B(x_i) \cdot x_i\big) \end{aligned} \tag{2.32}$$

  where ":" denotes "represented by".

  We choose minimum-intersection and would like to find a new set C represented by a new qfuzzy system represented as:

$$C = A \cap B : \sum_{i=1}^{n} \big(\mu_C(x_i) \cdot x_i\big) \tag{2.33}$$

$$\mu_C(x_i) = \min\big(\mu_A(x_i), \mu_B(x_i)\big) \tag{2.34}$$

There is no singular operation existing in quantum computers that can compare or find the minimum of the bias values of any two qubits. However, we can take advantage of the fact that the annealing naturally leads to the minimum energy of a set of interconnected qubits, thus finding the minimum of an objective function. Therefore the question is which objective function over which qubits is to be minimised.

Let us create another quantum system Y with n qubits. The idea behind this system is that by the end of its annealing, the collapsed qubits act as a 0/1 switch between sets A and B, so that if qubit $y_i = 0$, the ith member of A is selected, and if $y_i = 1$ the corresponding member of B is selected.

Let us define a BQM objective function for this system as:

$$f(Y) = \sum_{i=1}^{n} \big((\mu_B(x_i) - \mu_A(x_i)) \cdot y_i\big) \tag{2.35}$$

It can be seen that if $\mu_A(x_i) < \mu_B(x_i)$, the ith term of 2.35 takes its minimum when $y_i = 1$. Similarly if $\mu_B(x_i) < \mu_A(x_i)$, this term takes its minimum when $y = 0$. Collectively, $f(Y)$ takes its minimum when $y_i = 0$ for the terms with $\mu_A(x_i) < \mu_B(x_i)$ and $y_i = 1$ for the terms with $\mu_A(x_i) > \mu_B(x_i)$. For when $\mu_A(x_i) = \mu_B(x_i)$, the quantum annealing may randomly collapse y_i to either 0 or 1 randomly, which does not change the result. Finally, when each y_i is observed after the annealing, it indicates which set has the minimum membership function at the ith point.

Once the values y_i result from the annealing of system Y, the qfuzzy system C can be made and represented as:

$$C = A \cap B : \sum_{i=1}^{n} \big(\mu_C(x_i) \cdot x_i\big) \tag{2.36}$$

$$\mu_C(x_i) = (1 - y_i) \cdot \mu_A(x_i) + y_i \cdot \mu_B(x_i) \tag{2.37}$$

Let fuzzy sets A and B, with discrete membership functions $\mu_A(x_i)$ and $\mu_B(x_i)$ over the same universe of discourse $X = \{x_1, x_2, \ldots, x_n\}$ are represented as two qfuzzy systems as follows:

$$\begin{aligned} A &: \textstyle\sum_{i=1}^{n} \big(\mu_A(x_i) \cdot x_i\big) \\ B &: \textstyle\sum_{i=1}^{n} \big(\mu_B(x_i) \cdot x_i\big) \end{aligned} \tag{2.38}$$

where ":" denotes "represented by".

We choose minimum-intersection and would like to find a new set C represented by a new qfuzzy system represented as:

$$C = A \cap B : \sum_{i=1}^{n} \big(\mu_C(x_i) \cdot x_i\big) \tag{2.39}$$

$$\mu_C(x_i) = \min \big(\mu_A(x_i), \mu_B(x_i)\big) \tag{2.40}$$

There is no singular operation existing in quantum computers that can compare or find the minimum of the bias values of any two qubits. However, we can take advantage of the fact that the annealing naturally leads to the minimum energy of a set of interconnected qubits, thus finding the minimum of an objective function. Therefore the question is which objective function over which qubits is to be minimised.

Let us create another quantum system Y with n qubits. The idea behind this system is that by the end of its annealing, the collapsed qubits act as a 0/1 switch between sets A and B, so that if qubit $y_i = 0$, the ith member of A is selected, and if $y_i = 1$ the corresponding member of B is selected.

Let us define a BQM objective function for this system as:

$$f(Y) = \sum_{i=1}^{n} \big((\mu_B(x_i) - \mu_A(x_i)) \cdot y_i\big) \tag{2.41}$$

It can be seen that if $\mu_A(x_i) < \mu_B(x_i)$, the ith term of (2.41) takes its minimum when $y_i = 1$. Similarly if $\mu_B(x_i) < \mu_A(x_i)$, this term takes its minimum when $y = 0$. Collectively, $f(Y)$ takes its minimum when $y_i = 0$ for the terms with $\mu_A(x_i) < \mu_B(x_i)$ and $y_i = 1$ for the terms with $\mu_A(x_i) > \mu_B(x_i)$. For when $\mu_A(x_i) = \mu_B(x_i)$, the quantum annealing may randomly collapse y_i to either 0 or 1 randomly, which does not change the result. Finally, when each y_i is observed after the annealing, it indicates which set has the minimum membership function at the ith point.

Once the values y_i have resulted from the annealing of system Y, the qfuzzy system C can be made and represented as:

$$C = A \cap B : \sum_{i=1}^{n} \big(\mu_C(x_i) \cdot x_i\big) \tag{2.42}$$

$$\mu_C(x_i) = (1 - y_i) \cdot \mu_A(x_i) + y_i \cdot \mu_B(x_i) \tag{2.43}$$

- **Union Operation** ($Q.Union$): Similarly, the union operation (based on maximum) of qfuzzy systems A and B is defined by introducing an intermediate quantum system Y consisting of n qubits and a BQM objective function defined as:

$$f(Y) = \sum_{i=1}^{n} \big((\mu_A(x_i) - \mu_B(x_i)) \cdot y_i \big) \tag{2.44}$$

After annealing of Y, it can be similarly shown that:

$$C = A \cup B : \sum_{i=1}^{n} \big(\mu_C(x_i) \cdot x_i \big) \tag{2.45}$$

$$\mu_C(x_i) = (1 - y_i) \cdot \mu_A(x_i) + y_i \cdot \mu_B(x_i) \tag{2.46}$$

- **Alpha-cut Operation:** The alpha-cut operator takes a fuzzy set and produces a crisp set of values along the x-axis for which their membership grade is equal to or greater than a given alpha. Let fuzzy set A be represented as a qfuzzy system as:

$$A : \sum_{i=1}^{n} \big(\mu_A(x_i) \cdot x_i \big) \tag{2.47}$$

We would like to extract a crisp set Z, in which:

$$Z = \{ x \in A \,|\, \mu_A(x) \geq \alpha \} \tag{2.48}$$

Again, in the lack of any comparison operator, the alpha-cut operator has to be reformulated in BQM. Similar to the max operator, we would need an intermediate qubit system Y in order to binarily flag out the x-values which are the members of the alpha-cut from those who are not. An initial suggestion is that the required BQM objective function should penalize (i.e., produce positive terms for all $\mu_A(x_i) < \alpha$). This initial objective function can then be formulated as:

$$f(Y) = \sum \big(\alpha - \mu_A(x_i)\big) \cdot y_i \tag{2.49}$$

This function takes its minimum by setting $y_i = 0$ for those x_i with greater membership grades than α and setting $y_i = 1$ otherwise. However, the problem is that if there is a x_i point in which $\mu_A(x_i) = \alpha$, the corresponding linear bias in the objective function is zero, thus the result is independent of the corresponding y_i value. This means that the objective function takes its minimum equally for both $y_i = 0$ and $y_i = 1$, in which case the algorithm is unable to identify if x_i is in or is out of the α-cut. To resolve this issue we need to adjust the objective function, so that it produces no zero biases, i.e. it must produce either positive biases for $\mu_A(x_i) \geq \alpha$ or negative biases otherwise. For this, a parameter ϵ is needed that is equal to half of the membership function resolution, so that the objective function can be adjusted as:

$$f(Y) = \sum \big(\alpha - \epsilon - \mu_A(x_i)\big) \cdot y_i \tag{2.50}$$

In this case, setting $y_i = 1$ for all the α-cut members would give $f(Y)$ its most negative possible value. Therefore the collapsed values of y_i's after the annealing process are actually a membership flag for any corresponding x_i to the alpha-cut.

Finally, the alpha-cut Z can be redefined as:

$$Z = \{x_i \in A \,|\, y_i = 1; y_i \in Y\} \tag{2.51}$$

- **Maximum Operator** ($Q.Max$): The maximum operator determines the member(s) of a fuzzy set with maximum membership grade(s). This is particularly useful for defuzzification in FOM/MOM/LOM (First/Mean/Last of Maxima) methods. Let fuzzy set A be represented as a qfuzzy system:

$$A : \sum_{i=1}^{n} \big(\mu_A(x_i) \cdot x_i\big) \tag{2.52}$$

We would like to extract a crisp set M, in which:

$$M = \{x \in A \,|\, \mu_A(x) = \max_i \big(\mu_A(x_i)\big)\} \tag{2.53}$$

Although this looks similar to the case of the maximum operator, the key difference is that instead of switching between corresponding members of two sets, here the aim is switching between the members of the same set based on whether the member has the maximum membership function or not. For example, if the set takes its single maximum at point m, there will be a single 1 in the mth qubit of Y when collapsed to binary states. This key difference, as will be seen, leads to a substantially different algorithm, unlike the previously examined operations, adding some quadratic terms to the objective function.

Since there is no maximum operator existing for qubit biases, here we need another quantum system (Y) with n qubits, in which the binary state of the qubit y_i can indicate whether x_i belongs to set M or not. In other words, Y acts as a 0/1 flag so that x_i has taken the maximum membership grade in A only if $y_i = 1$.

The BQM objective function of system Y must be formulated in a way that it increases more if smaller values of $\mu_A(x_i)$ are flagged, compared to the larger values of $\mu_A(x_i)$. Since flagging is equivalent to setting $y_i = 1$, this means that the objective function must be increased (i.e. penalized) by $1 - \mu_A(x_i)$ if its corresponding y_i is 1. Thus we expect the objective function to be in the form of:

$$f(Y) = \sum \big(1 - \mu_A(x_i)\big) \cdot y_i + \ldots \tag{2.54}$$

A problem with this partial objective function is that it always takes its minimum when all $y_i = 0$. Note that $1 - \mu_A(x_i) \geq 0$ therefore the minimum of $f(Y)$ is zero by setting all y_i to zero. We must then need to penalize the case of all $y_i = 0$. On the other hand, having more than a single zero should be penalized too since this will always increase the value of $f(Y)$.

Without losing the generality, let us assume that $\mu_A(x_i)$ has a single maximum. In this case, we expect exactly a single 1 in the result, i.e., the sum of all y_i's should not be more nor less than 1. Therefore we define the penalty term as $(\Sigma y_i - 1)^2$. The objective function then becomes:

$$\begin{aligned}
f(Y) &= \Sigma\big(1 - \mu_A(x_i)\big)y_i + (\Sigma y_i - 1)^2 \\
&= (1 - \mu_A(x_1)) \cdot y_1 + (1 - \mu_A(x_2)) \cdot y_2 + \\
&+ \ldots + (1 - \mu_A(x_n)) \cdot y_n + \\
&+ y_1^2 + y_2^2 + \ldots + y_n^2 + 1 + \\
&+ 2y_1y_2 + 2y_1y_3 + \ldots + 2y_{n-1}y_n - \\
&- 2y_1 - 2y_2 - \ldots - 2y_n
\end{aligned}$$

Note that for binary values, $y_i^2 = y_i$, and that the constant value 1 has no effect on minimizing the objective function. Then the equivalent objective function (named the same for simplicity) is defined as:

$$\begin{aligned}
f(Y) &= -\mu_A(x_1)y_1 - \mu_A(x_2)y_2 + \ldots \\
&= -\mu_A(x_n)y_n + 2y_1y_2 + 2y_1y_3 + \ldots + 2y_{n-1}y_n
\end{aligned}$$

Therefore,

$$f(Y) = -\sum_{i=1}^{n} \mu_A(x_i) y_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 y_i y_j \tag{2.55}$$

The above is in BQM form with linear biases $-\mu_A(x_i)$ and all quadratic coefficients equal to 2. Practically, the quadratic terms of this system of qubits are independent of the membership grade, so that they can be pre-programmed.

If there is more than a single maximum point, $f(Y)$ will equally take its minimum by setting $y_i = 1$ for either of them (not for a combination of them). This means that running the quantum annealing algorithm by this point may not give a single answer to the fuzzy set operation (i.e., the maximum operation). Due to the embedded randomness of the quantum annealing, each run of the algorithm may give one of the possible answers.

To find a single answer in the form of binary bits, we first notice that any given answer can maximize the membership function. Let us assume that the mth bit of Y is found to be 1, and we call $\mu_A(x_m)$ as μ_M. Then we notice that by knowing μ_M, the required maximum operator is a special case of applying α-cut operator when $\alpha = \mu_M$. The implementation of this part is already explained in the previous subsection.

Therefore, applying the maximum operator can be split into two steps:

- A quantum annealing based on (2.55) to find a possible binary sequence leading to μ_M.
- Another quantum annealing based on (2.50) with $\alpha = \mu_M$ to find the final binary sequence.

Now that the binary sequence y_i is determined, the target set M can be simply defined as:

$$M = \{x_i \in A \,|\, y_i = 1\} \tag{2.56}$$

The resulting y_i bits can also be readily used for FOM/MOM/LOM defuzzification: The x_i values corresponding to the left-most, middle, and right-most 1's in the binary sequence are equivalent to the FOM, MOM, and LOM of X, respectively.

Note that by changing the linear and quadratic coefficients in (2.55) respectively to $1 - \mu_A(x_i)$ and 2 it is possible to compute the smallest membership grade of A (*Q.Min* operator).

- **Centroid Defuzzification** (*Q.Defuzzifier*): Centroid defuzzification for a discrete fuzzy set A with membership function $\mu_A(x)$ is defined as:

$$C = \frac{\sum_{i=1}^{n} x_i \mu_A(x_i)}{\sum_{i=1}^{n} \mu_A(x_i)} \tag{2.57}$$

where n is the number of samples along the x axis. To formulate this as a BQM problem let us consider a given discrete membership function $\mu_A(x_i)$, and let's assume that the centroid is located at the kth position between 0 and n. By definition, the centroid divides the set based on its center of gravity. This means that the sum of the membership values on the left side of k must be equal (or the closest, due to the quantization error) to the sum of the values on its right side. This is therefore equivalent to minimizing an objective function defined as:

$$f_c(k) = \left(\sum_{i=1}^{k} \mu_A(x_i) - \sum_{i=k+1}^{n} \mu_A(x_i) \right)^2 \tag{2.58}$$

This time, to convert the above problem to BQM it is used the Ising implementation on quantum annealers. Indeed, the Ising implementation has a simpler match to mapping the centroid problem to the BQM problem, since one can simply associate $y_i = 1$ to the centroid's left side and $y_i = -1$ to its right side (or vice-versa) and can target minimizing $f_1(Y)$ defined as:

$$f_1(X,Y) = \big(\sum_{i=1}^{n} y_i \mu_A(x_i)\big)^2; Y = \{1,...,1,-1,...,-1\} \tag{2.59}$$

For a given fixed set of A, let's rename the values $\mu_A(x_i)$ as simply μ_i. In this case, the objective function for a given set is a function of binary values y_i, defined as:

$$f_1(Y) = (\sum_{i=1}^{n} y_i \mu_i)^2 \quad ; \quad Y = \{1,...,1,-1,...,-1\} \tag{2.60}$$

The constraint of having a single switch-over point between 1 and -1 in Y is an extra limitation that makes f not readily mappable to BQM since the function can take smaller values if Y has more than a single switch-over point. We suggest adding to f_1 another function f_2 (as a penalty function) that adds positive values to it unless the sequence of y_i's has a single switch-over point. To realize this function, we notice that the sum of the square of differences between each two consecutive y_i's in Y becomes 4 if and only if there is a single-switch-over point in Y. Therefore we define:

$$f_2(Y) = (y_1 - y_2)^2 + (y_2 - y_3)^2 + ... + (y_{n-1} - y_n)^2 - 4 \tag{2.61}$$

For more than one switch-over point, $f_2(Y)$ takes positive values. In an extreme case, any added penalty value that is more than zero should be big enough to rule out any small value of $f_1(Y)$ from being detected as a minimum by the annealer. We notice that:

$$\begin{aligned} &max\big(f_1(Y)\big) = (n-2)^2 \\ &f_2(Y) \in \{0,4,8,...,(n-1)^2 - 4\} \end{aligned} \tag{2.62}$$

To make sure the penalty value is big enough, we consider a factor k so that the value of $f_2(Y)$ in the case of one switch-over point is equal to or greater than the maximum of $f_1(Y)$:

$$\begin{aligned} &kf_2(Y) \geq f_1(Y); \quad if \quad f_2(Y) \geq 4 \\ &4k \geq (n-2)^2 \quad or \quad k \geq \frac{n}{4}(n-4)+1 \end{aligned} \tag{2.63}$$

Therefore, the new objective function is defined as:

$$f(Y) = f_1(Y) + kf_2(Y); \quad k \geq \frac{n}{4}(n-4)+1 \tag{2.64}$$

There is still an outstanding issue, which is for when there is no switch-over point in Y, i.e., when $Y = \{1, 1, ..., 1\}$ or $Y = \{-1, -1, ..., -1\}$. For these two cases, $f_2(Y) = -4$ which gives $f(Y)$ a lower value than what it would be for any single or multiple switch-over forms of Y, thus misleading the annealer. To avoid these two cases, we notice that the outcome of a quantum annealer is a list of Y arrangements sorted by their energy levels. If one simply ignores the first two low energy levels and takes the third lowest one, the two problematic cases will be avoided. This policy has no effect on the objective function formulation but is something to be done programmatically after the annealing process.

Next, let's convert $f_1(Y)$, $f_2(Y)$ and $f(Y)$ to BQM forms, so that the coefficient values q_i and q_{ij} are determined.

$$\begin{aligned} f_1(Y) =& \left(\sum_{i=1}^{n} y_i\mu_i\right)^2 \\ &= y_1^2\mu_1^2 + ... + y_n^2\mu_n^2 \\ &+ 2y_1y_2\mu_1\mu_2 + ... + 2y_{n-1}y_n\mu_{n-1}\mu_n \\ &= \sum_{i=1}^{n}\mu_i^2 + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mu_i\mu_j y_i y_j \end{aligned} \tag{2.65}$$

In the above, we notice that $y_i^2 = 1$ for all i.

$$\begin{aligned}
kf_2(Y) =& k\big((y_1 - y_2)^2 + ... + (y_{n-1} - y_n)^2 - 4\big) \\
&= k(y_1^2 + 2y_2^2 + ... + 2y_{n-1}^2 + y_n^2) \\
&- k(2y_1y_2 + 2y_2y_3 - ... + 2y_{n-1}y_n) - 4k \\
&= k(2n-2) - 2k\sum_{i=1}^{n-1} y_iy_{i+1} - 4k \\
&= 2kn - 6k - 2k\sum_{i=1}^{n-1} y_iy_{i+1}
\end{aligned} \tag{2.66}$$

$$\begin{aligned}
f(Y) =& f_1(Y) + kf_2(Y) \\
&= \sum_{i=1}^{n} \mu_i^2 + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mu_i\mu_jy_iy_j \\
&+ 2kn - 6k - 2k\sum_{i=1}^{n-1} y_iy_{i+1} \\
&= \sum_{i=1}^{n} \mu_i^2 + 2kn - 6k \qquad \text{(constant term)} \\
&+ 2\big(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mu_i\mu_jy_iy_j - k\sum_{i=1}^{n-1} y_iy_{i+1}\big)
\end{aligned} \tag{2.67}$$

For minimizing $f(Y)$ in (2.67), we ignore the constant terms and factors, thus the aim is to minimize the following term:

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mu_i\mu_jy_iy_j - k\sum_{i=1}^{n-1} y_iy_{i+1} \tag{2.68}$$

To determine the BQM coefficient, (1.16) and (2.68) can be compared, therefore the linear and quadratic coefficients of (1.16) can be determined to be:

$$\begin{aligned}
q_i &= 0 \\
q_{ij} &= \mu_i\mu_j - \begin{cases} k & \text{if } j = i+1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{2.69}$$

k can be any value equal or greater than $\frac{n}{4}(n-4)+1$.

The linear and quadratic coefficients determined in (2.69) can be calculated for any given fuzzy set, and be used to set up a quantum annealer. Once the annealer reaches its collapsed state, the collapsed qubit values in the third lowest energy level can be picked up. The location of the single switch-over between 1 and -1 in (or vice-versa) in the collapsed bit array is equivalent to the location of the centroid.

- **Singleton Operator** (*Q.Singleton*) and **Replication Operator** (*Q.Replicate*): In order to define a pipeline implementing a Mamdani FRBS it is required to define two other operators: the *Q.Singleton* operator takes a crisp value $\hat{x}$ and makes a singleton fuzzy set $A = \{1/\hat{x}\}$ (i.e., having a single member $\hat{x}$ with $\mu_A(\hat{x}) = 1$); the *Q. Replicate* operator takes a crisp value $\hat{x}$ and makes a fuzzy set A, in which $\mu_A(x_i) = \hat{x}$ for all i .

2.2.2.III Implementing the Mamdani Fuzzy Inference on Quantum Annealers

In this subsection, it will be shown how each of the four building blocks of a Mamdani inference system (Fig. 1.4) can be made by pipelining the implemented operational units explained earlier. The challenge is to deliver the function of each block exclusively by using the operational units stated previously.

Let us assume a Mamdani fuzzy inference system is to be implemented with n crisp inputs ($x_1 ... x_n$), single crisp output (y), m rules, minimum intersection, maximum union, and centroid defuzzification operators.

- **Fuzzifier**: The fuzzifier block takes crisp values for each input variable, then given the antecedent fuzzy sets, it calculates the membership grade of each fuzzy set at the point of the corresponding input values. In other words, given $\hat{x}$ and a fuzzy set A, it must produce a non-normal singleton fuzzy set F with a single spike at $\hat{x}$ where $\mu_F(\hat{x}) = \mu_A(\hat{x})$. Different fuzzifier blocks are to be employed for each input and each relevant antecedent. Let us focus on a single block that converts x_1 to $F_1 = \{\Sigma\ \mu_{A_1}(x_1)/x_1\}$. The implementation of this single block is shown in Fig. 2.25. First, a Q.Singleton unit converts x_1 to a singleton set $\{1/x_1\}$, then a Q.Intersection unit, makes the intersection of this set with the antecedent set A_1. Since the singleton set has a single non-zero grade at x_1, the result is a singleton at x_1 with the grade $\mu_{A_1}(x_1)$.

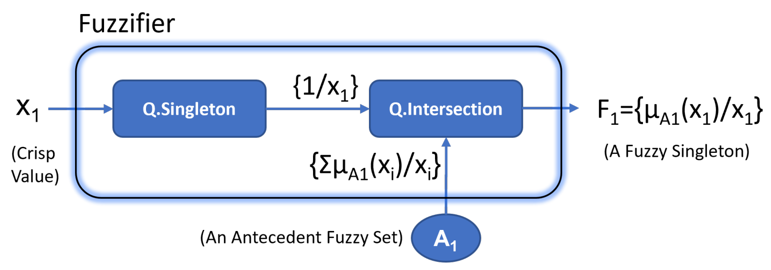


Figure 2.25: The implementation of a single fuzzifier block

- **Rule Strength Calculator**: The job of this block is to calculate the firing strength of each rule. Each rule has n antecedents, thus the inputs of this block are n fuzzified values. In Mamdani's inference method, the firing strength of the rule can be computed as the minimum of the fuzzified values. According to our design, the inputs to the block are n fuzzy singletons coming from the fuzzifier blocks. As shown in Fig. 2.26, calculating the firing strength of a rule (e.g. the first rule) includes two steps: first, a union set of all the incoming fuzzified values is produced (i.e., $\{\Sigma\ \mu_{A_i}(x_i)/x_i\}$) by Q.Union unit(s). This will be a set with all the fuzzified values aggregated in a single set, in which each non-zero grade is a fuzzified value located at its corresponding crisp input. Secondly, a Q.Min unit takes the produced union and calculates the maximum grade, which is the rule's firing strength by definition (i.e., w_1 for rule 1).

- **Rule Output Calculator**: Once the firing strength of all the rules is calculated, the consequent set of each rule is to be capped with the rule's strength

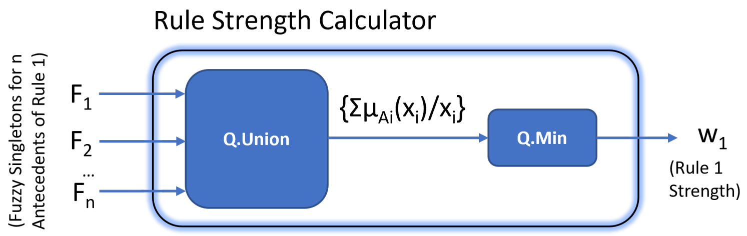


Figure 2.26: The implementation of a rule firing strength calculator

to produce the rule's output fuzzy sets ($O_1...O_m$). This is equivalent to intersecting the consequent set with an intermediate "flat" fuzzy set. The strength of the rule is the membership grade of all members of such a set. This can be implemented by a Q.Replicate unit followed by a Q.Intersection unit, as shown in Fig. 2.27. For example in rule 1, a consequent set $C_1 = \{\Sigma\ \mu_{C_1}(y_j)/y_j\}$ is defined over its universe of discourse Y (which can be different from X). A Q.Replicate unit in this block takes the calculated rule firing strength w_1 and creates an intermediate fuzzy set $\{\Sigma\ w_1/y_j\}$ with all the grades equal to w_1. Then, this set is intersected with C_1 in order to create a capped version of C_1. Formally, the output fuzzy set of this block can be expressed as $O_1 = \{\Sigma\ min[w_1, \mu_{C_1}(y_j)]/y_j\}$.

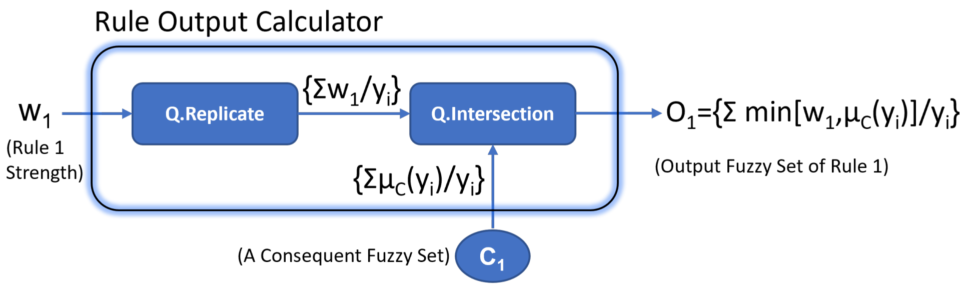


Figure 2.27: The implementation of a rule output calculator block

- **Aggregator and Defuzzifier**: After each rule's output set is created, the union of all the output sets is to be created and finally defuzzified in order to calculate the final crisp output of the inference. This can be implemented by serializing Q.Union unit(s) and a Q.Defuzzifier unit. As shown in Fig.

2.28, different output sets ($O_1...O_m$) coming from m rules are taken to the block, then their union set is produced as $O = \{\Sigma\ \mu_O(y_j)/y_j\}$ where $\mu_O(y_j) = max[\mu_{O_1}(y_j), ..., \mu_{O_m}(y_j)]$. Finally, the fuzzy output set O is to be defuzzified by a Q.Defuzzifier unit in order to calculate the final crisp output y. It is noticeable that the Q.Defuzzifier unit is an implementation of the centroid defuzzification. If more simplicity is needed, this unit can be replaced by a Q.Max that implements a MAX defuzzifier.

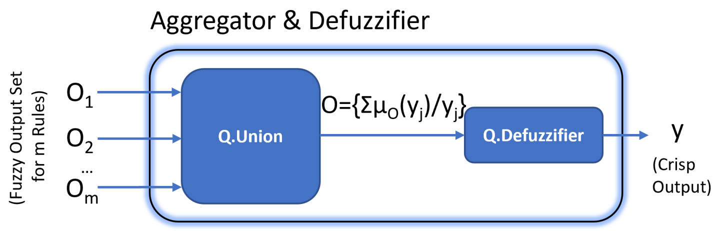


Figure 2.28: Implementing the block for aggregating the individual output sets of each rule, and defuzzifying it

Once the building blocks of the inference are designed based on the quantum annealer operational units, the blocks are to be replicated and pipelined according to the number of inputs and rules, in order to create the whole inference. The proposed design of the inference system is shown in Fig. 2.29. As illustrated, for a system with n inputs ($A_1...A_n$), up to n antecedent sets can exist ($A_1...A_n$). Each of the m rules (e.g., rule i) can have up to n stacked fuzzifier blocks. In each rule, the multiple outputs of the fuzzifiers corresponding to the different antecedents are given to a single rule strength calculator block, in which the rule's firing strength (w_i) is calculated. Each rule also has a single consequent set C_i (which can be repeated in other rules). The calculated w_i along with the consequent set of the rule are given to a single rule output calculator block in order to produce an output set of the rule. Repeating this process for m rules produces m output sets which are to be given to a single aggregator/defuzzifier block in order to produce the final crisp output.

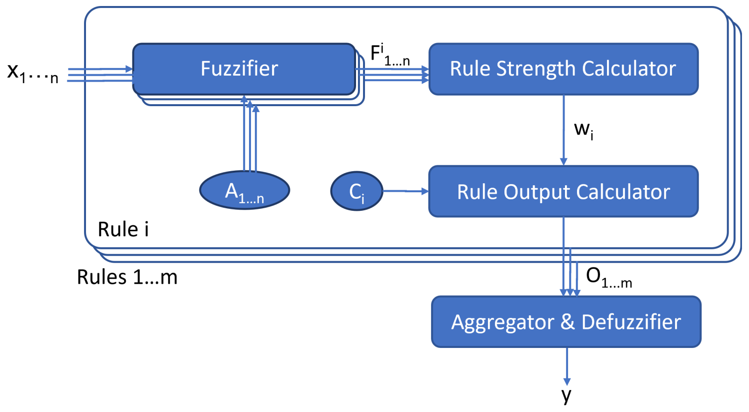


Figure 2.29: Mamdani inference system made of the pipelined building blocks

2.2.2.IV A Sample Numerical Example and Its Implementation on a Real Quantum Computer

In this subsection, the suitability of the proposed approach is first evaluated by a sample numerical example and secondly with a test on a real quantum annealer from the D-Wave systems.

Let us consider a simple fuzzy control system with two inputs, two antecedent fuzzy sets, one consequent fuzzy set, and two rules. A quantum annealer with 10 qubits is assumed to be available, therefore the x-values of the discretized fuzzy sets are integers 1 to 10. Let the rules be:

- Rule 1: IF x_1 IS A_1 and x_2 IS A_2 THEN y IS C_1
- Rule 2: IF x_1 IS A_2 and x_2 IS A_1 THEN y IS C_2

and let the fuzzy sets be:

- $A_1 = \{0.0/1, 0.5/2, 1.0/3, 1.0/4, 1.0/5, 0.5/6, 0.0/7, 0.0/8, 0.0/9, 0.0/10\}$
- $A_2 = \{0.0/1, 0.0/2, 0.0/3, 0.0/4, 0.5/5, 1.0/6, 1.0/7, 1.0/8, 0.5/9, 0.0/10\}$
- $C_1 = \{1.0/1, 1.0/2, 1.0/3, 0.8/4, 0.6/5, 0.4/6, 0.2/7, 0.0/8, 0.0/9, 0.0/10\}$
- $C_2 = \{0.0/1, 0.0/2, 0.0/3, 0.2/4, 0.4/5, 0.6/6, 0.8/7, 1.0/8, 1.0/9, 1.0/10\}$

Finally, let us set the inputs as $x_1 = 2$ and $x_2 = 6$.

Each of the two rules has two fuzzifier blocks. In the first block of the first rule, with reference to Fig. 2.25 and 2.29, the steps are conducted as follows:

- The Q.Singleton unit produces a singleton set $\{1.0/2\}$
- The Q.Intersection takes the above set and A_1 to produces a fuzzy set $F_1^1 = \{0.5/2\}$

Similarly, the outputs of the other three fuzzifier blocks are $F_1^2 = \{1.0/6\}$, $F_2^1 = \{0.0/2\}$ and $F_2^2 = \{0.5/6\}$.

The rule strength calculator block (Fig. 2.26) acts as follows:

- In rule 1: The Q.Union unit makes $F_1^1 \cup F_1^2 = \{0.5/2, 1.0/6\}$ then the Q.Min units calculates the rule's firing strength as $w_1 = 0.5$.
- In rule 2: The Q.Union unit makes $F_2^1 \cup F_2^2 = \{0.0/2, 0.5/6\}$ then the Q.Min unit calculates the rule's firing strength as $w_2 = 0.0$.

The rule output calculator block (Fig. 2.27) acts as follows:

- In rule 1: The Q.Replicate unit takes $w_1 = 0.5$ and creates a fuzzy set $\{0.5/1...10\}$, then the Q.Intersect unit produces the intersection of this set and C_1, which is $O_1 = \{0.5/1, 0.5/2, 0.5/3, 0.5/4, 0.5/5, 0.4/6, 0.2/7,$
$0.0/8, 0.0/9, 0.0/10\}$
- In rule 2: taking $w_2 = 0.0$, the Q.Replicate creates $\{0.0/1...10\}$, then the Q.Intersect unit produces the intersection of this set and C_2 which is $O_2 = \{0.0/1...10\}$ (i.e., rule 2 is not fired).

Finally, in the aggregator/defuzzifier block (Fig. 2.28) the Q.Union acts on O_1 and O_2 to create $O = \{0.5/1, 0.5/2, 0.5/3, 0.5/4, 0.5/5, 0.4/6, 0.2/7, 0.0/8, 0.0/9, 0.0/10\}$, and the Q.Defuzzifier calculates the centroid of O as the final inference output y.

Calculating the centroid of O yields 3.7. Given the system's resolution, the rounded centroid is $y = 4$.

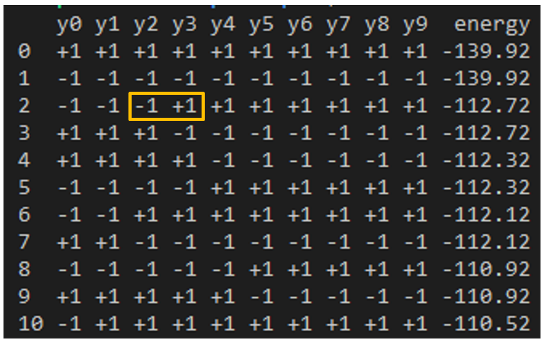
Let us now implement this numerical example on a real adiabatic quantum computer with 10 qubits. The implementation is based on D-Wave System a cloud-based real quantum computing platform. D-Wave also provides some Python libraries for programming using web-based and desktop IDE that connect to the same platform[6].

Each operational unit explained above is to be implemented individually, then pipelined as shown in Fig. 2.29. Sample implementations of these units on a real quantum computer are already shown in [139, 140]. So as not to exceed the size of this dissertation, it will be reported a real implementation of just a single building block of the inference process, namely the final Aggregator/Defuzzifier block. From the example, the aggregated set is $O = \{0.5/1, 0.5/2, 0.5/3, 0.5/4, 0.5/5, 0.4/6, 0.2/7, 0.0/8, 0.0/9, 0.0/10\}$ and hereafter it will be shown an implementation a Q.Defuzzifier unit to calculate its centroid y. The Q.Defuzzifier output is shown in Fig. 2.30, in which each row contains the states of the collapsed qubits. The rows are descendingly ranked by the system's energy levels. As explained, the defuzzified value is indicated by the first switchover point of the qubit spins in a single state (row). As highlighted in Fig. 2.30, the switchover is in the third row between the 3rd and the 4th qubits, i.e., the defuzzified value is a number between 3 and 4. We notice that the system's resolution is 1 and this result matches with y=3.7 which was theoretically calculated above.

2.2.2.V Summary

Considering that todays applications of fuzzy systems increasingly involve large amounts of data or large sets of rules, there is a strong emergence of identifying innovative computational paradigms capable of efficiently managing these types of

[6]more details is out of the scope of this paper, and can be found in https://docs.ocean.dwavesys.com



```
    y0 y1 y2 y3 y4 y5 y6 y7 y8 y9  energy
0   +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 -139.92
1   -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -139.92
2   -1 -1 -1 +1 +1 +1 +1 +1 +1 +1 -112.72
3   +1 +1 +1 -1 -1 -1 -1 -1 -1 -1 -112.72
4   +1 +1 +1 +1 -1 -1 -1 -1 -1 -1 -112.32
5   -1 -1 -1 -1 +1 +1 +1 +1 +1 +1 -112.32
6   -1 -1 +1 +1 +1 +1 +1 +1 +1 +1 -112.12
7   +1 +1 -1 -1 -1 -1 -1 -1 -1 -1 -112.12
8   -1 -1 -1 -1 -1 +1 +1 +1 +1 +1 -110.92
9   +1 +1 +1 +1 +1 -1 -1 -1 -1 -1 -110.92
10  -1 +1 +1 +1 +1 +1 +1 +1 +1 +1 -110.52
```

Figure 2.30: Output of Listing 1 on the real adiabatic quantum computer

systems. In this direction, the above study proposes the very first FIS able to be run on quantum annealers. The main goal of this study was not to demonstrate the advantage of using quantum annealers to implement a fuzzy logic system in the currently available machines but to prove that these quantum machines can be considered to perform a whole fuzzy inference process.

However, as soon as quantum annealers more resilient to noise and equipped with a larger number of qubits are available, the proposed approach could be used by distributing and parallelizing the computation of each operational unit in Fig- 2.29 over different sets of qubits, speeding up the whole inference process.

In the future, the plan is to investigate the results of the proposed approach on more complex fuzzy systems - with more variables and rules. Also, some real-world applications of the developed system will be explored and showcased. Moreover, this approach will be used to implement other inference methods such as zero-order TSK.

2.2.3 Fuzzy Inference on Digital Quantum Computers

In the previous section, a fuzzy inference system able to be run on quantum annealers was proposed. If on the one hand, this approach proves that quantum annealers are a suitable backend to compute fuzzy rules, on the other hand, it doesn't prove any computational advantages in running fuzzy systems over classical counterparts. In this subsection, this gap is filled by introducing the first methodology aimed at supporting the design and implementation of quantum algorithms able to perform a fuzzy inference process and take advantage of the computational benefits of this new computing paradigm. This methodology, named *Quantum Fuzzy Inference Engine (QFIE)*, allows the achievement of the above goals by introducing an oracle-based quantum algorithm (see Section 1.1.1.I). The implementation of QFIE is particularly relevant because it allows for an exponential speedup in the evaluation of fuzzy rules, in terms of oracle queries, evaluation of the antecedent part of a single fuzzy rule, and aggregation of rule outputs. In addition to the above advantages, the proposed methodology has another important side effect. It allows, for the first time, a digital quantum computer to be programmed linguistically. This result is also remarkable because, at present, the design of quantum algorithms is based on extensive knowledge of mathematical concepts, such as complex numbers and Hilbert vector spaces. Consequently, the proposed approach opens up new possibilities for using quantum computers for those who do not possess such skills, making this computational paradigm less difficult to use.

The design and implementation of a QFIE-based rule-based system are shown through a case study based on the inverted pendulum, in which it is possible to see how even a quantum computer programmed with fuzzy rules is able to identify in a correct way the point of equilibrium of the system. Successively, a real-world application like the control of real particle accelerators at CERN will be provided.

The section is organized as follows: firstly, a formulation of an FRBS as a boolean oracle is proposed; secondly, the QFIE algorithm is described; then, the pendulum case study is discussed; after this, it is shown how it is possible to control particle accelerators by using QFIE; lately, an analysis on which is the ideal quantum digital backend to execute QFIE is provided; finally, it is proposed a way to distribute QFIE according to the D-NISQ architecture proposed in [5].

2.2.3.I Oracle-based FRBS

This section introduces an oracle-based view of a FRBS. In this new vision, the rule base is replaced with an oracle able to reconstruct the relationships between the antecedent and consequent parts of the system. These changes result in the design of a new inference engine that uses the oracle to reconstruct the rule base of the system, before evaluating them (see Fig. 2.31).

For the sake of simplicity, the design of the oracle-based FRBS considers a MISO fuzzy system $\mathcal{S}$ with n input fuzzy variables $X_0, X_1, \dots, X_{n-1}$ and one output fuzzy variable Y. Each input variable X_j is defined by using m_j linguistic terms: $X_j = \{T_0^j, T_1^j, \dots, T_{m_j-1}^j\}$, with $j = 0, 1, \dots, n-1$. The output variable Y is defined by using m_Y linguistic terms: $Y = \{T_1^Y, T_2^Y, \dots, T_{m_Y}^Y\}$. The oracle-based design of an inference fuzzy system requires that input and output linguistic terms are encoded by binary strings. Then, for input variables, let $\{B_I^j\}_{j=0}^{n-1}$ be a family of sets, whose j-th component is defined as follows: $B_I^j = \{b_i^j | i = 0, 1, \dots, m_j - 1\}$, where b_i^j is the binary encoding of the number i, with $i = 0, 1, \dots, m_j - 1$, computed by using the conventional decimal to binary conversion, and such that $\bar{\eta}_j = |b_i^j| = \lceil \log_2(m_j) \rceil$ is the number of bits used to encode above strings. In this way, the j-th element of $\{B_I^j\}_{j=0}^{n-1}$ contains the binary representations b_i^j of the linguistic terms T_i^j related to the i-th input variable, where $i = 0, 1, \dots, n-1$. For the output variable, let B_O be a set defined as follows: $B_O = \{c_i | i = 1, 2, \dots, m_Y\}$, where c_i is the binary encoding of the number i, with $i = 1, 2, \dots, m_Y$, computed by using a one-hot encoding: c_i is a m_Y-bit binary string consisting of all 0's except for a 1 in the i-th position from left, and such that $|c_i| = m_Y$ is the number of bits used to encode above strings. Thus, B_O contains the binary representation c_i of the linguistic term T_i^Y related the output variable, where $i = 1, 2, \dots, m_Y$. Now, let $\mathcal{A}_\mathcal{S}$ and $\mathcal{C}_\mathcal{S}$ be two new sets defined as follows:

$$\mathcal{A}_\mathcal{S} = \textstyle\prod_{j=0}^{n} B_I^j; \quad \mathcal{C}_\mathcal{S} = B_O \cup \{c_0\} \tag{2.70}$$

where $\mathcal{A}_\mathcal{S}$ is a Cartesian product that contains the binary encoding of all possible antecedents that can be defined with the input variables $\{X_j\}_{j=0}^{n-1}$; $\mathcal{C}_\mathcal{S}$ contains the binary encoding of all possible consequent parts that can be defined with the output variable Y; and $c_0 = \{0\}^{m_Y}$ is a m_Y-bit string containing all 0's. Thus, the function

$$f : \mathcal{A}_\mathcal{S} \to \mathcal{C}_\mathcal{S} \tag{2.71}$$

is an oracle (as can be seen in (1.10)) that maps the antecedent parts to the consequent parts to compose fuzzy rules[7]. For a given $a \in \mathcal{A}_\mathcal{S}$, f returns a binary string other than $\{0\}^{m_Y}$ if a is a suitable antecedent part present for the system $\mathcal{S}$, $\{0\}^{m_Y}$ otherwise. Therefore, f is a functional view of a rule base that works as shown in the following example: let us suppose to have a system with two input variables X_0 and X_1, and a single output variable Y, each of which characterized by the following linguistic terms: $X_0 = \{T_0^0, T_1^0, T_2^0\}$, $X_1 = \{T_0^1, T_1^1, T_2^1\}$, and $Y = \{T_1^Y, T_2^Y, T_3^Y\}$. Then, the system's rule base can be reconstructed by running the following queries

[7]To be precise $\mathcal{A}_\mathcal{S}$ contains n-ple of binary strings, and not binary strings as required by the oracle definition. This abuse of notation, which can be easily fixed, does not compromise the behavior of f.

to the oracle:

$$\begin{aligned}
&f((00,00)) = 000; \quad && f((00,01)) = 010;\\
&f((00,10)) = 000; \quad && f((00,11)) = 000;\\
&f((01,00)) = 000; \quad && f((01,01)) = 100;\\
&f((01,10)) = 001; \quad && f((01,11)) = 000;\\
&f((10,00)) = 000; \quad && f((10,01)) = 000;\\
&f((10,10)) = 000; \quad && f((01,11)) = 000;\\
&f((11,00)) = 000; \quad && f((11,01)) = 000;\\
&f((11,10)) = 000; \quad && f((11,11)) = 000;
\end{aligned} \tag{2.72}$$

which corresponds to the following linguistic rules:

$$\begin{aligned}
&\text{IF}(X_0 \text{ is } T_1^0) \text{ and } (X_1 \text{ is } T_2^1) \text{ THEN } Y \text{ is } T_3^Y\\
&\text{IF}(X_0 \text{ is } T_0^0) \text{ and } (X_1 \text{ is } T_1^1) \text{ THEN } Y \text{ is } T_2^Y\\
&\text{IF}(X_0 \text{ is } T_1^0) \text{ and } (X_1 \text{ is } T_1^1) \text{ THEN } Y \text{ is } T_1^Y
\end{aligned} \tag{2.73}$$

Once the oracle-based rule-base concept is introduced, the fuzzy rule-based system architecture changes by considering the following steps:

1. Fuzzification
2. Inference Engine
   (a) Oracle-based Fuzzy Rules Preprocessing
   (b) Inference Operator
3. Defuzzification

Therefore, the main difference with the conventional approach is the new inference engine, composed of two sub-modules: *Oracle-based Fuzzy Rules Preprocessing* and *Inference Operator*. The Oracle-based Fuzzy Rules Preprocessing reconstructs the rule base through an appropriate number of queries to the Oracle f. The Inference

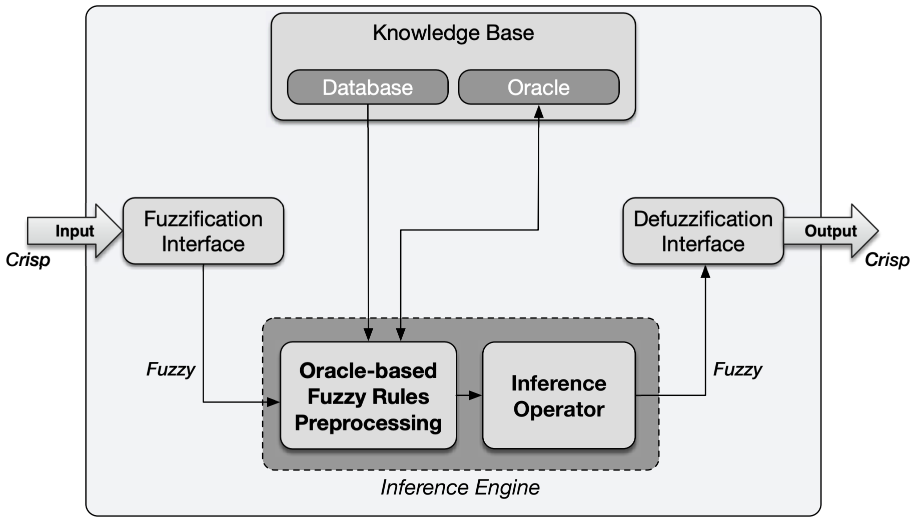


Figure 2.31: An oracle-based view of a fuzzy rule-based system.

Operator uses a set of fuzzy operators to apply the inference process to the rules identified by the oracle.

Although this new vision of a fuzzy system may seem strange and abstruse, it leads to three significant benefits: it allows the use of the superposition principle to develop an efficient quantum algorithm for a fuzzy inference engine; it defines a methodology to implement the above quantum algorithm via quantum circuits; it provides a framework by which to perform a computational complexity analysis of quantum and classical fuzzy inference engine.

2.2.3.II A Quantum Fuzzy Inference Engine

The design of this module leverages two main phases: 1) the encoding of the fuzzified values of the system in quantum states; 2) the set-up of a quantum oracle $\mathcal{O}_f$ corresponding to the Boolean function f defined in (1.10). As will be shown later, the definition of a quantum oracle $\mathcal{O}_f$ creates a positive side effect: in addition to selecting rules for the rule base of the system, it initiates a parallel process that allows QFIE to be computationally more efficient than its classical counterpart.

QFIE requires that the data on which a fuzzy inference engine operates, i.e., fuzzified values, be modeled through quantum states. This goal is achieved by mapping linguistic terms of each input fuzzy variable to the basis states of a Hilbert vector space, and using quantum amplitudes of these basis states to store the real values relative to the fuzzified inputs.

For the sake of simplicity, let us consider an FRBS system $\mathcal{S}$ composed of n input variables $X_0, X_1, \ldots, X_{n-1}$, and a single fuzzy variable Y. Each input variable X_j is defined by using m_j linguistic terms: $X_j = \{T_0^j, T_1^j, \ldots, T_{m_j-1}^j\} \cup \{N^j\}$, with $j = 0, 1, \ldots, n-1$. The linguistic term N^j is a dummy fuzzy set specifically added when the terms $\{T_0^j, T_1^j, \ldots, T_{m_j-1}^j\}$ do not represent a fuzzy partition, to allow the proposed quantum inference engine to properly work. The output variable Y is defined by using m_Y linguistic terms: $Y = \{T_1^Y, T_2^Y, \ldots, T_{m_Y-1}^Y\}$. Let B_I^j be the set containing the binary encodings of the linguistic terms of the j-th input fuzzy variable X_j, made as shown in the previous subsection, then the corresponding Hilbert computational basis is as follows:

$$\begin{aligned}\mathbf{B}_{\mathcal{H}_j} &= \{\left|b_i^j\right\rangle \mid b_i^j \in B_I^j, \text{ with } i = 0, 1, 2, \ldots, m_j - 1\} \\ &\cup \{\left|b_k^j\right\rangle \mid k = m_j, m_j + 1, \ldots, 2^{\eta_j} - 1\}\end{aligned}$$

where b_k^j is the binary encoding of k and $\eta_j = \lceil \log_2(|X_j|) \rceil$. Now, let $\{I^0, I^1, \ldots, I^{n-1}\}$ be a collection of fuzzy sets (e.g. singletons), where I^j represents the fuzzified j-th input to the inference engine, with $j = 0, 1, 2, \ldots, n-1$. Then, for each variable X_j the following values are calculated:

$$\alpha_i^j = \text{hgt}(T_i^j \cap I^j) \tag{2.74}$$

where $i = 0, 1, \ldots, m_j - 1$. Then, the quantum state of a fuzzy variable X_j to which a dummy fuzzy set has been added is described as follows:

$$\begin{aligned}|\psi_j\rangle &= \textstyle\sum_{i=0}^{m_j-1} \sqrt{\alpha_i^j} \left|b_i^j\right\rangle + \sum_{i=m_j}^{2^{\eta_j}-2} 0 \cdot \left|b_i^j\right\rangle \\ &+ \sqrt{1 - \left(\textstyle\sum_{i=0}^{m_j-1} \alpha_i^j\right)} \left|b_{2^{\eta_j}-1}^j\right\rangle\end{aligned} \tag{2.75}$$

where the first summation quantum encodes the set of linguistic terms of the variable X_j with respect to the fuzzified input I^j; the second summation encodes the basis states $\left|b^j_{m_j}\right\rangle, \left|b^j_{m_j+1}\right\rangle, \ldots, \left|b^j_{2^{\eta_i}-2}\right\rangle$ that are not associated with any linguistic term and, consequently, their amplitude is set to 0; in the last addend the basis state $\left|b^j_{2^{\eta_i}-1}\right\rangle$, defined as *garbage state* and associated to the dummy linguistic term N^j, ensures the normalization of the quantum state $|\psi_j\rangle$. If the dummy fuzzy set is not required, the quantum state of a fuzzy variable X_j is described as follows:

$$|\psi_j\rangle \quad = \quad \textstyle\sum_{i=0}^{m_j-1} \sqrt{\alpha^j_i}\,\left|b^j_i\right\rangle + \sum_{i=m_j}^{2^{\eta_j}-1} 0 \cdot \left|b^j_i\right\rangle \tag{2.76}$$

where only the summation that encodes the basis states $\left|b^j_{m_j}\right\rangle, \left|b^j_{m_j+1}\right\rangle, \ldots, \left|b^j_{2^{\eta_i}-1}\right\rangle$ that are not associated with any linguistic term is included to compensate for the fact that the number of linguistic terms of the variable X_j is not a power of 2. Also, this encoding procedure requires that for each variable X_j, $\sum_{i=0}^{m_j-1} \alpha^j_i \leq 1$, otherwise a preprocessing for scaling the values $\{\alpha^j_i\}_{i=0}^{m_j-1}$, with $j = 0, \ldots, n-1$, is necessary. For reasons of space and simplicity, let us consider the case in which the number of terms of the input fuzzy variables is a power of 2, and there is no need for normalization of $|\psi_j\rangle$. In these hypotheses the quantum state $|\psi_j\rangle$ related to variable X_j becomes:

$$|\psi_j\rangle = \sum_{i=0}^{m_j-1} \sqrt{\alpha^j_i}\,\left|b^j_i\right\rangle. \tag{2.77}$$

In order to define a quantum state that takes into account all n fuzzy variables involved, the tensor product of the individual states $\{|\psi_j\rangle\}_{j=0}^{n-1}$ is computed:

$$\begin{aligned}&|\psi_0, \psi_1, \ldots \psi_{n-1}\rangle = \\ &\sum_{j=0}^{m_0-1} \sum_{k=0}^{m_1-1} \cdots \sum_{l=0}^{m_{n-1}-1} \sqrt{\alpha^0_j \cdot \alpha^1_k \cdot \ldots \cdot \alpha^{n-1}_l}\,\left|b^0_j, b^1_k, \ldots, b^{n-1}_l\right\rangle\end{aligned} \tag{2.78}$$

This quantum state has an important property: its basis states correspond to all possible antecedent parts of fuzzy rules that can be generated with the variables $X_0, X_1, \dots, X_{n-1}$. Consequently, the squared amplitude of each basis state represents the degree of fulfillment of the related antecedent part.

As for the output fuzzy variable Y, it is embodied in a quantum state $|\psi_Y\rangle$ composed of m_Y qubits, each of which is initialized to $|0\rangle$:

$$|\psi_Y\rangle = |0_0, 0_1, \dots, 0_{m_Y-1}\rangle = |\bar{0}\rangle \, . \tag{2.79}$$

Once the input and output quantum states are defined, let us proceed to define the oracle $\mathcal{O}_f$. Oracles are implemented on quantum computers through unitary matrices (see (1.13)). A quantum oracle $\mathcal{O}_f$ for fuzzy inference engine is a $2^\xi \times 2^\xi$ unitary matrix, where:

$$\xi = \sum_{j=0}^{n-1} \eta_j + m_Y, \tag{2.80}$$

n is the number of input fuzzy variables, η_j is the number of bits used to binary encode the linguistic terms of the j-th input fuzzy variable, and m_Y is the number of the linguistic terms of the output variable. The main role of the oracle is to use the input state $|\psi_0, \psi_1, \dots, \psi_{n-1}\rangle$ to relate the actual antecedent parts of the system to the proper qubits of the quantum output state $|\psi_Y\rangle$, so as to reconstruct the rule base of the system. Thanks to this reconstruction, the quantum oracle computes a new output state $|\psi'_Y\rangle$ where the probability to measure the $|c_l\rangle$ encodes the output cutting value of the linguistic term T_l^Y. In order to better clarify the role of the oracle, let us consider a fuzzy system consisting of a single rule:

$$\begin{gathered}\text{IF}(X_0 \text{ is } T_{\bar{i}}^0) \text{ and } (X_1 \text{ is } T_{\bar{j}}^1) \text{ and } \dots \text{ and } (X_{n-1} \text{ is } T_{\bar{k}}^{n-1}) \\ \text{THEN } Y \text{ is } T_{\bar{l}}^Y\end{gathered} \tag{2.81}$$

Recalling that the input state is defined as in (2.78), the output state is $|\psi_Y\rangle = |\bar{0}\rangle$, and $c_{\bar{l}}$ is the one-hot encoding of output term $T_{\bar{l}}^Y$, the oracle $\mathcal{O}_f$ acts as follows:

$$
\begin{gathered}
|\psi'_Y\rangle = \mathcal{O}_f\,|\psi_0, \psi_1, \ldots, \psi_{n-1}\rangle\,|\psi_Y\rangle = \\
\mathcal{O}_f \left[\begin{array}{r} \displaystyle\sum_{i=0}^{m_0-1}\sum_{j=0}^{m_1-1}\cdots\sum_{k=0}^{m_{n-1}-1} \sqrt{\alpha_i^0\cdot\alpha_j^1\cdot\ldots\cdot\alpha_k^{n-1}} \\ \left|b_i^0, b_j^1, \ldots, b_k^{n-1}\right\rangle|\bar{0}\rangle \end{array} \right] = \\
\sum_{i,j,\ldots,k\neq\bar{i},\bar{j},\ldots,\bar{k}} \sqrt{\alpha_i^0\cdot\alpha_j^1\cdot\ldots\cdot\alpha_k^{n-1}}\left|b_i^0, b_j^1, \ldots, b_k^{n-1}\right\rangle|\bar{0}\rangle + \\
+\sqrt{\alpha_{\bar{i}}^0\cdot\alpha_{\bar{j}}^1\cdot\ldots\cdot\alpha_{\bar{k}}^{n-1}}\left|b_{\bar{i}}^0, b_{\bar{j}}^1, \ldots, b_{\bar{k}}^{n-1}\right\rangle|c_{\bar{l}}\rangle
\end{gathered} \tag{2.82}
$$

In other words, the oracle $\mathcal{O}_f$ creates a superposition of states $|\psi'_Y\rangle$, where the basis state related to the antecedent of the rule in the (2.81) is associated with the output state $|c_{\bar{l}}\rangle$, whereas all other basis states are associated with the state $|\bar{0}\rangle$. Once the quantum oracle has computed the new quantum state $|\psi'_Y\rangle$ relative to the input state $|\psi_0, \psi_1, \ldots, \psi_{n-1}\rangle$ associated with the rule in (2.81), the QFIE computes the probability $P_{c_{\bar{l}}}$ that the quantum measurement of $|\psi'_Y\rangle$ collapses into the state $|c_{\bar{l}}\rangle$ as follows:

$$
P_{c_{\bar{l}}} = \langle\psi'_Y|\,\mathcal{M}_{c_{\bar{l}}}^{\dagger}\mathcal{M}_{c_{\bar{l}}}\,|\psi'_Y\rangle = \alpha_{\bar{i}}^0\cdot\alpha_{\bar{j}}^1\cdot\ldots\cdot\alpha_{\bar{k}}^{n-1} \tag{2.83}
$$

where $\mathcal{M}_{c_{\bar{l}}} = |c_{\bar{l}}\rangle\langle c_{\bar{l}}|$ is the projection operator related to the basis state $c_{\bar{l}}$. The value $P_{c_{\bar{l}}}$ will be used by the inference operator (see Section 1.2.1) to cut the membership function of the output linguistic term $T_{\bar{l}}^Y$.

Now it will be shown how the oracle works on a system $\mathcal{S}$ characterized by a number of rules greater than one, where quantum computing exposes all its computational power by enabling the parallel computation of fuzzy rules. Indeed, since the quantum state $|\psi_0, \psi_1, \ldots\psi_{n-1}\rangle$ embodies all antecedents contained in $\mathcal{A}_{\mathcal{S}}$, then the quantum oracle can associate each antecedent with the correct consequent, if any, in parallel, as shown in the (1.13). Let us consider the following family of sets:

$$\mathcal{A}^i_{\mathcal{S}} = \{a \in \mathcal{A}_{\mathcal{S}} | a \rightarrow T^Y_i \text{ is a fuzzy rule of } \mathcal{S}\} \tag{2.84}$$

where $i = 1, \dots, m_Y$. Precisely, $\mathcal{A}^i_{\mathcal{S}}$ is the subset of $\mathcal{A}_{\mathcal{S}}$ containing all the antecedents $a \in \mathcal{A}_{\mathcal{S}}$ whose consequent is the linguistic term T^Y_i of the output variable Y. Consequently, $\mathcal{A}^0_{\mathcal{S}} = \mathcal{A}_{\mathcal{S}} - \cup_{i=1}^{m_Y} \mathcal{A}^i_{\mathcal{S}}$ is the set of antecedent parts not belonging to the rule base of $\mathcal{S}$. Finally, let F_a, where $a \in \mathcal{A}^i_{\mathcal{S}}$, be the fire strength of the rule having a as the antecedent part. Then, the action of the oracle $\mathcal{O}_f$ is:

$$\begin{aligned} |\psi'_Y\rangle &= \mathcal{O}_f \, |\psi_0, \psi_1, \dots, \psi_{n-1}\rangle \, |\bar{0}\rangle = \\ &= \sum_{i=1}^{m_Y} \sum_{a \in \mathcal{A}^i_{\mathcal{S}}} \left(\sqrt{F_a} \, |a\rangle \, |c_i\rangle\right) + \sum_{a \in \mathcal{A}^0_{\mathcal{S}}} \left(\sqrt{F_a} \, |a\rangle \, |\bar{0}\rangle\right) \end{aligned} \tag{2.85}$$

where the first addend represents a superposition of quantum states that relates the antecedents in $\mathcal{A}^i_{\mathcal{S}}$ to the output linguistic terms represented by the binary string c_i, where $i = 1, 2, \dots, m_Y$; the second addend is a superposition of states that relates antecedent in $\mathcal{A}^0_{\mathcal{S}}$ to the state $|\bar{0}\rangle$.

The state returned by the Oracle application is then used by the inference engine to compute the values P_{c_i}, corresponding to the probability of measuring the output quantum state as $|c_i\rangle$, to cut the membership function related to the output linguistic term T^Y_i, with $i = 1, 2, ..., m_Y$:

$$\begin{aligned} P_{c_i} &= \langle\psi'_Y| \, \mathcal{M}^\dagger_{c_i} \mathcal{M}_{c_i} \, |\psi'_Y\rangle = \\ &= \sum_{a \in \mathcal{A}^i_{\mathcal{S}}} \sum_{b \in \mathcal{A}^i_{\mathcal{S}}} \left(\sqrt{F_a} \, \langle a| \, \langle c_i|\right) \cdot \left(\sqrt{F_b} \, |b\rangle \, |c_i\rangle\right) = \\ &= \sum_{a \in \mathcal{A}^i_{\mathcal{S}}} \sum_{b \in \mathcal{A}^i_{\mathcal{S}}} \delta_{a,b} \sqrt{F_a} \sqrt{F_b} \, \langle a|b\rangle = \sum_{a \in \mathcal{A}^i_{\mathcal{S}}} F_a \end{aligned} \tag{2.86}$$

where $\delta_{a,b}$ is the Kronecker delta[8]. Once the P_{c_i} values have been calculated, the aggregation operator will use them appropriately to complete the inference process.

[8]The Kronecker delta is a function $\delta_{a,b}$ working as follows: $\delta_{a,b} = 1$ if a = b; 0 otherwise.

It is important to highlight that the unitarity of the quantum oracle $\mathcal{O}_f$ is ensured under two constrictions: firstly, the antecedent part of each rule has to be composed of a combination of clauses involving all the input variables of the system, and secondly, it is required that there are no conflicting rules. Indeed, the former ensures that each input of f has the same form (bit-strings of equal lengths); the latter results in the fact that each input $(b_i^0, b_j^1, \ldots, b_k^{n-1})$ has a unique output c_l ensuring that f is a bijective function.

At this point, starting from the values $\{P_{c_i}\}_{i=1}^{m_Y}$, the inference operator applies two sequential operations, namely implication and aggregation. The implication operator computes the output fuzzy set $\mu_i^Y(x)$ related to the i-th linguistic term of the output variable Y:

$$\mu_i^Y(x) = \min(P_{c_i}, \mu_{T_i^Y}(x)) \qquad \forall x \in U_i \tag{2.87}$$

where U_i is the universe of the discourse of the fuzzy set $\{(x, \mu_{T_i^Y}(x)) | x \in U_i\}$ that describes the linguistic term T_i^Y of the output variable Y, with $i = 1, 2, \ldots, m_Y$.

The aggregation operation combines, the fuzzy membership functions $\{\mu_i^Y(x)\}_{i=1}^{m_Y}$ to compute the output fuzzy set $S^Y = \{(x, \mu_Y(x)) | x \in \cup_{i=1}^{m_Y} U_i\}$ as follows:

$$\mu_Y(x) = \max_{i=1,2,\ldots,m_Y}(\mu_i^Y(x)) \qquad \forall x \in \bigcup_{i=1}^{m_Y} U_i \tag{2.88}$$

Then, the output crisp value can be obtained by applying the preferred defuzzification method on the fuzzy set S^Y, such as the CoG approach reported in (1.31).

Considering this, it is possible to analyze how to design a quantum circuit implementing the QFIE algorithm.

Firstly, according to the encoding procedure described above, for a FRBS system composed of n input variables $X_0, X_1, \ldots, X_{n-1}$ each defined with m_j linguistic terms, and one output variable Y defined with m_Y linguistic terms, $n+1$ quantum registers have to be initialized. In detail, regarding the input variables, a collection of n quantum registers is defined as follows:

$$\{\mathcal{QR}_{X_j} = \{q^0_{X_j}, q^1_{X_j}, \dots, q^{\eta_j-1}_{X_j}\}\}_{j=0}^{n-1} \tag{2.89}$$

where each set $\mathcal{QR}_{X_j}$, with $j = 0, 1, \dots, n-1$ is composed of $\eta_j = \lceil \log_2(|X_j|) \rceil$ qubits. With regard to the output variable, a single quantum register $\mathcal{QR}_Y = \{q^1_Y, q^2_Y, \dots, q^{m_Y}_Y\}$ composed of m_Y qubits, is defined. Thus, the quantum state related to the input variable (see (2.75)) is initialized by using the approach proposed in [158], where Shende et al. introduced a way of initializing quantum registers based on quantum multiplexor (QMUX) that is asymptotically optimal in the number of multi-qubits gates. The circuit block related to this initialization procedure for the variable X_j, with $j = 0, 1, \dots, n-1$, is shown in Fig. 2.32.

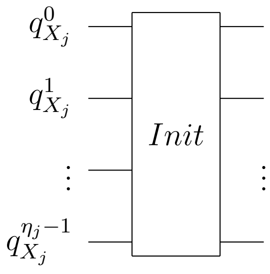


Figure 2.32: Initialization of quantum register $\mathcal{QR}_{X_j}$.

Regarding the quantum register related to the output variable, it is initialized by setting all the qubits in the state $|0\rangle$.

After completing the description of the initialization of the input and output quantum states, let us show how to construct the quantum circuit related to the $\mathcal{O}_f$ oracle introduced in (2.85). The oracle circuit is implemented by encoding each rule $(b^0_{\overline{i}}, b^1_{\overline{j}}, \dots, b^{n-1}_{\overline{k}}) \rightarrow c^Y_{\overline{l}}$ of the system by applying a MCX quantum gate. The control state of the MCX gate is the basis state related to the binary encoding of the antecedent part $(b^0_{\overline{i}}, b^1_{\overline{j}}, \dots, b^{n-1}_{\overline{k}})$ of the rule, used as follows: if the i-th bit of the j-th variable of the antecedent part is zero then the state of the qubit $q^{\eta_j-1-i}_{X_j}$ is inverted using the quantum gate X, otherwise the state of the qubit $q^{\eta_j-1-i}_{X_j}$ is left unchanged[9]. Regarding the target qubit related to the consequent part $c_{\bar{l}}^{Y}$, it is set to $q_{Y}^{\bar{l}}$. To better clarify the above circuital description, let us consider the following rule:

$$\text{IF } (X_0 \text{ is } T_0^0) \text{ and } (X_1 \text{ is } T_2^1) \text{ THEN } Y \text{ is } T_2^Y \tag{2.90}$$

The oracle view of the above rule is:

$$f((00,10)) = 010 \tag{2.91}$$

which corresponds to the quantum circuit shown in Fig. 2.33.

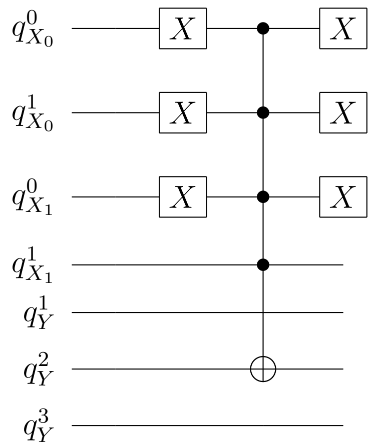


Figure 2.33: Implementing the rule $f((00,10)) = 010$ as quantum circuit.

In reference to the circuit version of QFIE it is important to take into account two important aspects. Since the rules are executed sequentially, it is necessary to reset the state of the qubits related to the antecedent part of the rules, when the computation switches from one rule to the next; this can be done using the reversible nature of the quantum gates, as shown in Fig. 2.33, where the X quantum gates are applied symmetrically with respect to the MCX gate. Furthermore, the nature of quantum measurement does not allow a quantum circuit to compute the probabilities P_{c_i}, with $i = 1, 2, \ldots, m_Y$, used by the QFIE inference operator. However,

[9]Note, that it is common in quantum computation using an indexing of qubits that is opposite to the indexing of classical bits.

these probabilities can be estimated by running the circuit a constant number of times (shots) and reconstructing a discrete probability distribution from the results obtained from each run of the circuit. In particular, the definition of Hellinger fidelity [138] can be used to retrieve the number of shots N_S required to reconstruct a certain discrete probability distribution with a given level of fidelity according to the following equation:

$$N_S \approx \frac{N_{c_i}}{(1-F)^2} \tag{2.92}$$

where N_{c_i} is the number of possible outcomes of the QFIE circuit execution, and F is the level of fidelity desired.

2.2.3.III QFIE Computational Complexity and Limitations in NISQ Era

In this subsection the computational complexity of QFIE is assessed in terms of the number of the input fuzzy variables n, and compared to the complexity of the classical oracle-based fuzzy inference system introduced in Section 2.2.3.I. Moreover, it will be highlighted some limitations related to the implementation of QFIE on NISQ devices.

Firstly, since both QFIE and the classical system use the same fuzzification and defuzzification interfaces, the discussion will focus on the complexity analysis related to the evaluation of the rules and the aggregation of the obtained fuzzy outputs. A common approach to evaluating the complexity of an oracle-based quantum algorithm is to count the required number of queries to the oracle. For instance, Grover's algorithm requires $O(\sqrt{N})$ queries to the oracle for finding the desired item in the N-dimensional unsorted database, or the Deutsch-Josza algorithm requires $O(1)$ queries to the oracle for establishing if a given Boolean function is balanced or constant. For the sake of simplicity, let us consider a fuzzy system $\mathcal{S}$ composed of n input fuzzy variables each of them having m linguistic terms, and one output fuzzy variable Y composed of m_y linguistic terms. Then, the cardinalities of the sets $\mathcal{A}_\mathcal{S}$ and $\mathcal{C}_\mathcal{S}$

defined in (2.70) are m^n and m_y, respectively.

Let us consider the worst case in which the rule base of $\mathcal{S}$ consists of all possible m^n rules definable by the antecedent parts of $A_{\mathcal{S}}$. Then, the classical oracle-based fuzzy inference system needs to perform m^n queries to the oracle f in order to reconstruct the rule base of $\mathcal{S}$. Once the rule base is reconstructed, the degree of fulfillment of each rule is calculated according to (1.27). This operation requires the computation of $n-1$ t-norms for each rule. Then, m^n implication functions have to be computed to obtain the output fuzzy set of each rule according to (1.28). Finally, the aggregation operator combines all these fuzzy sets according to (1.29), in which m^n max operations are carried out. Overall, the computational complexity of a fuzzy inference that equips a classical fuzzy oracle-based engine is $O(m^n)$, since there are to be considered $O(m^n)$ queries to the oracle, $O(m^n)$ t-norm operations for computing the degree of fulfillment of the whole rule base, $O(m^n)$ implication operations, and $O(m^n)$ disjunction operations.

On the other hand, the quantum implementation of an oracle-based fuzzy inference engine requires initializing n quantum states related to the n input fuzzy variables. The computational cost of Shende's initialization approach is $O(4^q)$, where q is the number of qubits to initialize. Since QFIE uses $\lceil \log_2(m) \rceil$ qubits for each one of the n input fuzzy variables, the total cost of this operation is:

$$\textstyle\sum_{j=0}^{n-1} O(4^{\lceil \log_2(m) \rceil}) = \sum_{j=0}^{n-1} O(m^2) = O(n \cdot m^2) = O(n).$$

Successively, the degree of fulfillment of all rules is computed in $O(1)$ thanks to the application of the tensor product between the Hilbert spaces related to the previously initialized quantum registers. Moreover, QFIE needs $O(1)$ query to the quantum oracle f to reconstruct the rule base of $\mathcal{S}$. Even if the estimation of the probabilities $\{P_{c_i}\}_{i=1}^{m_y}$ requires to repeat above query N_s times, N_s is a constant value. Finally, as shown in (2.87), m_y implication operations are performed to compute the output fuzzy sets, which are finally aggregated according to (2.88), by using m_y max operations. Summing up, the computational complexity of QFIE is $O(n + m_y) = O(n)$, since there are to be considered $O(1)$ queries to the oracle[10], $O(n)$ operations related to the initialization of quantum states, $O(m_y)$ implications and $O(m_y)$ disjunction operations for obtaining the whole output fuzzy set. Overall, the above complexity analysis demonstrates that, for the same computational cost required to perform classical and quantum oracle, QFIE results in an exponential quantum advantage in fuzzy inference.

This quantum advantage is currently limited by some technological issues that plague NISQ computers. As seen in sec. 1.1.1.III, these devices are characterized by a high level of noise that adversely affects calculations. This issue turns out to be particularly serious for quantum circuits containing a high number of multi-controlled gates, as those required for computing QFIE in real big data contexts. Indeed, during the transpiling phase, these gates have to be decomposed in terms of CNOT gates that can be actually executed on real superconductive hardware. This decomposition procedure generates deeper quantum circuits that often, cannot be executed on the hardware reliably, due to the high level of noise of CNOT gates (on average current CNOTs have error rates of 10^{-2}). At the time of writing, such a level of noise does not allow QFIE to handle fuzzy systems composed of a large number of rules because, otherwise, the circuit resulting from the transpiling phase would be too prone to noise and return erroneous results. Moreover, NISQ devices are equipped with a limited number of qubits and they are not yet ready to deal with big data contexts. For example, considering a system characterized by a thousand input variables, each modeled by 7 fuzzy sets, QFIE would require more than $1000 \cdot \lceil \log_2 7 \rceil = 3000$ qubits to operate, and current quantum computers do not yet achieve such performance. Consequently, although QFIE shows important theoretical advantages, it will have to wait until the post-NISQ quantum era to make it fully applicable. By virtue of

[10] N_s is a constant value absorbed by O(1).

such limitations in the next section, QFIE will be tested on a conventional case study for FRBSs, such as the inverted pendulum, where the number of variables and rules allows for a reliable QFIE design of the system.

2.2.3.IV A Case Study: The Inverted Pendulum

This section shows the suitability of QFIE to be used as a fuzzy inference engine of an FRBS for the inverted pendulum. It will be provided a discussion about the inverted pendulum system, the usage of QFIE in this application scenario, and its implementation through an IBM Quantum backend. Finally, QFIE will be compared with a traditional Mamdani fuzzy system in order to show its good performance. Due to the technological limitations of NISQ quantum devices, experiments have been carried out by noise-free simulation and using a constrained number of qubits, which, at present, does not allow dealing with complex systems characterized by a large number of variables or rules.

Firstly, let us describe the environment used as the benchmark control problem. The inverted pendulum system consists of a pendulum in a dynamic equilibrium, i.e. it continuously slips from the vertical and is brought back through the application of torque resulting from a motor to which is applied a certain current I. The dynamic of such a system is regulated by the following two equations [86]:

$$\theta_{new} = \theta_0 + w_0 t + \frac{t^2}{2}\left(\frac{3\tau}{ml^2} I - \frac{3g}{2l}\cos(\theta_0)\right)$$

$$\omega_{new} = \omega_0 + t\left(\frac{3\tau}{ml^2} I - \frac{3g}{2l}\cos(\theta_0)\right)$$

where g is the gravitational constant, τ the torque of the motor, m the pendulum mass, l its length and t the time step considered. Fig. 2.34 reports the setup of the inverted pendulum system.

In this context, the role of a FRBS is to compute the right current I according

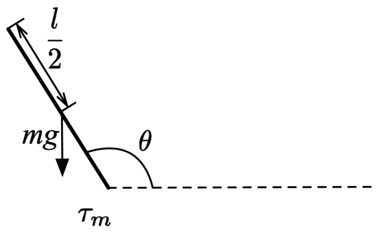
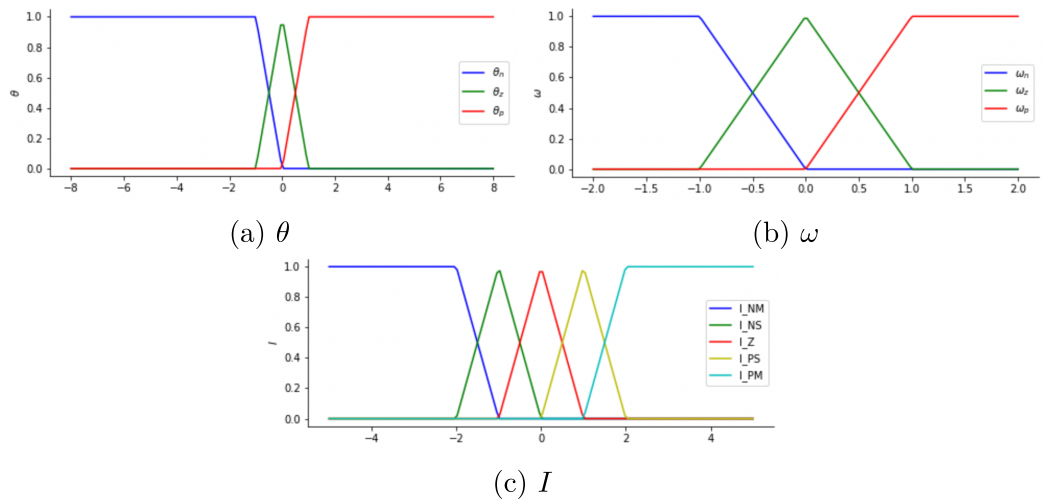


Figure 2.34: Inverted pendulum setup.

to the detected angular position θ of the pendulum and its angular velocity ω. This new value of I is used to determine the new angular position θ_{new} and velocity of the pendulum ω_{new}. The database of a traditional FRBS for the inverted pendulum contains two input fuzzy variables, θ, and ω, and an output one, I, partitioned as shown in the Fig. 2.35. In detail, the input variables are characterized by the following linguistic terms: *negative* (N), *zero* (Z), and *positive* (P). The output variable uses the following linguistic terms: *negative medium* (NM), *negative small* (NS), *zero* (Z), *positive small* (PS) and *positive medium* (PM). The normalization factors are $\frac{\pi}{4}$rad, $\pi\frac{\text{rad}}{\text{s}}$ and 1 A for θ, ω and I respectively, this means that, for instance, a value of $\theta = 1$ corresponds to an angle of $\theta = \frac{\pi}{4}$ of the pendulum with respect the vertical.



(a) θ (b) ω

(c) I

Figure 2.35: Fuzzy partitions for fuzzy variables (a) θ, (b) ω and (c) I.

Finally, the rule base of a conventional Mamdani-based FRBS is composed of the following rules:

Table 2.16: Mapping between binary strings and fuzzy linguistic terms for each variable used in the QFIE-based inverted pendulum.

θ			ω			$\boldsymbol{I}$		
N	$\rightarrow$	00	N	$\rightarrow$	00	NM	$\rightarrow$	00001
Z	$\rightarrow$	01	Z	$\rightarrow$	01	NS	$\rightarrow$	00010
P	$\rightarrow$	10	P	$\rightarrow$	10	Z	$\rightarrow$	00100
						PS	$\rightarrow$	01000
						PM	$\rightarrow$	10000

1. If θ is Z and ω is Z Then I is Z;
2. If θ is Z and ω is N Then I is PS;
3. If θ is Z and ω is P Then I is NS;
4. If θ is P and ω is Z Then I is NS;
5. If θ is P and ω is P Then I is NM;
6. If θ is P and ω is N Then I is Z;
7. If θ is N and ω is Z Then I is PS;
8. If θ is N and ω is P Then I is Z;
9. If θ is N and ω is N Then I is PM.

Now, let us describe how QFIE works to model the inverted pendulum system. Precisely, let us define the sets B_I^θ, B_I^ω, and B_O^I as follows:

$$\begin{aligned}
B_I^\theta &= \{00, 01, 10\} \\
B_I^\omega &= \{00, 01, 10\} \\
B_O^I &= \{10000, 01000, 00100, 00010, 00001\}
\end{aligned}$$

where Table 2.16 reports the mapping between binary strings in B_I^θ, B_I^ω, and B_O^I and the collection of linguistic terms of the database.

This mapping enables the definition of an oracle f as follows:

$$\begin{aligned}
f((01,01)) &= 00100;\\
f((01,00)) &= 01000;\\
f((01,10)) &= 00010;\\
f((10,01)) &= 00010;\\
f((10,10)) &= 00001;\\
f((10,00)) &= 00100;\\
f((00,01)) &= 01000;\\
f((00,10)) &= 00100;\\
f((00,00)) &= 10000;
\end{aligned}$$

Now, let us move from the classical to the quantum world by introducing the appropriate set of basis states of the Hilbert vector spaces used to model quantum states in QFIE:

$$\begin{aligned}
\mathbf{B}_{\mathcal{H}_\theta} &= \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}\\
\mathbf{B}_{\mathcal{H}_\omega} &= \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}\\
\mathbf{B}_{O_I} &= \{|00001\rangle, |00010\rangle, |00100\rangle, |01000\rangle,\\
&\quad |10000\rangle\}
\end{aligned}$$

where $\mathbf{B}_{O_I}$ is a subset of the canonical basis of a Hilbert space of dimension 2^5, containing only the output states of interest for the implementation of QFIE in the context of the inverted pendulum; $\mathbf{B}_{\mathcal{H}_\theta}$ and $\mathbf{B}_{\mathcal{H}_\omega}$ are the basis states of the 2^2-dimensional Hilbert spaces related to the variable θ and ω, respectively, as depicted in (2.74). Since for both the variables θ and ω the linguistic terms are 3 and in both cases they form a fuzzy partition in which no dummy fuzzy set has to be considered, the state $|11\rangle$ included in $\mathbf{B}_{\mathcal{H}_\theta}$ and $\mathbf{B}_{\mathcal{H}_\omega}$ is associated with no linguistic term and, for this reason, the corresponding amplitude is set to 0, as reported in (2.76). Starting from these definitions it is possible to introduce a $2^9 \times 2^9$ unitary matrix acting as the quantum oracle $\mathcal{O}_f$.

In order to show the quantum execution of the rules through the oracle $\mathcal{O}_f$, let us consider an example in the absence of gravity where the initial state of the inverted pendulum is $\theta = -0.5$ and $\omega = 0.5$. After fuzzification of the system inputs, the following values are calculated using (1.26):

$$\begin{array}{lll} \alpha_N^\theta = 0.5; & \alpha_Z^\theta = 0.5; & \alpha_P^\theta = 0 \\ \alpha_N^\omega = 0; & \alpha_Z^\omega = 0.5; & \alpha_P^\omega = 0.5 \end{array}$$

Recalling the encoding procedure introduced in the description of the QFIE algorithm, the quantum input state of the system is defined as:

$$\begin{aligned} |\psi_\theta, \psi_\omega, \psi_I\rangle &= \sqrt{0.5 \cdot 0.5}\,|00, 01, 00000\rangle + \\ &+ \sqrt{0.5 \cdot 0.5}\,|00, 10, 00000\rangle + \\ &+ \sqrt{0.5 \cdot 0.5}\,|01, 01, 00000\rangle + \\ &+ \sqrt{0.5 \cdot 0.5}\,|01, 10, 00000\rangle \end{aligned}$$

The oracle $\mathcal{O}_f$ takes the quantum state $|\psi_\theta, \psi_\omega, \psi_I\rangle$ as input and works as follows:

$$\begin{aligned} \mathcal{O}_f\,|\psi_\theta, \psi_\omega, \psi_I\rangle &= \sqrt{0.5 \cdot 0.5}\,|00, 01, 01000\rangle + \\ &+ \sqrt{0.5 \cdot 0.5}\,|00, 10, 00100\rangle + \\ &+ \sqrt{0.5 \cdot 0.5}\,|01, 01, 00100\rangle + \\ &+ \sqrt{0.5 \cdot 0.5}\,|01, 10, 00010\rangle \end{aligned}$$

Next, a series of appropriate quantum projection operations compute the following values:

$$
\begin{aligned}
P_{c_{NM}} &= 0 \\
P_{c_{NS}} &= 0.25 \\
P_{c_{Z}} &= 0.5 \\
P_{c_{PS}} &= 0.25 \\
P_{c_{PM}} &= 0
\end{aligned}
$$

Finally, QFIE's inference operator and the CoG defuzzification use $P_{c_{NM}}$, $P_{c_{NS}}$, P_{c_Z}, $P_{c_{PS}}$ and $P_{c_{PM}}$ to calculate the output value for the variable I and the corresponding values of θ_{new} and ω_{new} after an interval of time $t = 0.05$ s (the following values are computed according to (2.93) considering $g = 0$):

$$
\begin{aligned}
I &= 0 \\
\theta_{new} &= -0.475 \\
\omega_{new} &= 0.5
\end{aligned}
$$

The next step is to show QFIE at work on a quantum device by means of its circuit representation.

According to the implementation of QFIE via quantum circuits, for the inverted pendulum system, three quantum registers are required. Namely, $\mathcal{QR}_\theta = \{q_\theta^0, q_\theta^1\}$ and $\mathcal{QR}_\omega = \{q_\omega^0, q_\omega^1\}$ are the two registers needed to encode the input values after the fuzzification process, while $\mathcal{QR}_I = \{q_I^1, q_I^2, q_I^3, q_I^4, q_I^5\}$ is the register required to encode the output probabilities that are then used from the inference operator and the CoG defuzzification. Considering this, the input quantum registers can be initialized and the oracle $\mathcal{O}_f$ is implemented by means of MCX gates. Fig. 2.36 shows the implemented circuit, where a measurement operation on each qubit of $\mathcal{QR}_I$ is added after the implementation of the oracle.

At this point, the aforementioned quantum circuit has to be executed for a certain number of shots to estimate the probabilities $\{P_{c_i}\}_{i=1}^{m_y}$ encoded in the output register.

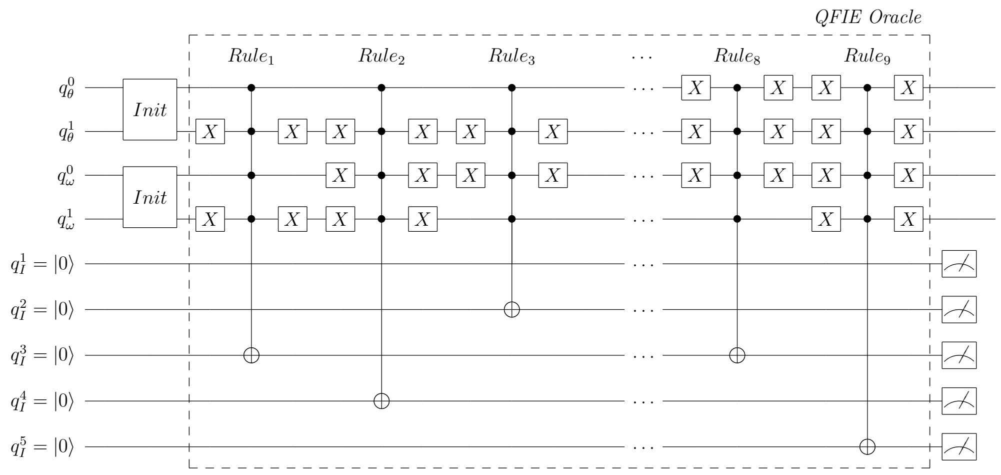


Figure 2.36: Quantum circuit implementing QFIE for controlling the inverted pendulum system.

As an example, Fig. 2.37 reports estimated probabilities after executing[11] the circuit for $N_S = 8000$ shots. N_S is computed according to (2.92) where $N_{c_i} = 5$ and the desired level of fidelity is $F = 0.975$ and using as input of QFIE the values in (2.93).

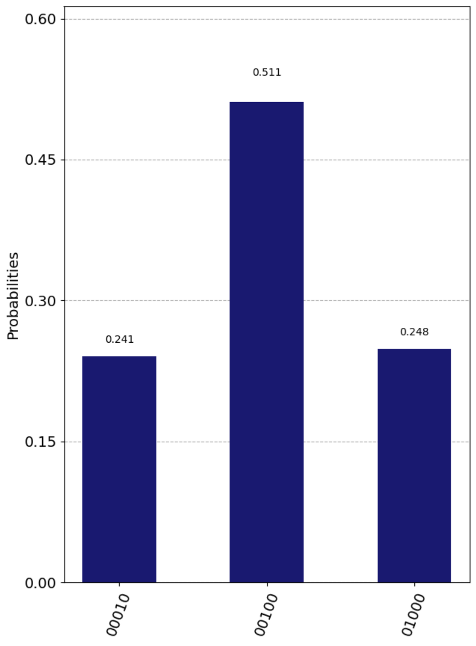


Figure 2.37: Estimation of probabilities used as cutting values of the linguistic terms related to the output variable I.

[11]In our experimentation the quantum circuit implementing QFIE have been simulated using the IBM *Qasm Simulator*.

Each bar in the histogram represents the frequency with which the related basis state was measured. Considering the mapping among basis states and linguistic terms shown in Table 2.16, the obtained results reflect the theoretical expectation of the values $\{P_{c_i}\}_{i=1}^{m_y}$ reported in (2.93). Finally, the value of the current that has to be applied to the pendulum is computed by means of the inference operator and the CoG defuzzification method and the new values of θ and ω can be computed consequently as follows:

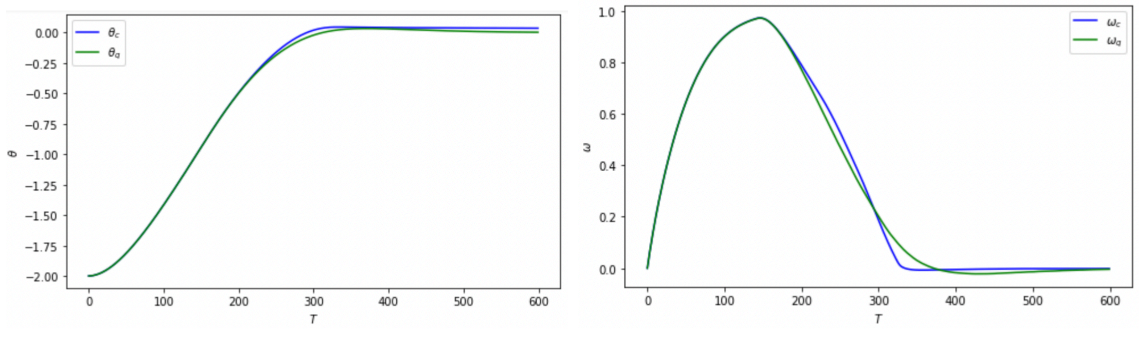
$$\begin{aligned} I &= -0.006 \\ \theta_{new} &= -0.475 \\ \omega_{new} &= 0.499 \end{aligned}$$

Finally, to investigate the suitability of QFIE to act as a fuzzy inference engine, a comparison with a traditional Mamdani fuzzy system has been carried out. In detail, such a fuzzy system uses the min operator as the T-norm for aggregating fuzzy propositions in the antecedent part of rules, the min operator as the implication method, and the max operator for accumulating the outputs of fuzzy rules in a final output fuzzy set to be undergone to defuzzification. In our experiments, CoG defuzzification is used for both the classical Mamdani system and the one equipped with QFIE.

The comparison is carried out in two different scenarios: 1) in the absence of gravity ($g = 0\frac{\mathrm{m}}{\mathrm{s}^2}$); 2) with gravity ($g = 9,8\frac{\mathrm{m}}{\mathrm{s}^2}$).

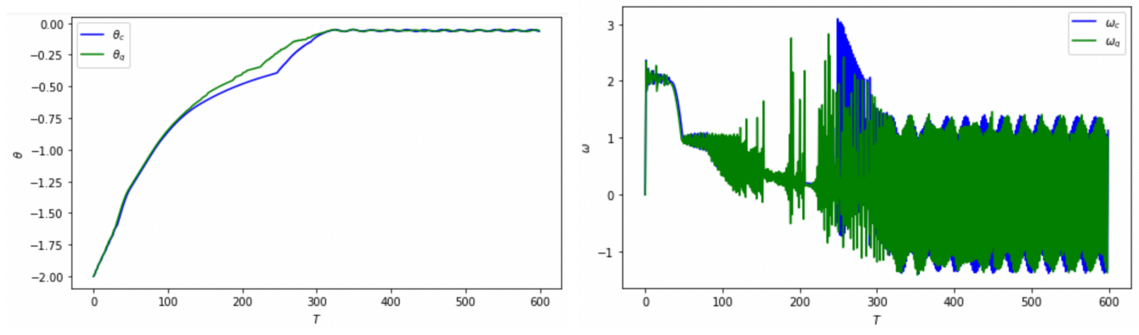
In the first experimentation, the pendulum environment was: $l = 30\mathrm{cm}$, $m = 28\mathrm{g}$, $\tau = 112\mathrm{K_g}\frac{m}{A}$. QFIE is iterated for $T = 600$ time steps with $t = 0.01\mathrm{s}$ and the values of θ and ω obtained at each iteration have been compared to the ones obtained in the same condition using the traditional Mamdani fuzzy system. The starting point for both the classical and the quantum system is: $\theta = -2$, $\omega = 0$. As shown by plots in Fig. 2.38, both QFIE-based and classical Mamdani fuzzy systems reach the vertical equilibrium of the pendulum approximately after 3s.

In the second experiment (with gravity), the pendulum environment consists of



(a) $\theta(T)$ (b) $\omega(T)$

Figure 2.38: Trend of θ (in radiant (rad)) in (a) and ω (in $\frac{\text{rad}}{\text{s}}$) in (b) under the action of classical (blue line) and quantum fuzzy control (green line) over the time in the case of $g = 0\frac{m}{s^2}$.



(a) $\theta(T)$ (b) $\omega(T)$

Figure 2.39: Trend of θ (in rad) in (a) and ω (in $\frac{\text{rad}}{\text{s}}$) in (b) under the action of classical (blue line) and quantum fuzzy control (green line) over the time in the case of $g = 9.8\frac{m}{s^2}$.

the same configuration as for the case in the absence of gravity, except for the motor torque τ which is increased by a factor $\times 100$. Also in this case the starting point for both the systems is: $\theta = -2$ and $\omega = 0$. Fig. 2.39 reports the trends of θ and ω during the control action of $T = 600$ time steps, each one of $t = 0.01$s. Also in this case, both FRBSs are able to keep the pendulum in its vertical position after around 3 seconds, although there are oscillations due to gravity.

In conclusion, the two case studies show that QFIE is able to suitably regulate an inverse pendulum like a conventional Mamdani fuzzy system.

In the next section, the suitability of QFIE in controlling complex environments will be assessed thanks to the development of control systems able to control for the very first-time particle trajectories in two accelerators of the CERN (European Organization for Nuclear Research) facilities.

2.2.3.V Quantum Fuzzy Inference Engine for Particle Accelerators Control

Despite the computational advantage proven in the previous sections, until now QFIE has been only tested by means of ideal simulations and on a control application characterized by a simple dynamic, such as an inverse pendulum, and, as a result, there is no concrete evidence of its operational usability.

This section fills this gap by performing two important steps: for the very first time, QFIE will be run on an actual Quantum device from those available on IBM Quantum and secondly, it will be shown that QFIE can be used to efficiently manage complex systems such as those related to particle accelerator facilities at CERN. The suitability of QFIE for the automatic control of particle accelerators has been demonstrated in two different experimental settings, related to two different CERN facilities: the T4 target station at the CERN Super Proton Synchrotron (SPS) fixed target physics beam line and the Advanced Proton Driven Plasma Wakefield Acceleration Experiment (AWAKE).

The first beam line involves directing a proton beam towards a target with a single control variable, while the second beam line, related to the AWAKE experiment, entails ten control parameters to be manipulated for electron beams. The project primarily utilizes accurate simulations of these beam lines to develop and test the implementation of QFIE for control purposes. The environments for these simulations are constructed using the OpenAI GYM template [48]. Finally, by the end of this section, it will be shown the performance of the QFIE-based FRBS when employed to control in real time the actual AWAKE beam line.

Target Steering Environment Control

This section aims to show the application of QFIE in controlling the 1-dimensional beam target steering environment based on the beam optics of the TT24-T4 transfer line at CERN [60]. This line is about 170 m long and transports protons with a momentum of 400 GeV/c from the Super Proton Synchrotron (SPS) to some of the fixed-target physics experiments installed in the CERN North Area. TT24 is equipped with several dipole and quadrupole magnets to steer and focus the beam, various beam position monitors (BPMs), and the actual target, which is placed at the end of the line. The objective of the task is to optimize the number of particles hitting the target by tuning the first dipole magnet in the line to maximize the event rates in the particle detectors. The left-hand side of Fig. 2.40 shows the relevant elements of TT24 together with horizontal beam trajectories obtained from tracking simulations for three different settings of the main bending dipole (orange). Depending on the dipole deflection angle, the particles hit the target (grey, hatched) at different horizontal positions, as illustrated by the zoomed view on the right-hand side of the figure. There are focusing (purple) and defocusing (olive) quadrupoles along the beam line to keep the beam particles confined.

Overall, the QFIE-based controller implemented aims to deflect the beam via the magnetic dipole according to two input variables such as the position reading of one of the BPMs installed in the beam line (cyan) X_{bpm} and the desired target position X_T. The allowed range of deflection angles Y is [-140, 140] μrad.

Two scenarios have been considered: in the first configuration (C1) X_T was any value in the range $[-1.5, 1.5]$ mm, while in the second configuration (C2), X_T was set to zero, and therefore X_{bpm} was the only input variable of the system. This simplification of the environment enables the construction of smaller quantum circuits for QFIE that can be reliably executed on real quantum hardware. Finally, to assess

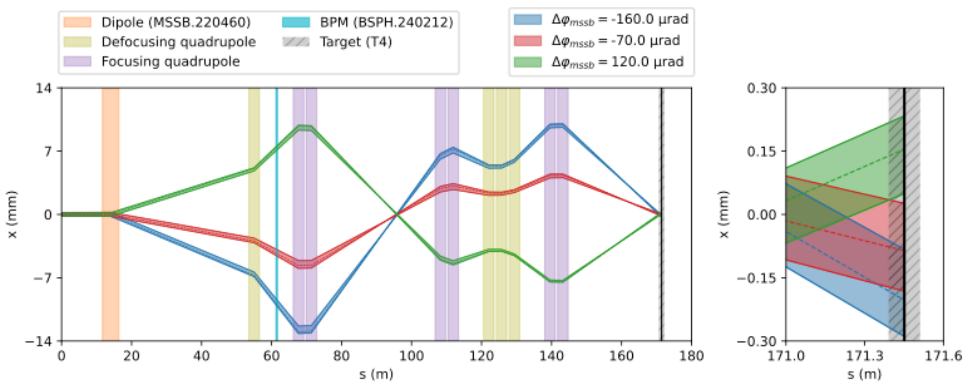


Figure 2.40: 1-dimensional beam target steering task at the CERN TT24-T4 beam line. Left: Horizontal beam trajectories obtained from tracking simulations are shown for three different settings of the main deflecting dipole (orange). Right: Zoomed view on the target (grey, hatched) region showing the horizontal position of impact of the beam for the three settings of the main dipole [155].

how well the beam is hitting the target, the following reward function is considered:

$$\mathcal{R} = -(1 - I) \tag{2.93}$$

where I represents the intensity of the Gaussian beam in the range $X_T \pm 3\sigma$, where σ is the beam size following from a fixed emittance of 11.8 nm rad. The beam is hitting the perfect target position when $\mathcal{R} = 0$.

The fuzzy partitions and the rule base used by QFIE in controlling C1 and C2 are reported in Fig. 2.41 and Fig. 2.42, respectively. For C1, five linguistic terms were considered for each variable: Negative (N), Medium Negative (MN), Zero (Z), Medium Positive (MP), and Positive (P). As a consequence, the number of fuzzy rules having as antecedent an *and* connection of two linguistic terms related to the different input variables is 25. For C2, three linguistic terms were considered for the input and the output variable: Negative (N), Zero (Z), and Positive (P). This leads to a set of 3 fuzzy rules. The limited number of fuzzy rules in C2 enables the execution of QFIE also on real NISQ devices, while their high level of noise forces the control of C1 via noiseless simulations of QFIE.

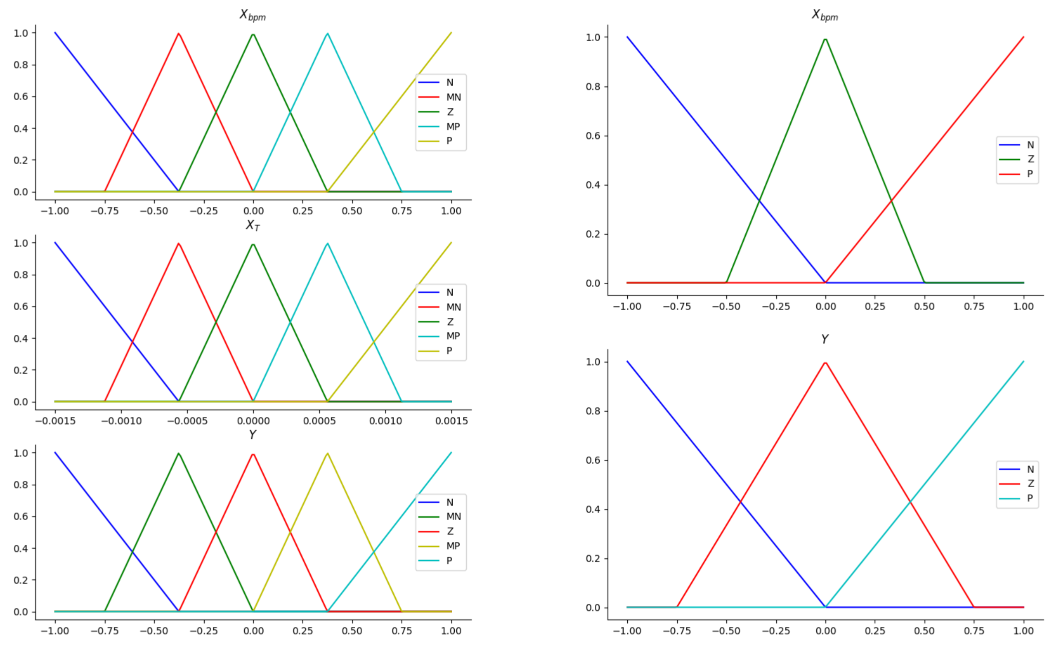
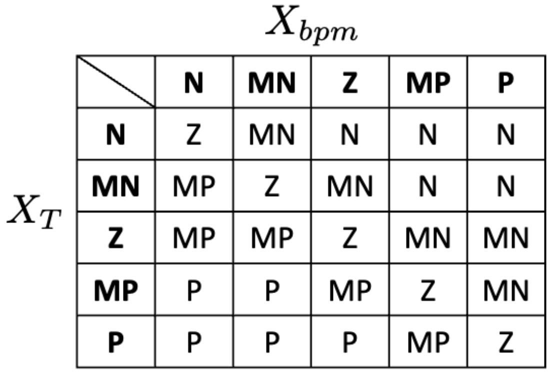


Figure 2.41: Membership functions for C1 (a) and C2 (b).

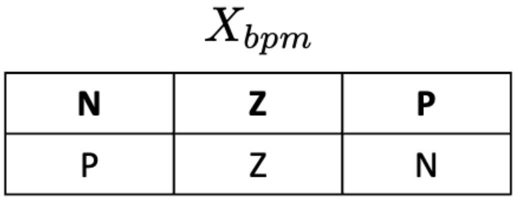
For C1 evaluation 20 different episodes with different initial beam positions were considered, while the target X_T was randomly selected in the range $[-1.5, 1.5]$ mm. Both X_{bpm} and Y have been normalized in the simulations in the range [-1, 1]. For the sake of space, the reported results refer just to a particular target position, but the performances are very similar for all the different target positions tested. For C2 evaluation, due to the limitation in time for the availability of IBM quantum devices, just 10 different initial beam positions were considered. In particular, these experiments have been carried out on the IBMQ Montreal device. In each episode and for both C1 and C2, the controller deflects the beam until a threshold reward value of -0.1 is reached. The results obtained for both C1 and C2 are reported in Fig. 2.43. In detail, the upper plots represent the number of controller actions (steps) required to achieve the reward threshold. The second plot reports the value of the reward function $\mathcal{R}$ at the beginning of the episode (red line) and at the end of it



X_{bpm}

X_T

	N	**MN**	**Z**	**MP**	**P**
N	Z	MN	N	N	N
MN	MP	Z	MN	N	N
Z	MP	MP	Z	MN	MN
MP	P	P	MP	Z	MN
P	P	P	P	MP	Z



X_{bpm}

N	**Z**	**P**
P	Z	N

Figure 2.42: Rule set for C1 (a) and C2 (b). The conjunction of the elements of the first row and column represents the antecedent part of a fuzzy rule having as consequent the corresponding matrix element. For instance, the first rule in C1 corresponds to the sentence *If X_{bpm} is Negative and X_T is Negative then the correction angle is Zero.*

(green line). The two boxes on the bottom of the figure, represent the dipole angle in mrad (Y) and the beam position at the BPM in mm respectively. The dashed lines represent the ideal output of the system, while the red and green lines show the initial and final values, respectively. For configuration C2, the ideal output for both these plots is 0, where the target was set. It can be seen that QFIE is able to control the environment in all the episodes and for both configurations. The absence of noise during the simulation for C1 enables the control of the beam with two steps at worst. On the other hand, the noise in computing the QFIE action that occurs in C2 and the lower granularity of the input fuzzy partition, make more steps required for achieving the optimal solution. Overall, these results prove the suitability of QFIE in controlling this kind of environment, both by means of simulations of the quantum circuits and by means of execution on real quantum hardware.

Electron Beam Line Control

This section reports the experiments carried out by using QFIE to control the AWAKE beam line. The Advanced Wakefield Experiment (AWAKE) at CERN uses high-intensity 400 GeV proton bunches from the Super Proton Synchrotron (SPS)

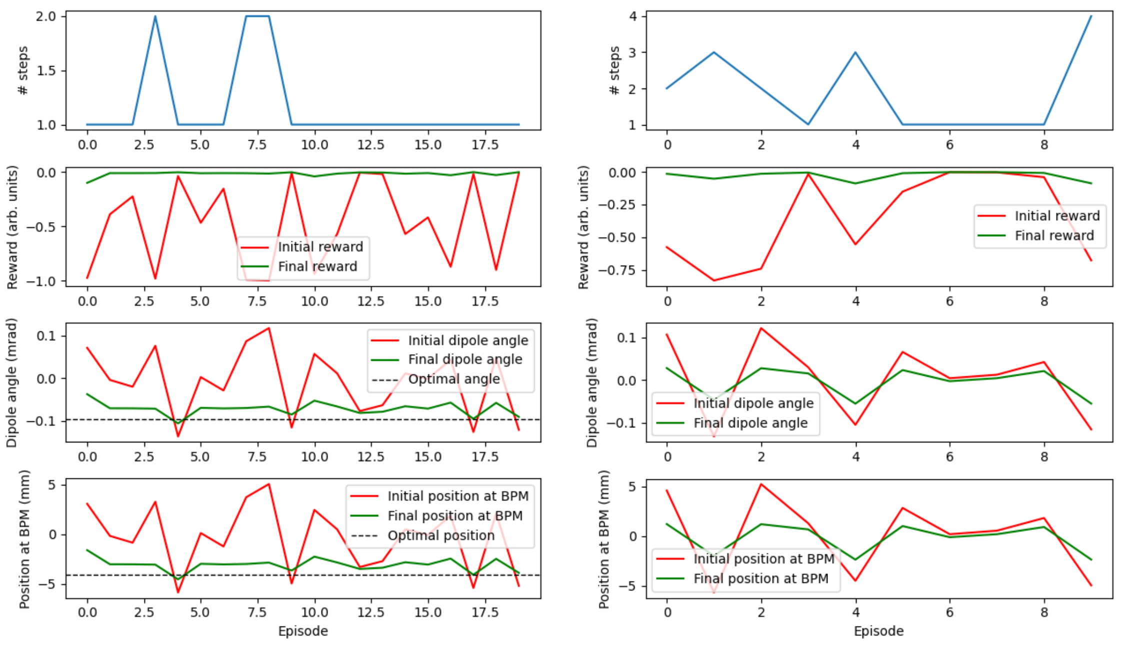


Figure 2.43: Experimental results for the target steering environment control by computing QFIE quantum circuits via noiseless simulations C1 (a) and via execution on IBMQ Montreal quantum processor C2 (b).

as a plasma wakefield driver. Electron bunches are simultaneously steered into the plasma cell to be accelerated by the proton-induced wakefields. Electron energies up to 2 GeV have been demonstrated over a plasma cell of 10 m length corresponding to an electric field gradient of 200 MV/m [20]. The ultimate goal for AWAKE is to reach a field gradient of 1 GV/m. These numbers are to be compared to conventional accelerating structures using radio-frequency (rf) cavities in the X-band regime, which are currently limited to accelerating field gradients of about 150 MV/m [22]. The AWAKE electron source and beam line are particularly interesting for algorithm preparation and testing due to the high repetition rate and insignificant damage potential in case of losing the beam at accelerator components. The AWAKE electrons are generated in a 5 MV photocathode rf gun, accelerated to 18 MeV, and then transported through a beam line of 12 m to the AWAKE plasma cell. The trajectory is controlled with 10 horizontal and 10 vertical steering dipoles according to the measurements of 10 BPMs (per plane), see Fig. 2.44. The BPM electronic read-out

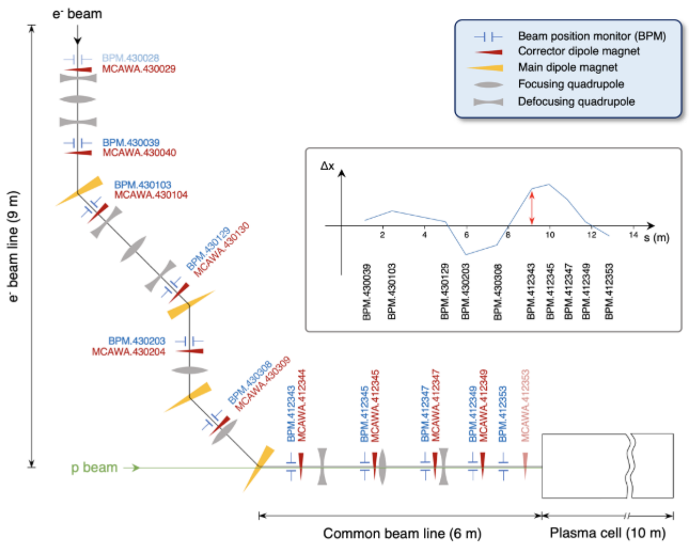


Figure 2.44: Representation of the CERN AWAKE beam line with 10 trajectory correctors and 10 beam position readings per plane along the line.

is at 10 Hz and acquisition through the CERN middleware is at 1 Hz.

The goal was to develop a QFIE-based controller able to correct the horizontal trajectory with similar accuracy as the response matrix-based singular value decomposition (SVD) algorithm that has been traditionally used [55].

The input state of the controller is formalized as a 10-dimensional vector of horizontal beam position measured with respect to the reference trajectory. Accordingly, the controller action is a 10-dimensional vector of corrector dipole magnet kick angles within a range of $\pm 300\,\mu$rad. To evaluate the performance of the controller, a reward function is used that consists of the negative root-mean-squared (rms) of the measured beam trajectory with respect to the reference at all the BPMs.

Developing a single QFIE controlling simultaneously all the corrector dipole magnets along the AWAKE trajectory would reflect in a quantum circuit too big for being classically simulated or executed on a current NISQ device. Therefore to solve the control problem an approach based on the D-NISQ reference model proposed in [5] has been exploited: the original 10-dimensional problem was divided into ten 1-dimensional control problems, where each corrector dipole magnet K_i with $i \in [1, 10]$

is controlled by a QFIE, $QFIE_i$ with $i \in [1, 10]$. Each $QFIE_i \quad \forall i \in [2, 10]$ acts considering two input variables x_i and dk_i, where the former refers to the distance from the ideal position of the beam registered by the corresponding BPM_i, while the latter refers to the sum of the deviation carried out by the magnets that are placed previously to the i-th magnet on the AWAKE beam line. Formally, denoting with $\hat{y}_i$ the corrector dipole magnet kick angles computed by $QFIE_i$, the dk_m input variable for $QFIE_m$ is defined as follows:

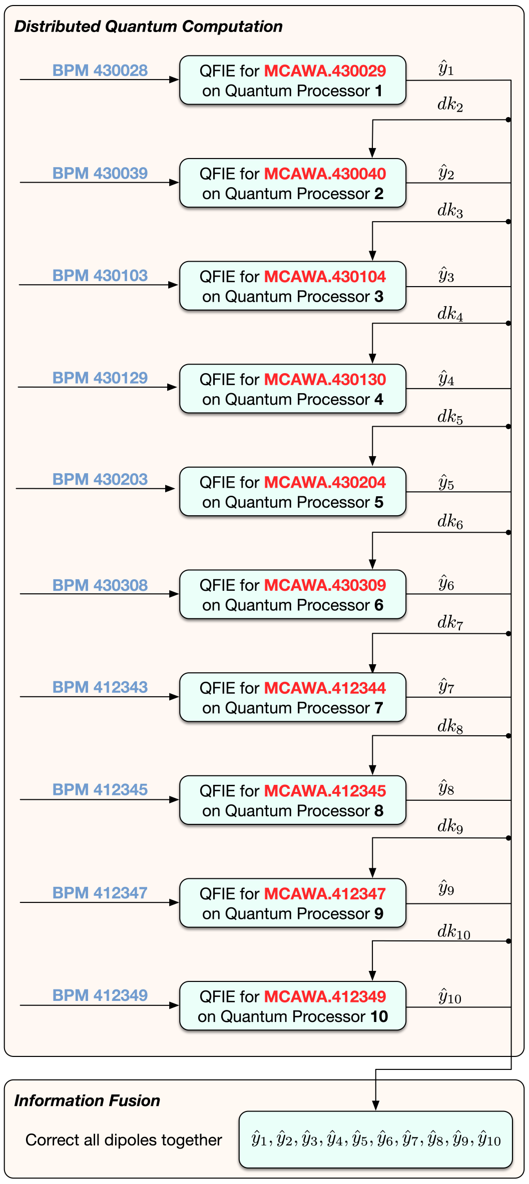
$$dk_m = \sum_{i=1}^{m-1} \hat{y}_i. \tag{2.94}$$

The action of $QFIE_1$ depends just on the position of the particle beam at the first beam position monitor along the trajectory. In detail, dk_i is defined in an interval [-2,2] $\forall i \in [2, 10]$; x_i is defined in an interval [-1,1] $\forall i \in [1, 10]$; the output corrector dipole magnet kick angles y_i are defined in the normalized interval [-1,1] $\forall i \in [1, 10]$.

Fig. 2.45 summarizes graphically how the whole FRBS has been distributed according to the D-NISQ architecture. However, due to the quantum noise affecting current quantum processors, the quantum circuits implementing the different $QFIE_i$ $\forall i \in [1, 10]$ were ideally simulated.

The fuzzy partitions used for the variables of each $QFIE_i$ are the same. In particular, Fig. 2.46 shows them for $QFIE_{10}$. Moreover, Fig. 2.47 shows the fuzzy rule base for $QFIE_i \quad \forall i \in [1, 10]$.

To minimize the number of interactions of the whole controller with the environment a bias factor b has been multiplied by the ten $QFIE$s output. Formally, denoting with $\hat{y}_i$ the output computed by $QFIE_i$, the final corrector dipole magnet



Distributed Quantum Computation
BPM 430028
QFIE for MCAWA.430029 on Quantum Processor 1
BPM 430039
QFIE for MCAWA.430040 on Quantum Processor 2
BPM 430103
QFIE for MCAWA.430104 on Quantum Processor 3
BPM 430129
QFIE for MCAWA.430130 on Quantum Processor 4
BPM 430203
QFIE for MCAWA.430204 on Quantum Processor 5
BPM 430308
QFIE for MCAWA.430309 on Quantum Processor 6
BPM 412343
QFIE for MCAWA.412344 on Quantum Processor 7
BPM 412345
QFIE for MCAWA.412345 on Quantum Processor 8
BPM 412347
QFIE for MCAWA.412347 on Quantum Processor 9
BPM 412349
QFIE for MCAWA.412349 on Quantum Processor 10
$\hat{y}_1$ dk_2 $\hat{y}_2$ dk_3 $\hat{y}_3$ dk_4 $\hat{y}_4$ dk_5 $\hat{y}_5$ dk_6 $\hat{y}_6$ dk_7 $\hat{y}_7$ dk_8 $\hat{y}_8$ dk_9 $\hat{y}_9$ dk_{10} $\hat{y}_{10}$
Information Fusion
Correct all dipoles together
$\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4, \hat{y}_5, \hat{y}_6, \hat{y}_7, \hat{y}_8, \hat{y}_9, \hat{y}_{10}$

Figure 2.45: D-NISQ computation of the quantum FRBS for the AWAKE beam line.

kick angle y_i used to modify the environment state is obtained as follows:

$$y_i = \begin{cases} \hat{y}_i \cdot b & \text{if } \hat{y}_i \cdot b \in [-1, 1] \\ 1 & \text{if } \hat{y}_i \cdot b > 1 \\ -1 & \text{if } \hat{y}_i \cdot b < -1 \end{cases} \tag{2.95}$$

In our experiments, b has been set to 10.

Fig. 2.48 reports the experimental results obtained by simulating the AWAKE

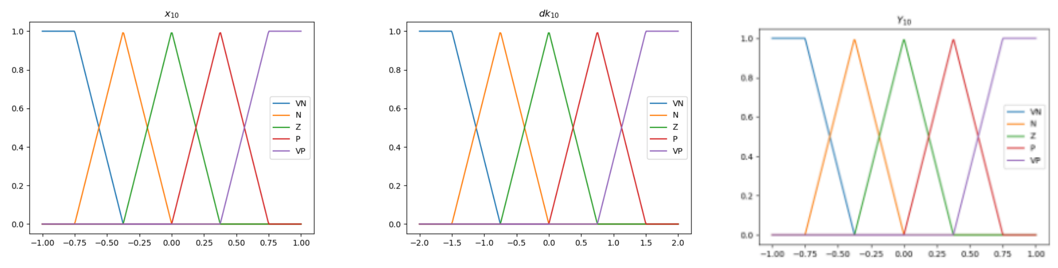
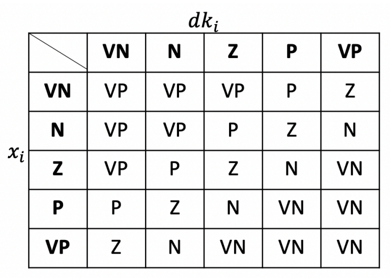
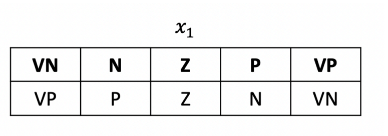


Figure 2.46: Fuzzy Partitions for $QFIE_{10}$ Input variables x_{10}, dk_{10} and the Output variable Y_{10}.



dk_i / x_i

	VN	**N**	**Z**	**P**	**VP**
VN	VP	VP	VP	P	Z
N	VP	VP	P	Z	N
Z	VP	P	Z	N	VN
P	P	Z	N	VN	VN
VP	Z	N	VN	VN	VN



x_1

VN	**N**	**Z**	**P**	**VP**
VP	P	Z	N	VN

Figure 2.47: Rule set for $QFIE_i$ $\forall i \in [2, 10]$ (Left) and $QFIE_1$ (Right). The conjunction of the elements of the first row and column represents the antecedent part of a fuzzy rule having as consequent the corresponding matrix element. For instance, the first rule in $QFIE_i$ corresponds to the sentence *If* dk_i *is Very Negative and* x_i *is Very Negative then the correction angle is Very Positive.*

environment controlled by QFIEs. The simulations stop when the reward objective reaches an rms value of 2 mm. As shown by the plots, considering 50 different episodes, where the initial condition of the beam is far away from the threshold rms value, the quantum fuzzy control system is able to align the particle beam to the ideal trajectory in 100% of the episodes. Moreover, in all the episodes the desired trajectory is obtained at the worst by means of two interactions of the controllers with the environment, proving the sample efficiency of the proposed quantum controller also in this case study.

The QFIE-based FRBS was also evaluated on the real AWAKE environment to test sim-to-real transfer. The same configuration of QFIE considered for the simulated environment has been exploited. In detail, the distributed approach used to compute the 10-D action enables a fast simulation time of the circuits, which made it possible to test the proposed approach in real time on the actual AWAKE

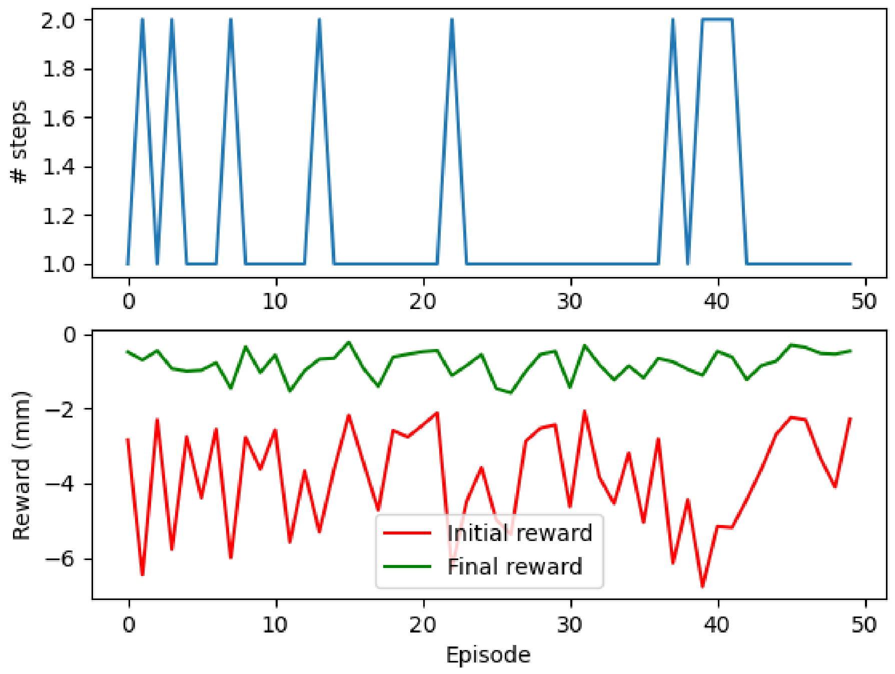
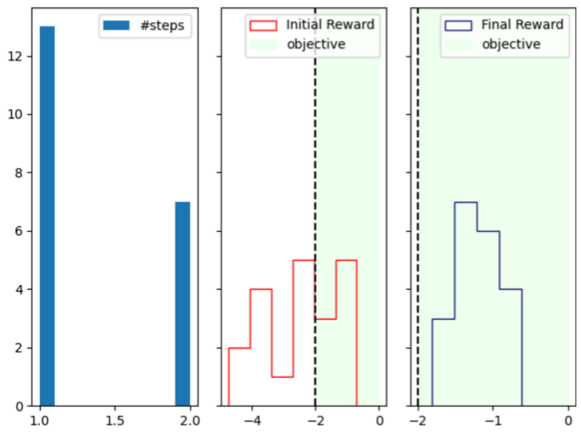
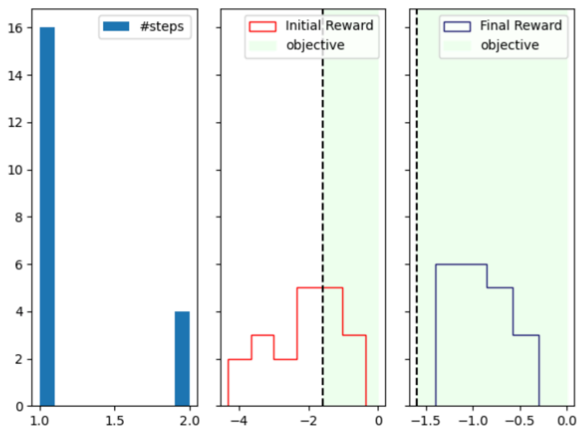


Figure 2.48: Experimental results on the simulated AWAKE environment.

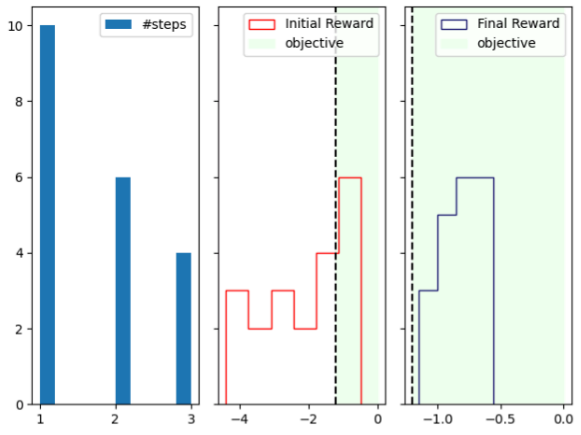
facility. Fig. 2.49 shows the histograms reporting the obtained results: QFIE has been tested by considering four different levels of reward objective threshold value, -2 mm (Fig. 2.49 (a)), -1.6 mm (Fig. 2.49 (b)), -1.2 mm (Fig. 2.49 (c)) and -0.8 mm (Fig. 2.49 (d)). A lower absolute reward value reflects in more precise control of the particle beam. For each threshold value, 20 independent episodes were collected. Histograms in Fig. 2.49 report respectively the distribution of the number of interactions environment-control required to reach the desired rms value, the distribution of the 20 initial state reward values, and the distribution of the 20 final state reward values. As highlighted by the plots, the QFIE-based controller is able to solve the control problem also in the real AWAKE environment. Indeed the objective reward value is reached in 100% of the episodes considered. The number of steps required to achieve such an impressive result is in line with the simulated environment, except for an outlier in the scenario with an rms threshold equal to 0.8 mm.



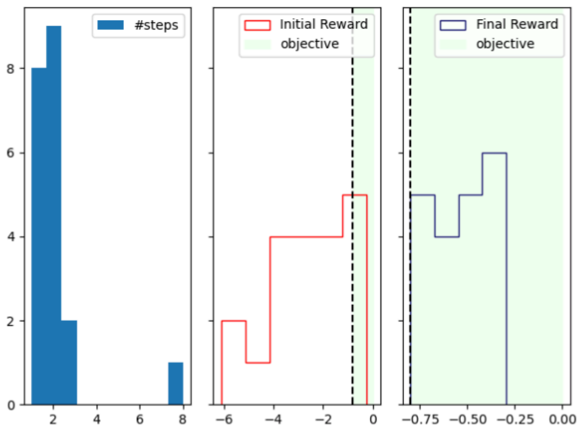
(a) rms = 2 mm



(b) rms = 1.6 mm



(c) rms = 1.2 mm



(d) rms = 0.8 mm

Figure 2.49: Online experiments on the real AWAKE beam line. Each plot refers to a different value of rms threshold.

To summarize, these sections have shown how a QFIE-based FRBS is able to control real-world environments, such as those related to particle physics accelerators at CERN facilities. The main result obtained from this research is twofold: on the one hand, it has been shown that QFIE is able to control these complex environments both by means of simulations and real executions of quantum circuits implementing the algorithm on a quantum computer; on the other hand, it has been proved that quantum FRBSs could be a valid tool for the real-time control of particle accelerators for the physics experiments at CERN, as shown by the test carried out on the real AWAKE beam line. In detail, the research was carried out on two successive scenarios, two existing CERN beam lines representing control problems of different degrees of complexity. In the first scenario, the QFIE controller implemented aims to deflect the beam via the magnetic dipole in an environment based on the TT24-T4 transfer line at CERN: in this simpler context, the whole algorithm was executed on a real IBM Quantum computer, proving the feasibility of QFIE in controlling this kind of environment on real quantum devices. The second scenario consists of the AWAKE use case, where the 10-dimensional environment is much more complex and current NISQ devices are not ready to handle the resulting QFIE circuits. However, in this case, the simulated quantum circuits were tested on real data as an online controller of the beam line. This result proves for the very first time the capability of a FRBS to control a real particle accelerator.

2.2.3.VI A Comparison of Quantum Computer Architectures in Running Fuzzy Inference Engines

In the previous section, the first run of a QFIE on a real quantum computer has been presented. In detail, it covered the application of a quantum-based FRBS to control an environment based on the TT24-T4 transfer line at CERN. The quantum circuits implementing the QFIE algorithm were executed on a superconductive quantum device from the IBM Quantum facilities. However, as pointed out in Section 1.1.1.II different technologies are currently under study to implement the current and the future generation of quantum computers. This section aims to analyze which technology fits better to be used as a backend for running the QFIE algorithm. Among the technologies introduced in Section 1.1.1.II superconductors and trapped ions are the ones that currently appear to be the most advanced and 'mature' and therefore the theoretical/experimental comparison carried out in this section is limited to them. The superconductive and the ion-trap-based quantum devices were compared in several works [42, 113, 89], that from a theoretical and experimental point of view highlight both the pros and cons of each. From these works, it is possible to point out the differences between these hardware technologies, highlighting which can mainly affect the performance of QFIE.

- **Physics of the Systems**: Ion trap quantum computers store qubits in the internal states of individual electrically charged atoms (ions). The ions are laser-cooled and confined with electric fields from nearby electrodes, forming a 1D crystal in free space and in a vacuum environment [182]. Quantum gates are implemented by poking the ions with focused laser beams that affect a qubit state-dependent force that modulates the Coulomb interaction between the ions and effectively wires together any pair of qubits. On the other hand, superconducting quantum computer systems store qubits in Josephson junctions typically arrayed on a 2D lattice and cooled to very low temperatures

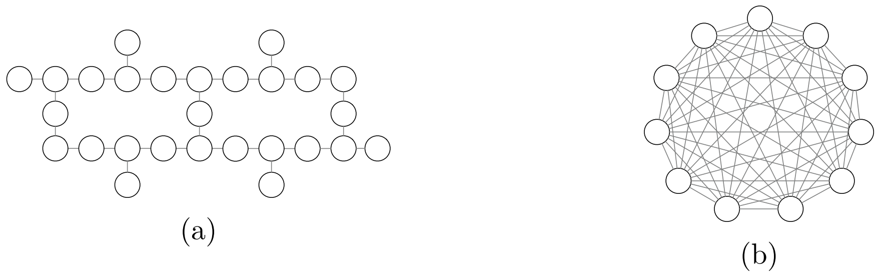


Figure 2.50: Copuling Map of the quantum computers used in the experiments. Nodes represent qubits, edges represent the available two-qubit gates between qubits. (a) *ibmq mumbai* (27 qubits); (b) *ionq harmony* (11 qubits).

[98]. Here, quantum gates are executed by running electrical currents through the device in a way that either changes the state of a single qubit or exploits the nonlinear interaction between adjoining qubits.

- **Qubits Connectivity**: Both ion-trap and superconducting quantum computers have their own topology and layout. As shown in sec. 3.1.0.II, this is expressed by means of a graph that indicates the connections between qubits. If two qubits are connected, then a two-qubit quantum gate can be executed between them. Otherwise, SWAP gates have to be inserted in the original quantum circuit to preserve the logical structure of the process. These operations strongly increase the depth of the original quantum circuit degrading its performance. Superconducting systems generally have fewer connections because of the two-dimensional wiring between nearest-neighbor Josephson junctions. On the other hand, ion-trap quantum computers can be fully connected with direct gates available between any pair of qubits (see Fig. 2.50).

- **Native Gates**: Another important difference between ion-trap and superconducting quantum devices is surely the basis gates set that can be physically executed on the machine. Considering the two devices used to perform our tests, namely *ibmq mumbai* and *ionq harmony*, it can be seen that for the former the basis gate set is composed of $[CX, ID, RZ, SX, X]$, while for the latter the set of native gates is $[GPI, GPI2, MS]$. Focusing our attention on multi-qubit gates, which are the gates with the highest error rate, both the universal set of gates have one entangling gate: for superconducting devices this is the controlled not (CX) gate [130], for ion trap devices it is the MølmerSørensen gate (MS) [123]. The MS gate is native to ion-trap systems and it has an important difference with respect to the CX gate used by superconducting systems: MS can potentially be applied to multiple ions to entangle multiple qubits simultaneously, while CX acts just between two qubits. However, the current implementation of IonQ Hardware enables the execution of MS just between two qubits. In this case, the MS gate is equivalent to an $XX(\theta)$ interaction.

- **Coherence and Clock Speed**: A factor to take into account when comparing quantum devices is the coherence time of qubits and the time required to execute quantum gates. It is important to highlight that such a comparison is strictly related to the current status of the technology development and could be completely overturned in the future. Today, while a superconducting computer has gate times in the nanosecond range, ion traps reach "only" microseconds. The faster gate times are, the higher the number of circuits that can be executed according to the algorithm and, as of today, it seems to favor the superconducting quantum computer. In contrast, ion trap computers can shine with coherence times, which express the stability of a quantum state and it is of the order of about a few seconds. Superconducting quantum computers currently have a coherence time of microseconds. This comparison can be interpreted as an indication that ion trap-based architectures can better tolerate more complex algorithms, i.e. longer circuits, at the current stage of construction.

- **Scalability**: A crucial point in the development of future quantum processors is how these quantum technologies can scale to large systems. One of the biggest challenges is the management of the control complexity in larger

> systems and potential cross-talk from overlapping qubit interactions or control buses. In most superconducting designs, there are many current-carrying wires necessary for controlling and biasing the individual qubits, and this may be difficult to route through a large superconducting chip [133]. It will likely become a great challenge to manage the dilution refrigerator heat budget with such circuitry. Alternative modular superconducting architectures improve connectivity by integrating qubits with microwave cavity modes, at the expense of significant added volume per qubit [47]. Ion trap designs will hinge upon the stable and accurate delivery of laser beams (or near-field microwave sources) to address each qubit individually in a vacuum chamber. The fully connected structure of the ion trap architecture may not scale to arbitrarily large numbers of qubits, owing to the spectral overlap of collective normal modes of motion. However, full connectivity between 20-100 trapped ion qubits seems possible [64] and a modular approach for scaling to much larger systems with high connectivity and distance-independent operations seems promising [124].

From the above analysis, two advantages of using an ion-trap backend for QFIE can be highlighted. Firstly, the oracle used in QFIE uses MCXs for its implementation on a digital device. Considering this, the ideal backend should have high connectivity between the inputs and the output quantum registers used by QFIE. Therefore, the one-to-all connectivity of ion-trap devices can result in shorter QFIE quantum circuits than the equivalent circuits mapped on superconducting devices. Secondly, MS gates executed on ion-trap could involve simultaneously more qubits, and therefore the decomposition of MCX gates in a set of single and two-qubit gates could be avoided, decreasing the depth of the quantum circuit for QFIE dramatically. However, at the time of writing such an implementation of MS gate is not available and the two-qubit gates decomposition also remains on ion-trap devices.

To experimentally analyze the differences in running the QFIE on ion-trap and superconducting devices, the control problem related to a one-dimensional beam target steering environment based on the beam optics of the TT24-T4 transfer line at CERN has been used as a testbed. The details of the environment have already been discussed in sec. 2.2.3.V. In particular, in the following experiments, the C2 configuration of the FRBS was exploited (see the fuzzy partitions in Fig. 2.41 (b) and the fuzzy rules in Fig. 2.42 (b)). To perform the comparison of the different backends, 16 independent random initial positions of the beam were considered. For each episode, the QFIE-based controller was run on three different backends: an ideal quantum circuit simulator (*qasm simulator*), the ion-trap *ionq harmony* device, and the superconducting *ibmq mumbai* processor. In each episode, each controller rotates the beam until a threshold reward value (2.93) of -0.1 is reached. The number of rotations carried out by each controller in a given episode will be denoted as a number of steps. If in 10 steps the threshold isn't reached, then the episode automatically ends.

Figure 2.51 shows the experimental results obtained simulating the TT24-T4 environment controlled by the QFIE executed on the different backends for all the episodes tested. All the backends used to compute QFIE are useful to have a controller able to achieve the desired reward value. However, a clear behavior arises from these plots: in noiseless simulation, QFIE is able to control the particle beam efficiently using a maximum of one step for pointing the beam to the target position. This trend is also confirmed for QFIE computed on *ionq harmony* for 75% of the tested episodes while in the remaining 25% of the cases, QFIE takes a maximum of two steps to reach an acceptable reward value. In contrast, QFIE computed on *ibmq mumbai* takes one step to reach the threshold in only 37.5% of the tested cases, and in the remaining, it sometimes takes more than two steps (as for episodes 10 and 15 - this trend confirms the results presented in Fig. 2.43 (b). The main reason behind the less accurate computation per step of QFIE on the *ibmq mumbai* device is related to the transpiling process: the original quantum circuit implementing QFIE (see Fig.

2.52) is transpiled differently on the two backends and the one-to-all connectivity of *ionq harmony* results in a shallower quantum circuit than the equivalent circuit run on the IBMQ device. In the former case, the final quantum circuit depth is 89, while in the latter case, it is 111. Moreover, the quantum circuit executed on the *ionq harmony* has a total of 19 XX entangling gates, while the circuit executed on *ibmq mumbai* has a total of 47 CX gates, many of which are related to the decomposition of SWAP gates required to respect the processor topology. Considering the high level of noise of these two-qubit gates, the difference between the two algorithms can be related to that.

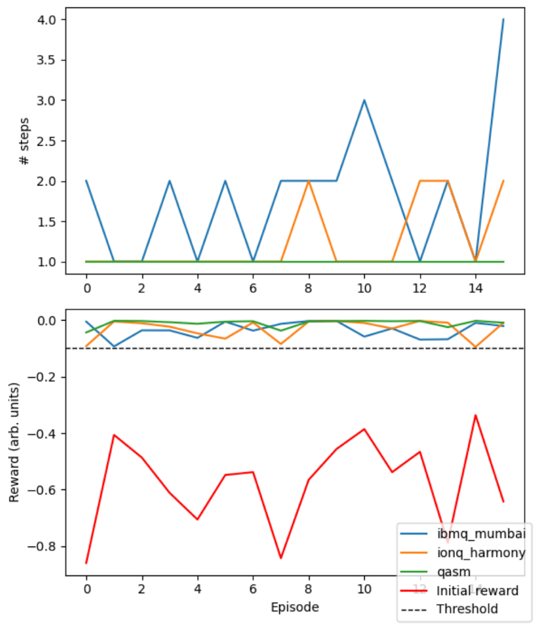


Figure 2.51: Number of Steps and Reward function for each episode tested.

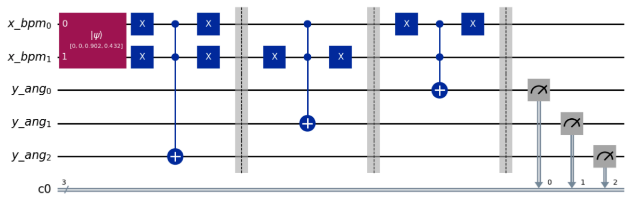


Figure 2.52: QFIE quantum circuit.

To conclude, this study has presented the first comparative study to evaluate the performance of superconducting and trap-ion quantum computers in running fuzzy inference engines, paving the way for a practical application of fuzzy systems in the quantum world. Results showed that, at this pioneering stage in the development of quantum technologies, quantum ion trap computers proved to be more adequate for doing fuzzy computation by limiting the noise levels typical of quantum computation.

2.2.3.VII Distributing Fuzzy Inference Engines on Quantum Computers

In the previous sections, it has been shown that the QFIE algorithm can be run on actual quantum computers. However, independently from the technology, these NISQ devices are characterized by a limited number of qubits and are unable to implement error correction codes to efficiently minimize the negative impact of quantum noise on computation. As a result, there is a strong need to think of technological tricks through which to use NISQ devices to run QFIE by minimizing the effects of quantum noise and making this innovative and efficient inference engine usable in real application environments where the number of variables and the number of rules is higher than the examples shown above. The research described hereafter bridges this gap by introducing a distributed version of QFIE (D-QFIE) based on the *Distributed Noisy Intermediate-Scale Quantum (D-NISQ)* approach, a reference model for distributed quantum computation [5]. This model consists of a hybrid and hierarchical architecture in which classical and quantum processors interact to decompose a problem into a collection of sub-problems, solve each sub-problem using an appropriate quantum algorithm, and fuse the output quantum information to compute the final solution to the problem posed. From this point of view, D-NISQ enables the implementation of a distributed version of QFIE by decomposing the fuzzy rule base into sets of rules characterized by the same output, where each subset of rules is evaluated on a different quantum processor; then the results from the different quantum processors are aggregated and appropriately defuzzified by a classical processor. This computational distribution approach simplifies and reduces the size of the quantum circuits involved in the execution of the fuzzy inference engine. In particular, these circuits are characterized by a low depth and a small number of multi-qubit quantum gates, which are the ones that generate the most noise in the computation. The proposed quantum/fuzzy framework was tested also experimentally and a significant improvement in the accuracy of the calculations performed could be appreciated.

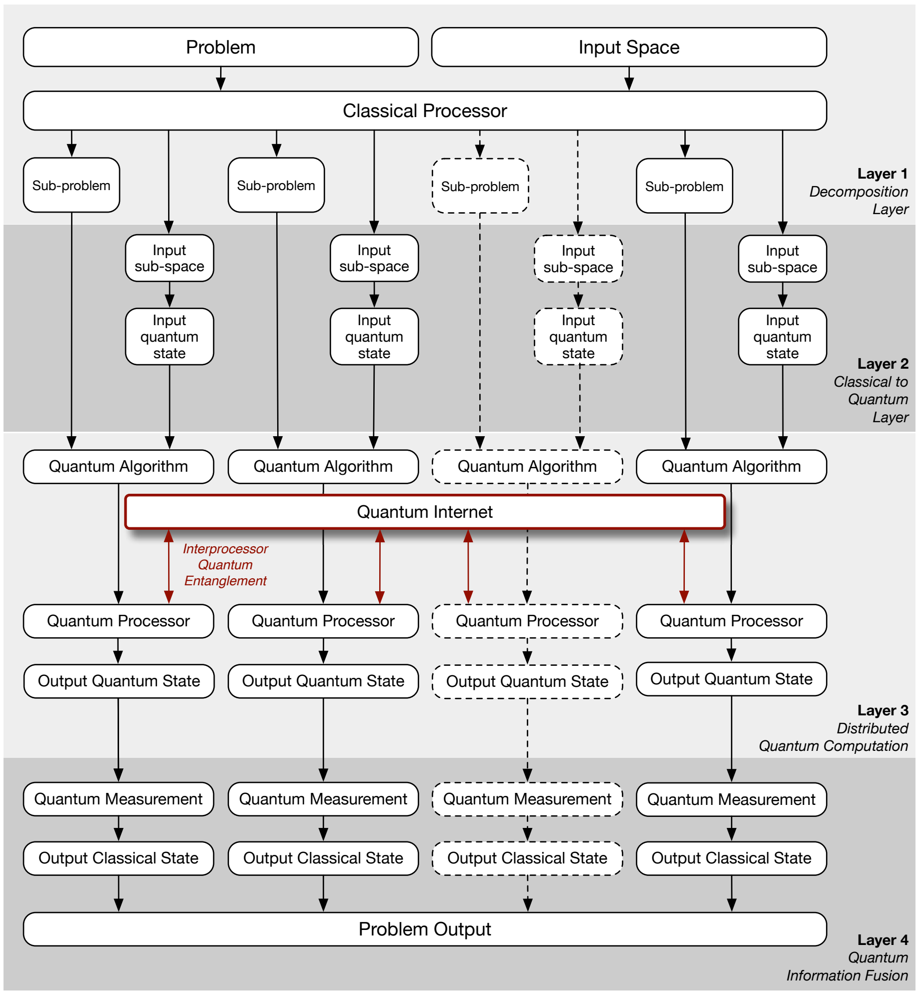


Figure 2.53: Reference model of D-NISQ architecture introduced in [5].

In order to understand how QFIE can be distributed over more quantum processors according to the D-NISQ architecture, it is necessary to better describe this reference model. In detail, this architecture is structured in four layers, as shown in Fig. 2.53:

- **Decomposition Layer**: Let P be a generic problem that needs to be solved by means of a quantum algorithm QA_P. Let I_P be the input space corresponding to P, e.g. a space of possible solutions to the problem. In this first layer, the problem is decomposed in a given number n of sub-problems for which a reduced version of QA_P can be implemented on a separate quantum processor. In particular, n sub-problem instances P_j are created and I_P is divided in n

sub-spaces I_P^j so that:

$$I_P = \bigcup_{j=0}^{n-1} I_P^j \tag{2.96}$$

- **Classical to Quantum Layer**: In the second layer, each partition I_P^j of the classical input space I_P must be expressed as a set of quantum states that can serve as inputs to a quantum algorithm QA_{P_j}. This can be achieved by defining n maps $\Phi_j : I_P^j \to \mathcal{H}_j$, that associate each classical input in I_P^j with a quantum state $|\psi_j\rangle \in \mathcal{H}_j$, where $\mathcal{H}_j$ is the Hilbert space corresponding to the j-th sub-problem. This encoding allows the feeding of each sub-problem to a quantum processor, along with its corresponding input sub-space.

- **Distributed Quantum Computation Layer**: A set of n quantum algorithms QA_{P_j} is run on n independent quantum devices. Each algorithm returns an output quantum state, on which a measurement operation is performed.

- **Quantum Information Fusion Layer**: Thanks to the measurement operation, a probability distribution can be extracted at the end of each quantum algorithm QA_j, whose most likely value encodes the classical solution to the j-th sub-problem. In this final layer, the n classical solutions are pieced back together by means of a fusion operator Ξ (whose action depends on the problem at hand and the strategy adopted to express it in a distributed form), so that the complete classical solution to the original problem P can be estimated.

Let's now retrace the four steps of the D-NISQ architecture in the particular case where such a model is applied to a control problem P to be addressed by means of QFIE.

1. *Decomposition Layer*: The original control problem P is broken down in $n = m_Y$ subproblems, where m_Y is the number of linguistic terms associated with

the output fuzzy variable Y. By doing so, each sub-problem will be associated with one single linguistic term among the ones describing the output Y, as will be clarified in the following. The input space I_P is made up of the knowledge base of the original problem, i.e. a database of fuzzy input/output variables along with their respective fuzzy sets and a rule base ($\mathcal{RS}$) containing a given number of if-then statements. The first layer deals with instantiating n equal sub-problems P_j, each one having its own input sub-space I_P^j. In detail, each I_P^j contains:

- the same database of fuzzy variables and fuzzy sets of the original problem, except that only one linguistic term T_j^Y is allowed for Y;
- a subset $\mathcal{RS}_j$ of the original rule base including only the rules whose consequent proposition contains T_j^Y.

Note that such a partition of the original input space is coherent with (2.96). In terms of Boolean oracle formulation of $\mathcal{RS}_j$, it can be modeled as described in Section 2.2.3.I by considering the same antecedent set $\mathcal{A}_\mathcal{S}$ and a different consequent set $\mathcal{C}_\mathcal{S}^j = \{1, 0\}_j$ for each sub-problem. By doing so, n Boolean oracles f_j can be considered, in such a way that if $a \in \mathcal{A}_\mathcal{S}$ is an antecedent present for $\mathcal{RS}_j$ then $f_j(a) = 1_j$, otherwise $f_j(a) = 0_j$.

2. *Classical to Quantum Layer:* In this layer the input and the output quantum registers are initialized. For the former, the amplitude encoding procedure described in (2.75) is carried out for each sub-problem. For the latter, considering that $|\mathcal{C}_\mathcal{S}^j| = 2 \ \forall j \in [1, m_Y]$, then in each sub-problem P_j the output quantum register will be composed of only 1 qubit initialized in the state $|0_j\rangle$.

3. *Distributed Quantum Computation Layer:* A quantum oracle $\mathcal{O}_{f_j}$ is applied on each quantum state $|\psi_j, 0_j\rangle$ in order to reproduce the transformation reported in (2.85).

4. *Quantum Information Fusion Layer:* The final quantum states are measured several times to retrieve the distribution of probability encoded in each output register. In detail, for each sub-problem P_j, the probability of measuring the output qubit in the state $|1_j\rangle$ will be equivalent to a specific probability P_{c_j} reported in (2.86). Therefore, the set $\{P_{c_j}\}_{i=j}^{m_Y}$ is obtained and finally the fusion operator Ξ can return the crisp output value by performing the implication, aggregation, and defuzzification steps as for the not-distributed QFIE algorithm.

The quantum circuits of D-QFIE have two important advantages over the quantum circuits of QFIE. Firstly, the depth of the latter is given approximately by the sum of the depths of the different oracles used in D-QFIE, which are for this reason much shallower and thus more resistant to quantum noise than the not-distributed counterpart. Secondly, each quantum circuit in D-QFIE uses an output register composed of just one qubit against the m_Y qubits used in the standard QFIE algorithm. This last feature also has the additional side effect of reducing the number of shots N_S used to sample the probabilities $\{P_{c_j}\}_{j=1}^{m_Y}$ from which the implication operation is performed. Indeed, the number of shots required to reconstruct with some fidelity F these probabilities is given by (2.92). Therefore, in a noisy not-distributed case $N_{c_j} = 2^{m_Y}$, while for each sub-circuit of D-QFIE $N_{c_j} = 2$. From the above analysis, it is clear that as more rules and output linguistic terms are in an FRBS more D-QFIE is profitable compared to standard QFIE.

To evaluate the suitability of D-QFIE in being executable on noisy quantum hardware, two FRBSs equipped respectively with QFIE and D-QFIE have been implemented for the control of a MISO environmental lighting system. Two input variables are considered: the environmental light l (described by means of three linguistic terms, i.e. *dark*, *medium* and *light*) and its changing rate r (*negative small*, *zero* and *positive small*, in the following denoted by NS, Z, and PS respectively).

The output variable is the dimmer control d of a bulb light (*very small* (VS), *small* (S), *big* (B) and *very big* (VB))[12]. Overall, considering the FRBS setup, the D-QFIE algorithm is distributed over $m_Y = 4$ quantum circuits. To perform the comparison between the FRBS equipped with QFIE and D-QFIE three control surfaces have been computed (see Fig. 2.54): the first one obtained by running QFIE on an ideal quantum simulator (i.e., a simulator that reproduces the behavior of a quantum computer with perfect gate implementations and protection from noise and decoherence); the second, by running QFIE on a noisy quantum simulator (i.e., a simulator that reproduces the imperfect behavior of a real NISQ device); the third, by computing D-QFIE using the same noisy simulator for each of the quantum devices required by the Distributed Quantum Computation layer. All the above quantum simulations have been run by means of *Qiskit* SDK. In particular, the *qasm simulator* backend has been used for the ideal simulations, while the noisy simulations reproduce the behavior of *ibmq guadalupe*, a 16-qubit IBM Quantum processor. Moreover, for each quantum simulation, the number of repeated executions (shots) of the quantum circuits has been set to 16000.

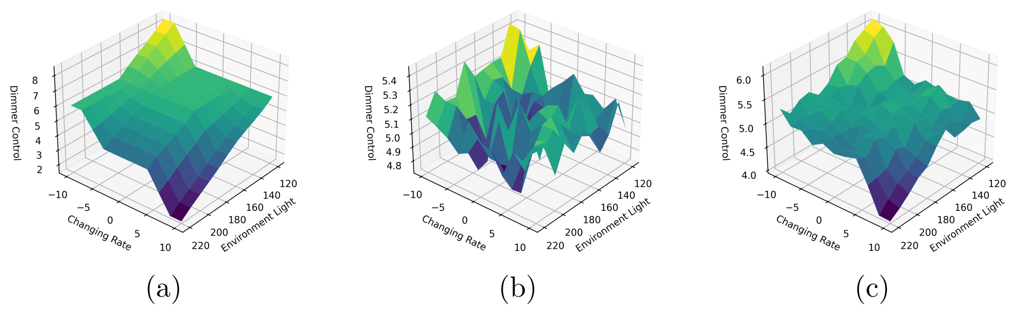


Figure 2.54: Control surfaces for the lighting control system obtained by running QFIE and D-QFIE on different backends. Surface (a) refers to ideal simulations; surface (b) refers to standard QFIE noisy simulations; and lastly, surface (c) refers to D-QFIE noisy simulations.

As graphically shown in Fig. 2.54, the control surface obtained by means of noisy simulations of quantum circuits in D-QFIE (Fig. 2.54c) looks more similar to the

[12]The fuzzy partitions defined for each variable and the rule base for the system can be found at: `https://qfie.readthedocs.io/en/latest/Lighting_Control.html`

Table 2.17: RMSD and HD Metrics.

Reference surface	Noisy surface	RMSD	HD
QFIE ideal simulator	QFIE	1.31	3.73
QFIE ideal simulator	D-QFIE	1.01	2.44

ideal one (Fig. 2.54a), if compared to the one returned by QFIE (Fig. 2.54b), due to the advantages of the distributed approach discussed above. Indeed, the compiled quantum circuit computed in QFIE results in a final depth of 586 with a number of 326 two-qubit gates. On the other hand, the four quantum circuits computed in D-QFIE related to the four different linguistic terms VS, S, B, and VB have respectively a final depth of 66, 131, 326, 66 and a number of two-qubit gates of 38, 74, 182, 38. To also quantitatively compare the different control surfaces two similarity metrics were considered. Say S and S' are two generic surfaces to compare. The first metric considered is the well-known root-mean-square deviation (RMSD), namely:

$$\mathrm{RMSD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{r}_i - \mathbf{r}'_i)^2} \tag{2.97}$$

where $\mathbf{r}_i$ ($\mathbf{r}'_i$) denotes the three-dimensional coordinates of the i-th point of S (S') and N is the total number of surface points. The second metric is the Hausdorff distance (HD), defined as:

$$\mathrm{HD} = \max\left\{\sup_{\mathbf{r}\in S} d(\mathbf{r}, S'), \sup_{\mathbf{r}'\in S'} d(\mathbf{r}', S)\right\} \tag{2.98}$$

where $d(\mathbf{r}, S')$ is the Euclidean distance between the surface S' and a given point belonging to S (or vice versa). Both (2.97) and (2.98) tend to zero in the limiting case of identical surfaces. Their values are reported in Table 2.17. The distributed approach allows us to improve the control surface by 22.9% with respect to the RMSD, and by 34.6% with respect to the HD.

The other thing to analyze is the additional effect related to the distributed approach of QFIE about a lower number of shots required to sample the output distribution probability with a certain level of fidelity. To experimentally analyze this aspect, the control surfaces of the FRBSs controlling the lighting systems were computed with different numbers of shots both for QFIE and D-QFIE with noiseless and noisy simulators. Let us denote with $S(Q, I, N_S)$ and with $S(Q, N, N_S)$ the control surfaces obtained with a FRBS equipped with QFIE computed with N_S shots and by using ideal and noisy simulations of the quantum circuit respectively. Moreover, let $S(DQ, I, N_S)$ and $S(DQ, N, N_S)$ be the corresponding surfaces obtained by using the FRBS equipped with D-QFIE. Our goal was to analyze how these surfaces were sensible to the parameter N_S. The numbers of shots N_S investigated are 16000, 8000, 4000, 1000, 500, 100, 50, 20 and 10. To assess this aspect, the following steps were performed: first, the surfaces $S(Q, I, 16000)$, $S(Q, N, 16000)$, $S(DQ, I, 16000)$, $S(DQ, N, 16000)$ were computed and considered as reference surfaces; then, the number of shots was gradually decreased in each configuration and it was calculated the RMSD between the new surfaces and the reference ones. The lower this value is as the number of shots decreases, the less sensitive the relative quantum inference engine is to the variable N_S. The results are shown in Fig. 2.55. It can be seen that, for ideal simulations, there are no appreciable advantages that make D-QFIE preferable to QFIE in terms of shot minimization. For noisy simulations, on the other hand, D-QFIE generally leads to smaller RMSD values with respect to QFIE at a fixed number of shots. For the FRBS implemented in our work, the difference between the ideal and the noisy case can be explained because according to (2.92), the fidelity with which the P_{c_j} are esteemed is related to the number of possible outcomes in the measurement operation N_{c_j}. In the noiseless simulations of QFIE $N_{c_j} = m_Y = 4$, because only the states in $\mathcal{C}_{\mathcal{S}} - \{c_0\}$ can be measured on the output register. On the other hand, when noise affects the computation any quantum state in the output register can be measured, and therefore $N_{c_j} = 2^{m_Y} = 2^4$. Contrary, for the quantum

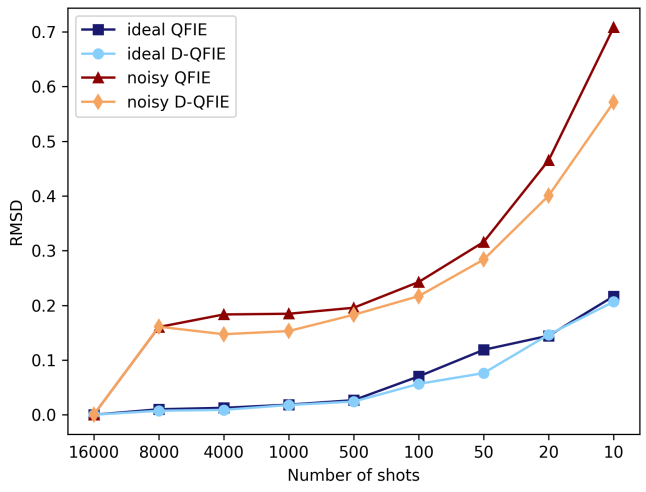


Figure 2.55: RMSD distance between a reference control surface (obtained with quantum simulations where the number of shots is set to 16000) and new control surfaces corresponding to decreasing numbers of shots. Please note that the number of shots is equally spaced for readability reasons.

circuits in D-QFIE both for noisy and noiseless simulations $N_{c_j} = 2$ because just one qubit in each circuit is measured.

To summarize, in this section the QFIE algorithm has been distributed according to the D-NISQ architecture, paving the way for fuzzy inference systems that can be computed on real quantum devices. Indeed, the D-QFIE approach has two great advantages over the not-distributed counterpart: firstly, it decreases quantum noise that disrupts computation by reducing the depth of the quantum circuits used to perform fuzzy inference; secondly, it reduces the number of shots to be carried out on actual quantum devices required to sample the probability distributions encoded in the output quantum state of each quantum circuit. These features are analyzed experimentally by using noisy and noiseless simulations of D-QFIE and QFIE employed in an FRBS controlling a lighting system.

2.2.3.VIII Summary

Section 2.2.3 has introduced the very first QFIE able to achieve an exponential speed-up with respect to classical Mamdani inference engines. This is due to properly formulating a fuzzy rule set as a boolean oracle and a specific amplitude encoding of fuzzified values in quantum states. QFIE was evaluated on different benchmark problems, such as the inverse pendulum environment and the control system for particle accelerators at CERN. In this latter scenario, QFIE has been executed both on quantum simulators and real quantum machines. Remarkably, by using the simulation of QFIE it has been possible to develop and test successfully the first QFIE-based FRBS for the real-time control of the actual AWAKE accelerator.

However, as already pointed out, the current status of NISQ devices limits the range of applicability for QFIE, and today it is not yet possible to use QFIE in a context too complicated for classical FRBS. This is mainly due to the high error rate of NISQ devices which particularly affects the quantum circuits implementing the Boolean oracle in QFIE. Still, this research proves that not all the quantum backends degrade QFIE performance in the same way: it has been shown that thanks to the full connectivity of trapped-ions devices the performance achieved by QFIE is higher than those achieved by running it on superconductive devices, where the low level of connectivity severely complicate the structure of the original QFIE circuit. Moreover, to begin to close the gap there is toward reliable execution of QFIE on real quantum machines, a distributed version of QFIE has been proposed. This enables the running of shallower quantum circuits whose output is classically aggregated so as to get the same final global output of a non-distributed QFIE circuit. Experimentally, it was seen that control surfaces obtained in this way also in noisy environments are much closer to the ideal ones than those obtained by not distributing QFIE.

In the future, QFIE will be tested on more complicated environments, and in particular, more compact circuit designs of the Boolean oracle will be studied. Finally, the encoding procedure introduced in this thesis will also be exploited to implement quantum TSK-FRBS.

Chapter 3

Computational Intelligence for Quantum Computing

As already discussed, this work aims to study two complementary paths: if in Chapter 2, quantum-enhanced computational intelligence algorithms have been proposed in the field of genetic optimization and fuzzy reasoning, this chapter aims to analyze the other pillar of this thesis, namely the exploitation of classical CI techniques to improve the current state of NISQ quantum devices. This research field can be divided into four macro-areas: CI for the control of quantum processors, CI for quantum compiling, CI for quantum error correction, and CI for quantum error mitigation. The first area involves all those approaches of CI used to optimize processes very close to the quantum hardware, e.g. pulse optimization [119, 145] or readout optimization [111]. The second area is related to the use of CI algorithms to adapt quantum circuits to hardware structures of quantum processors, in such a way that quantum algorithms can be effectively executed on a quantum backend [147, 102]. The third area is devoted to developing error correction schemes based on CI approaches [129, 134]. However, the restriction in the number of qubits available on NISQ devices doesn't enable the practical implementation of such approaches, which on the other hand will be crucial for the transition to the next generation of quantum processors. Considering this, to maximize the reliability of computation on current NISQ devices, techniques of error mitigation have been developed, i.e. software approaches to minimize the effect of quantum noise that affects the computation on NISQ devices, which don't require further hardware resources. Many of these techniques are based on CI techniques [18, 8].

In this thesis, however, the focus is on the compiling problem. Section 3.1 introduces an approach based on Neural Networks to map quantum circuits over physical qubits composing a superconductive quantum processor. These results have been recently published in [11] and [13].

3.1 Deep Neural Networks for Quantum Circuit Mapping

Sec. 1.1 has introduced the fundamental concepts of quantum computing and its division into two computational paradigms, such as the gate-based quantum computers and the quantum annealing paradigm. Let us now focus on the former paradigm. As said, theoretical quantum algorithms have already been designed for such a computational approach, but physical realizations of quantum computers able to run these algorithms have been considered a kind of utopia for a long time. Nevertheless, this changed in recent years in which quantum computers more and more evolved from an academic idea to an upcoming reality thanks to the effort of main majors acting in information and communication technology in developing so-called NISQ devices, i.e., quantum computers with 50-100 qubits may be able to perform tasks which surpass the capabilities of todays classical digital computers [144]. In this scenario, several companies are providing access to their quantum computers to a broad audience of researchers and practitioners by means of cloud computing technologies. However, although both advanced techniques for designing quantum algorithms and technologies capable of executing such algorithms are available, there are still some technological limitations, which slow down the race to quantum supremacy and advantage. Specifically, there is a significant gap between the quantum resources required to execute quantum algorithms and the resources available in current NISQ devices. In particular, current quantum processors based on superconductive qubits (the most widespread technology at the moment) are characterized by a weakly interconnected coupling map, which limits the interactions among different qubits. As a consequence, a quantum algorithm needs to be adapted to a specific quantum processor's coupling map in order to be correctly and efficiently run. This adaptation requires adding a set of SWAP quantum gates to the original quantum algorithm, increasing the quantum error rate and potentially compromising calculations. By virtue of this, there is a strong emergence for efficient techniques of circuit mapping able to minimize the number of SWAP gates useful to run the quantum algorithm in a correct way. This research faces this key challenge in quantum technologies by introducing *Neural Layout*, the very first approach for quantum circuit mapping based on machine learning, which uses deep neural networks to improve the performance of current methods based on mathematical solvers and heuristic cost functions. Indeed, as shown in Section 3.1.0.IV, Neural Layout is able to speed up state-of-the-art circuit mapping algorithms used by IBM Qiskit[1] Transpiler [2] when it is used to perform circuit mapping operations on 5-qubits IBM quantum processors. Moreover, the experiments show that Neural Layout outperforms other well-known machine learning techniques in carrying out quantum circuit mapping. The obtained results show that approaches based on deep neural networks can effectively support the design of quantum devices and open the way towards a completely new research area that blends the foundations of machine learning with those of quantum computing.

The chapter is structured as described hereafter. Section 3.1.0.I presents a collec-

[1]In this work the version of Qiskit meta-package used is the 0.19.6

[2]https://qiskit.org/documentation/stubs/qiskit.compiler.transpile.html

tion of approaches currently used to solve quantum circuit mapping, and it provides an overview about the application of machine learning to the design of classical electronics circuits. Section3.1.0.II gives a formal definition of the mapping quantum circuits problem. Section 3.1.0.III shows the design of Neural Layout and its capabilities in efficiently performing quantum circuit mapping. In Section 3.1.0.IV the performance of Neural Layout is assessed and compared to the performance obtained by other machine learning techniques, and to the performance yielded by state-of-the-art circuit mapping algorithms used in IBM Qiskit. Section 3.1.0.V provides some insights about future research directions mainly aimed at improving the performance of the proposed approach when applied to perform quantum circuit mapping on larger processors previously introduced.

3.1.0.I Related Works

Limiting the error during the current quantum computation is one of the biggest challenges researchers are facing. Efficiently mapping a quantum circuit to a processor is a fundamental step in this direction. Previous approaches to this problem can be classified into two classes. One is to formulate the circuit mapping problem into an equivalent mathematical problem and then apply well-known solvers [118, 51, 156, 157, 181, 115, 173, 172, 43, 132, 41]. However, these methods lack scalability as the number of qubits in the circuits increases, suffering from a very long run time. Thus, even though optimal solutions computed by these approaches are theoretically useful, they are impractical to be actually used in real scenarios. For this reason, some research proposes a second set of approaches, which are based on heuristic search for sub-optimal solutions [24, 153, 112, 180, 160, 99, 100, 40, 135]. However, most of the above methods were developed for ideal 1D/2D lattice models and they are not usable with superconductive NISQ devices, which are characterized by more complicated coupling maps. In this scenario, a quantum compiler used in IBM quantum computers, named IBM Transpiler, uses two different, not trivial, heuristic approaches to addressing circuit mapping[3]: *Dense Layout*[4] and *NoiseAdaptiveLayout* [127]. DenseLayout, carries out a breadth-first search starting from each qubit belonging to the processor, so as to compute a collection of connected subsets of qubits. Successively, this approach selects the subset characterized by the highest connectivity and low noise. Finally, the selected subset is given in input to the *reverse Cuthill Mckee algorithm* to order the qubits in the selected set in ascending order of degree of connectivity. *NoiseAdaptiveLayout* leverages a qubit mapping technique that uses the calibration information from the backend devices and evaluates several optimal and heuristic mappings. To the best of our knowledge, there are no previous works about the possibility to use artificial intelligence (AI) and machine learning (ML) to address the quantum circuit mapping problem as proposed in this manuscript.

Although quantum technologies are very recent and machine learning has never been used to support the development of these technologies, there is a strong scientific production where machine learning has been widely used in supporting design, simulation and optimization of classical circuits, as reported in [152]. Some examples of the recent and significant application of machine learning techniques applied to circuit design are shown hereafter. The research presented in [175] proposes a two-layer evolutionary scheme based on genetic programming (GP) and neural network (NN), which uses a divide-and-conquer approach to design analog circuits by especially focusing on how to select component values and topology sizes for a given circuit topology. Several researchers, instead, focus their attention on developing electronic design automation (EDA) techniques using machine learning [106, 146]. Furthermore, NNs have been used in applications ranging from circuits optimization [52], replacing empirical modeling solutions or numerical modeling methods limited by their computationally expansive behavior, to resource optimization for circuit

[3]At the time of this research the transpiler did not yet provide the possibility of using SABRE qubit placement algorithm presented in [107]

[4]DenseLayout

simulation [50] or to circuit partitioning, as shown in [184], where a neural network model was proposed for circuit bipartitioning. In particular, in this research, the massive parallelism of a NN has been successfully exploited to reduce the external wiring between the partitions and to balance the partitions of the circuit. In [28, 26, 163, 49, 27] NNs were also applied to develop a fault-diagnostic system for analog electronic circuits.
NNs are not the only machine learning algorithms applied to analog and digital circuit design and optimization. The research in [137] offers a review of different methods of Machine Learning such as K-nearest neighbor (KNN) Logistic regression (LR) or Support vector machines(SVM) and their usage in analog circuits. In particular, in [110] SVM and LR are compared to NN for the task of automated performance-driven placement of analog integrated circuits.

All the above studies prove that machine learning algorithms are particularly suitable to address issues in the design, simulation, and optimization of classical circuits by providing several benefits both in terms of the accuracy of solutions and computational time. These studies were the inspiration for using machine learning to support the design of quantum technologies by efficiently carrying out a critical task such as quantum circuit mapping, so as to prove that the aforementioned benefits can be used to address the limits previously highlighted by current circuit mapping approaches.

3.1.0.II Circuit Mapping on Quantum Processors

Currently, quantum computing is becoming a popular paradigm in computation thanks to the cloud-based availability of so-called Noisy Intermediate-Scale Quantum (NISQ) devices, i.e., quantum processors equipped with a low number (typically 50-100) of qubits not fully tolerant to quantum noise, which enable researchers and practitioners in developing quantum algorithms [144]. Besides the issues related to their size and noise, another critical problem that characterizes this kind of technol-

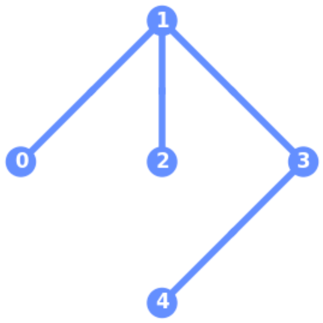


Figure 3.1: IBM Q Burlington coupling map

ogy is the low connectivity of their *coupling map*, for which each qubit is connected to a limited number of other qubits. As a consequence, two-qubit gates, such as a CNOT, cannot be placed in a circuit if the target and control qubits are not physically connected in the processor coupling map.

As an example, Fig. 3.1 shows the coupling map of an IBM Q processor named Burlington composed of $n = 5$ qubits. The IBM Q Burlington coupling map defines a set $\{(0,1), (1,0), (1,2), (2,1), (1,3), (3,1), (3,4), (4,3)\}$ containing the pairs of qubits that can be used as target and control in a CNOT gate; thus, on this processor, only 8 out of 20 $(n^2 - n)$ pairs can be used to position a CNOT in a circuit, severely limiting the possibilities offered by quantum devices. However, current quantum devices use so-called quantum compilers that are able to solve the above issue. These compilers use a sequence of SWAP operations (see Tab. 1.1) between adjacent qubits so as to enable the computation of a CNOT gate between two non-adjacent qubits. However, this approach could very negatively affect the execution of a quantum circuit for two fundamental reasons:

- It increases the circuit depth and, consequently, increases the probability of a decoherence error, with the consequent loss of the information that the circuit carries.
- Each CNOT gate has a typical error rate order 10^{-2}. Thus, the execution of a single SWAP operation involves the execution of three gates with a high error

rate, negatively affecting the computation of the whole circuit.

As a consequence, there is a strong emergence for quantum compilers able to identify an optimal initial mapping among circuit qubits and physical qubits, so as to minimize the number of SWAP operations required to execute the compiled circuit. This optimization problem is known as circuit mapping and it is formalized as described hereafter. Let $C_n = \{c_i\}_{i \in \mathbb{N}}$ be the set of all quantum circuits that can be designed with n qubits belonging to the set $Q_c = \{q_0^c, q_1^c, \ldots, q_n^c\}$, and let P be a NISQ processor composed of $m \geq n$ qubits belonging to the set $Q_p = \{q_0, q_1, \ldots, q_m\}$ and characterized by a coupling map $M_p = \{(q_i, q_j) \mid q_i, q_j \in Q_p \text{ and } i \neq j\}$, then a collection of initial quantum circuit mappings consists in a set of functions $F_M = \{f_k : C_n \to \wp(Q_C \times Q_P)\}_{k \in \mathbb{N}}$ where each f_k relates the qubits of each circuit in C_n to the qubits of the processor P; the solution of the circuit mapping problem is a function $f^* \in F_M$ able to minimize the number of SWAP operations to add to original circuits and enable its efficient execution on the processor P. In Fig.[5] 3.2 there is a graphical representation of a circuit mapping function $\bar{f} \in F_M$, which maps a circuit $\bar{c} \in C$ designed with $n = 5$ qubits on a processor P composed of $m = 20$ qubits and a coupling map M_P shown on the right side of the Figure. In particular $\bar{f}(\bar{c}) = \{(q_0^C, q_1), (q_1^C, q_0), (q_2^C, q_5), (q_3^C, q^6), (q_4^C, q_7)\}$.
The circuit mapping problem has been proved to be NP-complete in [45] and, as a consequence, the computation of its exact solution could be not suitable to deal with the future generation of quantum processors characterized by thousands or millions of qubits [6]. Thus, there is a strong need for approximate algorithms capable of efficiently computing sub-optimal solutions for this problem and paving the way for the next generation of compilers for quantum computers. In this research, this challenge has been faced by introducing the first approach based on machine learning to efficiently address the circuit mapping problem on real IBM Q processors.

[5]https://qiskit.org/documentation/apidoc/transpiler.html
[6]IBM Quantum Roadmap 2023

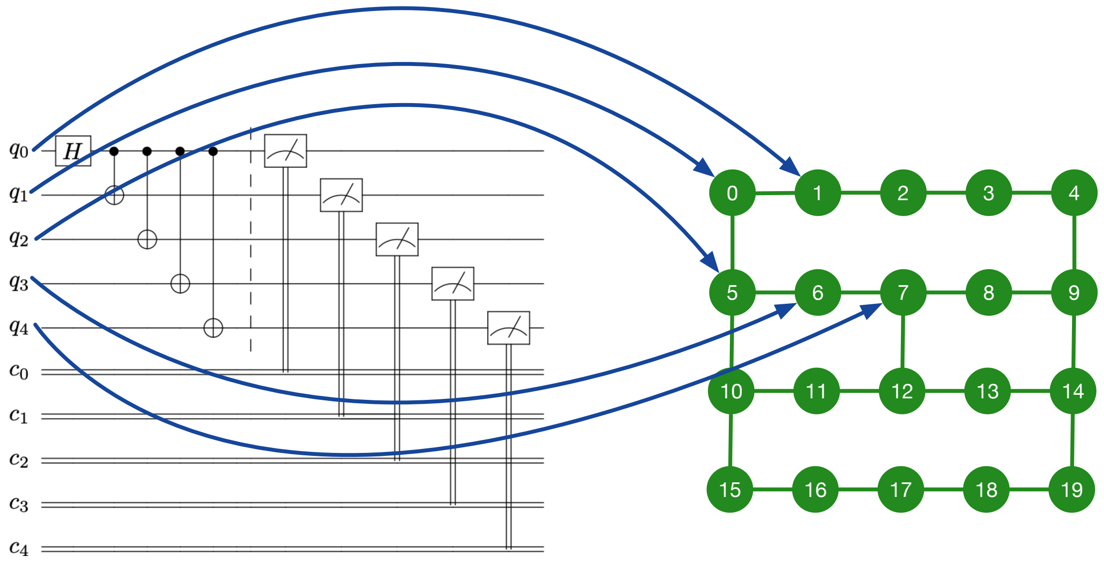


Figure 3.2: Graphical representation of a circuit mapping function $\bar{f} \in F_M$.

3.1.0.III Neural Layout

This section introduces Neural Layout, a machine learning-based approach aimed at solving the above circuit mapping problem. In particular, Neural Layout acts in three sequential steps:

1. initially, it models the quantum circuit mapping problem as a conventional classification task;
2. successively it trains an appropriate deep neural network aimed at efficiently addressing the classification task for circuit mapping;
3. finally, it uses a refinement step to make neural network predictions compliant with some logical constraints that characterize quantum processors.

Modelling Quantum Circuit Mapping as a Classification Task

In general, a classification problem can be defined as the task of estimating a label y from a K-dimensional input vector $\mathbf{x}$, where $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^K$ and $y \in \mathcal{Y} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_Q\}$. This task is accomplished by using a classification rule implemented by a function $g : \mathcal{X} \rightarrow \mathcal{Y}$ able to predict the label of new patterns, where g is learned and adjusted by using a training set composed of N points, represented by a set $D = \{(\mathbf{x}_i, y_i), \text{with } i = 1, \ldots, N\}$. In order to model the quantum circuit mapping problem by a classification task, let us consider a quantum circuit $c \in C_k$ composed of k qubits belonging to the set $Q_C = \{q_1^C, q_2^C, \ldots, q_k^C\}$, and a quantum processor P composed of n qubits belonging to the set $Q_P = \{q_1, q_2, \ldots, q_n\}$ and characterized by a specific coupling map M_P. Then, a classification task for quantum circuit mapping is implemented by a function ϕ that uses an input vector $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^K$, containing a set of features characterizing both the circuit c and the processor P on which running c, to estimate an array $\mathbf{y}$ composed of n elements, where each element belongs to the mapping label set $\mathcal{Y} = \{-1\} \cup \{1, \ldots, k\} \subset \mathbb{N}$. The function ϕ is learned and adjusted by using a training set $D_\phi = \{(\mathbf{x}_i, \mathbf{y}_i), \text{with } i = 1, \ldots, N\}$, where $\mathbf{x}_i \in \mathcal{X}$ is a feature set and $\mathbf{y}_i \in \mathcal{Y}^n$ is an array of mapping labels representing an ideal mapping from the circuit c to the processor P. Reasoning in this way, the function ϕ will be able to provide mapping capabilities similar to the best algorithms currently used in quantum circuit mapping, but with lower computational complexity, as shown in sec. 3.1.0.IV. The aforementioned input vector $\mathbf{x}$ is composed of a set of features characterizing both the circuit c and the processor P. In particular, with respect to the circuit c, the following pieces of information have been considered:

- an integer value representing the number of qubits composing the circuit;
- an integer value representing the total number of CNOT gates in the circuit c;
- a matrix of integer values where the item $[i, j]$ contains the number of CNOT gates between the control qubits q_i^C and the target qubit q_j^C of the circuit c. To make the network work with any circuit width less than or equal to the number of processor qubits, this matrix of integers is set of size $n \times n$.

At the same time, with respect to the processor P, the following pieces of information have been taken into account:

- an array of real values where each value represents the error rate of a CNOT using q_i as control qubit and q_j as target qubit for each $(q_i, q_j) \in M_P$;
- an array of real values where each value represents the execution time of a CNOT using q_i as control qubit and q_j as target qubit for each $(q_i, q_j) \in M_P$;
- an array of real values where each value represents the transverse relaxation time (T_2) characterizing a qubit q_i of the processor P
- an array of real values where each value represents the longitudinal relaxation time (T_1) characterizing a qubit q_i of the processor P
- an array of real values where each value represents the readout error characterizing a qubit q_i of the processor P.

A schematic view of the features related to the quantum circuit mapping problem is provided in Table 3.1. The output array $\mathbf{y} = [y_1, y_2, \ldots, y_k]$ is composed of n items where each $y_i \neq -1$ represents the mapping between the qubits $q^C_{y_i} \in C$ and $q_i \in P$, whereas $y_i = -1$ means that the processor qubit q_i must remain un-mapped. In other words, the collection $\{\mathbf{y}\}$ of all arrays generated by the above function ϕ by applying it to all circuits $c \in C_k$ encodes a function $f' : C \rightarrow \wp(Q_C \times Q_P) \in F_M$, and our approach uses the dataset D_ϕ to attempt to learn a function ϕ capable of approximating the aforementioned function f^* in a proper way.

In order to provide further clarification about the representation of the circuit mapping problem by a classification task, a practical example is proposed hereafter. Let us to consider a quantum circuit c made up of 5 qubits $Q_C = \{q^C_0, q^C_1, \ldots, q^C_4\}$ and a quantum processor P composed of 5 qubits $Q_P = \{q_0, q_1, \ldots, q_4\}$, and let us suppose that the above classification function ϕ computes an array $\mathbf{y} = [3, 2, 4, 0, 1]$ starting from the set of features modelling the circuit c and the processor P. Then, the array $\mathbf{y} = [3, 2, 4, 0, 1]$ corresponds to the following mapping between circuit and

Table 3.1: Summary table of the features used by the proposed DNN approach for circuit mapping. The first column contains the name of the feature; the column contains the description of each feature; the last column represents the dimensionality of each feature.

Quantum Circuit Features		
Feature Name	**Feature Description**	**Dimension**
N_{qubits}	Number of qubits in C	1
N_{CNOT}	Number of CNOTs in C	1
$N_{CNOT}^{i,j}$	Total number of CNOT gates between each pair of qubits q_i and q_j in C.	20
Quantum Processor Features		
$CNOT_{ER}$	CNOT error rate between each pair of qubits connected in the coupling map	8
$CNOT_{ET}$	CNOT execution time between each pair of qubits connected in the coupling map	8
T_2	Transverse relaxation time for each processor qubit	5
T_1	Longitudinal relaxation time for each processor qubit	5
E_{Ro}	Readout error for each processor qubit	5

processor qubits:

$$\begin{aligned} q_3^C &\to q_0; \\ q_2^C &\to q_1; \\ q_4^C &\to q_2; \\ q_0^C &\to q_3; \\ q_1^C &\to q_4. \end{aligned} \tag{3.1}$$

Once the circuit mapping problem has been defined as a classification task, it is necessary to design an appropriate algorithm based on a deep neural network to learn the function ϕ.

Designing a Deep Neural Network for Circuit Mapping

The identification of the aforementioned function ϕ is based on the design of Neural Layout, a deep neural network extended with a constraint satisfaction method. This network is composed of an input layer, a collection of hidden layers, and an output layer. The input layer has been designed by taking into account the size of the vector $\mathbf{x}$ introduced in the definition of circuit mapping as a classification task and containing the features that characterize both a quantum circuit and a quantum processor on which running the circuit. Let us suppose to have a circuit c composed of k qubits, P a quantum processor composed of m qubits, M_p the coupling map related to P, and l the cardinality of M_p, then the input layer of the proposed network is composed of $2 + (m^2 - m) + 3 \cdot m + 4 \cdot l$ nodes. The first term of this formula is related to two features representing the number of qubits and the number of CNOT gates belonging to the circuit c; the term $(m^2 - m)$ refers to the maximum number of CNOT gates that can be placed between each pair of qubits belonging to a generic circuit computable by the processor P; the term $3 \cdot m$ describes the number of features related to the qubits calibration data of the processor P, namely the decoherence times T_1, T_2, and the readout error; finally the term $4 \cdot l$ refers to the error rate and execution time of the CNOT gate for each pair of qubits connected in the coupling map M_p. The output layer of the proposed network has been designed to model the vector $\mathbf{y}$ introduced in the definition of circuit mapping as a classification task and contains the set of labels used to map the qubits related to the quantum circuit to the qubits related to the quantum processor where the circuit will be run. In particular, the output layer is organized in a collection of m slots, where each slot outputs a single item of the vector $\mathbf{y}$.

More properly, each slot can be considered as a complex neural structure composed of different hidden layers and an output layer embodying $m + 1$ neurons and equipped with a *softmax* activation function [131]. Formally speaking, the output layer of the ith slot, named $slot_i$, is an array $P(y_i) = \left[p_0^i, p_1^i, \ldots, p_{m-1}^i, p_{-1}^i\right]$, where p_j^i corresponds to a neuron representing the probability that the ith item of the array $\mathbf{y}$ is equals to j, and $\sum_{j=-1}^{m-1} p_j^i = 1$, due to the use of the softmax activation function. By virtue of this modeling scheme, the output vector $\mathbf{y}$ can be obtained by selecting the $\arg\max$ of each $P(y_i)$ value related to each $slot_i$ with $i = 0, 1, \ldots, m$, as shown in (3.2).

$$\mathbf{y} = [\arg\max(P(y_0)), \arg\max(P(y_1)), \ldots, \arg\max(P(y_m))] \tag{3.2}$$

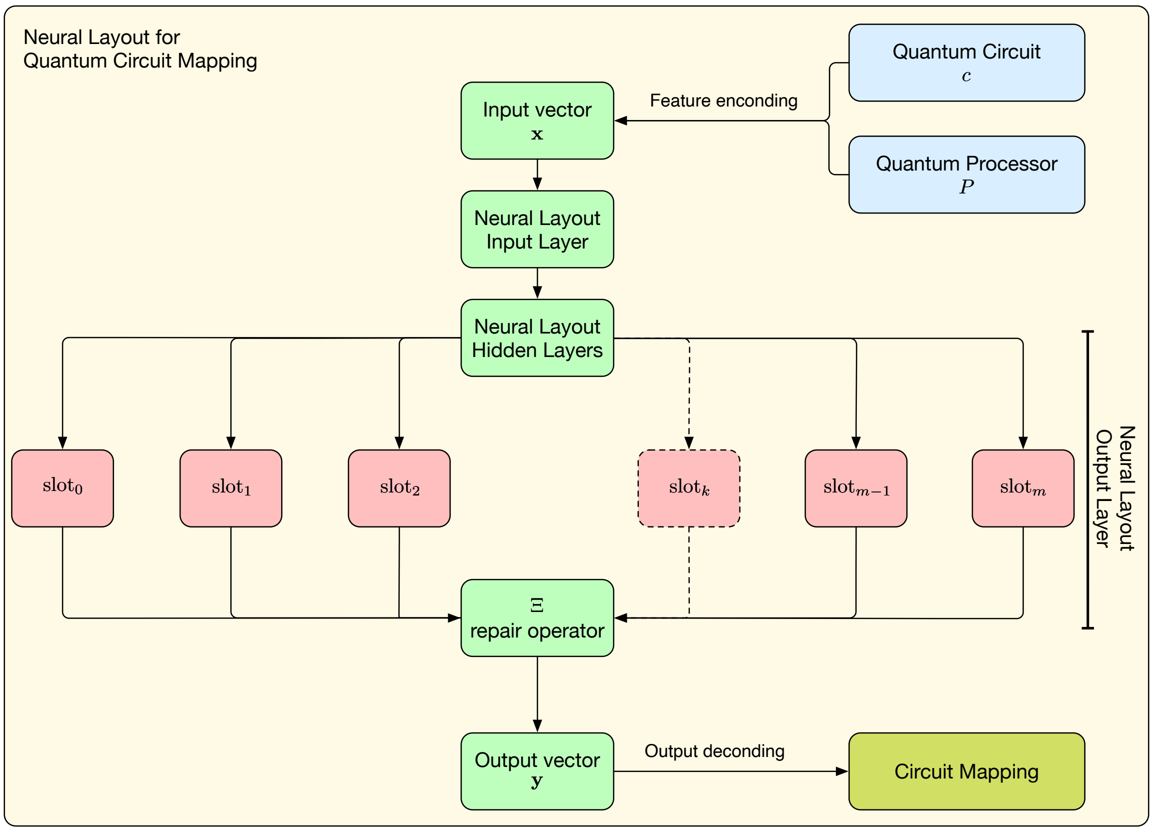


Figure 3.3: Circuit mapping workflow via Neural Layout.

However, this approach can compute not feasible solutions for the circuit mapping problem, because it is not certain that $\arg\max(P(y_i)) \neq \arg\max(P(y_j))$ $\forall i, j$, and $i \neq j$. When that happens, it would lead to mapping the same circuit qubit q^C onto two different processor qubits q_i and q_j (see the mapping in (3.3) in the case study for a practical demonstration of this issue). In order to address this feasibility issue, our approach introduces and uses a repair operator to move the set of unfeasible solutions generated by the above deep neural network to a feasibility area. This repair operator, named Ξ, can be summarized as follows:

1) Let $\mathcal{F} = \{P(y_i)\}_{i=0}^{m-1}$ be the set of vectors computed by the different slots of the DNN;

2) Let $\bar{\mathbf{y}} = [\bar{y_0}, \bar{y_1}, \ldots, \bar{y_m}]$ be a feasible output vector composed of m items and initially initialized as $\bar{\mathbf{y}} = [-1, -1, \ldots, -1]$;

3) Let $\mathcal{F}' = \{P'(y_i)\}_{i=0}^{m-1}$ be the set of vectors composed by the first k components of each vector in $\mathcal{F}$;

4) Let $\kappa = 1$ be an iteration variable;

5) Let y_{jt} be the κ-th maximum among all probability values stored in the vectors $P'(y_i)$, with $i = 0, \ldots, m-1$, belonging to the set $\mathcal{F}'$, located in the t-th position of the vector $P'(y_j)$;

6) If the value t is not present in $\bar{\mathbf{y}}$, set the value of the j-th item of the vector $\bar{\mathbf{y}}$ to t: $\bar{\mathbf{y}}_j = t$. Go to step 8);

7) If t is present in $\bar{\mathbf{y}}$, set $\kappa = \kappa + 1$ and go to the step 5);

8) Remove $P'(y_j)$ from the collection $\mathcal{F}'$: $\mathcal{F}' = \mathcal{F}' - P'(y_j)$;

9) If the number of items equal to -1 in the vector $\bar{\mathbf{y}}$ is equal to $m - k$ go to step 10), else go to step 4) and find a new item for the vector $\bar{\mathbf{y}}$;

10) End.

In the case study presented in the next section, a step-by-step application of the repair operator is provided (See (3.4)).

Once obtained a feasible output vector $\bar{\mathbf{y}}$ is then possible to decode the circuit mapping that it encodes by following the procedure shown in Section 3.1.0.III. The number and type of hidden layers of the proposed network are identified by means of an experimental approach, as shown in the Section 3.1.0.IV. A complete graphical view of the proposed approach for quantum circuit mapping is summarized in Fig. 3.3.

A case study for IBM Q Burlington

This section shows the proposed circuit mapping approach at works on a real 5-qubits processor from IBM, named IBM Q Burlington, whose coupling map is shown in Fig. 3.1. From this map it appears that $m = 5$ and $l = 4$ and, as a consequence, the required input layer is composed of 53 nodes, whereas the output layer is characterized by $m = 5$ slots, where the i-th slot is a neural structure composed of two hidden dense layers characterized by a relu activation function [131] and an output layer containing $m + 1 = 6$ neurons representing the output vector $P(y_i) = [p_0^i, p_1^i, p_2^i, p_3^i, p_4^i, p_{-1}^i]$ and characterized by a softmax activation function. The structure of the generic slot is shown in Fig. 3.5.

The set of hidden layers for the whole network is composed of three different layers, where the first two are dense layers, whereas the third layer is a dropout layer that during training time, turns off in a random way some neurons from the previous layer helping to prevent overfitting. The architecture of Neural Layout for circuit mapping on IBM Q Burlington is graphically presented in Fig. 3.4. In the topology shown in the figure, each block is a neural network layer and the number of neurons for each corresponds to the number of columns of the output tensor.

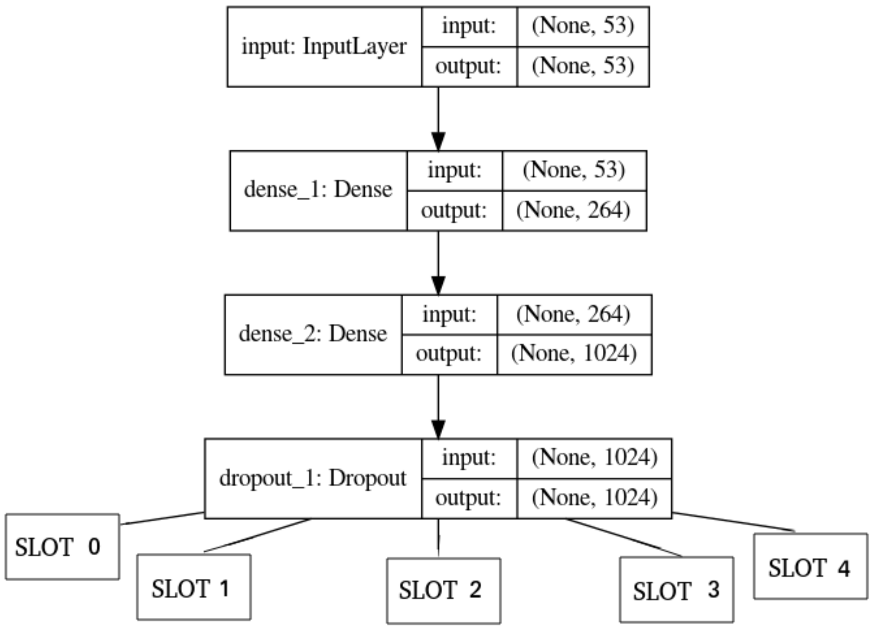


Figure 3.4: DNN topology in Neural Layout for mapping quantum circuit on IBM Q Burlington processor.

The behaviour of Neural Layout applied to the IBM Q Burlington has been

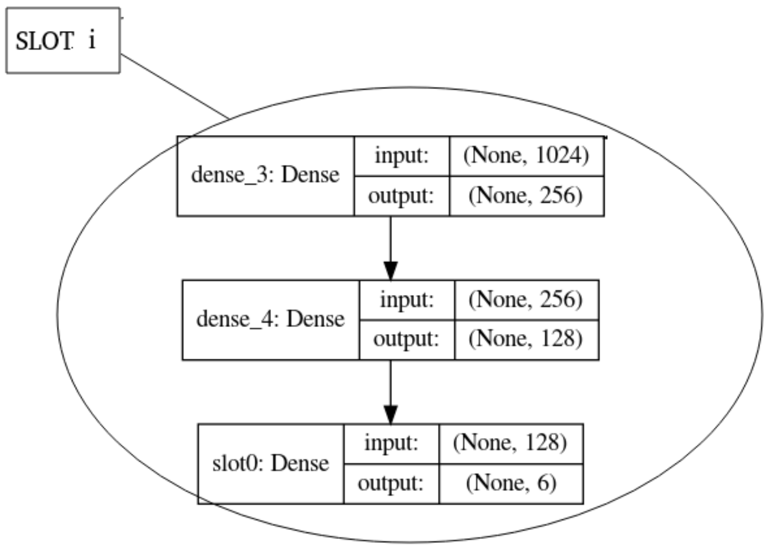


Figure 3.5: SLOT topology.

assessed by considering a random quantum circuit shown in the Fig. 3.6. Moreover, the mapping result computed by Neural Layout has been compared with the so-called *Trivial Mapping Algorithm* provided by IBM Qiskit.

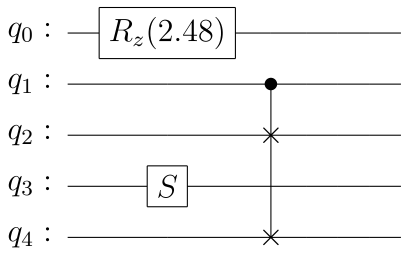


Figure 3.6: A case study quantum circuit

Before applying circuit mapping algorithms it is necessary to convert the starting circuit into an equivalent circuit containing only elementary gates used by the IBM Q Burlington processor and belonging to the set $\{U_1, U_2, U_3, CNOT, I\}$. This step is named circuit unrolling. Once the circuit is unrolled, its 53 features are encoded in a vector $\mathbf{x}$ which is given input to Neural Layout so as to compute the following set of output vectors:

$$\begin{aligned}
&Slot_0 \rightarrow P(y_0) \rightarrow [0.15, 0.23, 0.12, 0.06, \mathbf{0.4}, 0.04] && \rightarrow \arg\max = 4 \\
&Slot_1 \rightarrow P(y_1) \rightarrow [0.08, 0.16, \mathbf{0.25}, 0.13, 0.2, 0.18] && \rightarrow \arg\max = 2 \\
&Slot_2 \rightarrow P(y_2) \rightarrow [0.05, 0.06, 0.31, \mathbf{0.33}, 0.2, 0.05] && \rightarrow \arg\max = 3 \\
&Slot_3 \rightarrow P(y_3) \rightarrow [\mathbf{0.45}, 0.1, 0.03, 0.15, 0.17, 0.1] && \rightarrow \arg\max = 0 \\
&Slot_4 \rightarrow P(y_0) \rightarrow [\mathbf{0.27}, 0.2, 0.18, 0.06, 0.18, 0.11] && \rightarrow \arg\max = 0
\end{aligned} \tag{3.3}$$

The outputs related to the different slots of Neural Layout are aggregated together to compute the circuit mapping output vector $\mathbf{y} = [4, 2, 3, 0, 0]$, which unfortunately corresponds to a not feasible solution. However, Ξ operator used by Neural Layout is able to solve this issue by incrementally computing the output vector $\bar{\mathbf{y}}$ as follows:

$$\begin{aligned}
&\mathcal{F}' = \{P'(y_0), P'(y_1), P'(y_2), P'(y_3), P'(y_4)\} && \rightarrow \quad y_3^0 \quad \rightarrow \quad \bar{\mathbf{y}} = [-1, -1, -1, 0, -1] \\
&\mathcal{F}' = \{P'(y_0), P'(y_1), P'(y_2), P'(y_4)\} && \rightarrow \quad y_0^4 \quad \rightarrow \quad \bar{\mathbf{y}} = [4, -1, -1, 0, -1] \\
&\mathcal{F}' = \{P'(y_1), P'(y_2), P'(y_4)\} && \rightarrow \quad y_2^3 \quad \rightarrow \quad \bar{\mathbf{y}} = [4, -1, 3, 0, -1] \\
&\mathcal{F}' = \{P'(y_1), P'(y_4)\} && \rightarrow \quad y_1^2 \quad \rightarrow \quad \bar{\mathbf{y}} = [4, 2, 3, 0, -1] \\
&\mathcal{F}' = \{P'(y_4)\} && \rightarrow \quad y_4^1 \quad \rightarrow \quad \bar{\mathbf{y}} = [4, 2, 3, 0, 1]
\end{aligned} \tag{3.4}$$

Thus, the final vector representing the computed quantum circuit mapping is $\bar{\mathbf{y}} = [4, 2, 3, 0, 1]$. The quantum circuit obtained by using the circuit mapping encoded in the vector $\bar{\mathbf{y}}$, shown on the right side of Table 3.2, is characterized by 10 CNOT gates and a single SWAP gate. Vice versa, a quantum circuit obtained by using the trivial approach provided by IBM Qiskit, shown on the left side of Table 3.2, is characterized by 16 CNOTs and 3 SWAP gates. In this way, it is possible to state that Neural Layout generates a quantum circuit more tolerant to the effects of quantum noise than traditional approaches when used to run the circuit shown in Fig. 3.6.

3.1.0.IV Experimental Results

In this section, the suitability of Neural Layout in performing quantum circuit mapping operations is assessed by comparing its performance with those yielded by other machine learning techniques, and other techniques used in real quantum compilers, such as IBM Qiskit Transpiler. These comparisons were made taking into account both quality of mappings and computational times. The experiments were conducted by using a dataset composed of 5-qubit random circuits mapped on a specific processor provided by the IBM Q Experience, namely IBM Q Burlington.

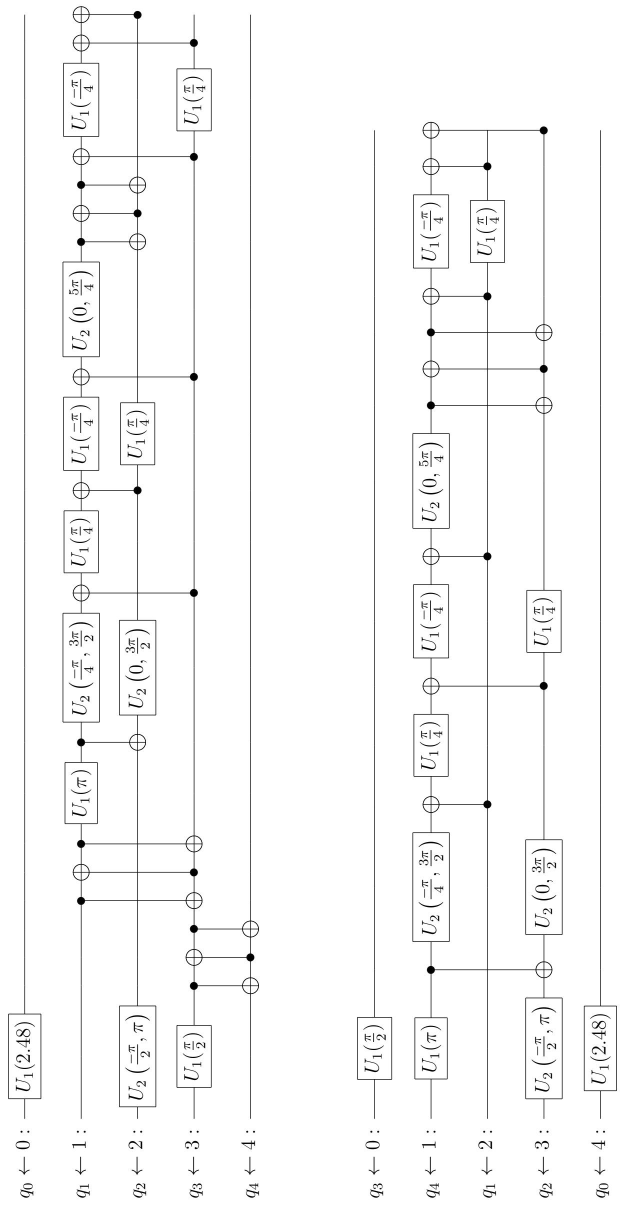
Dataset Creation for Quantum Circuit Mapping

The dataset used to train and test Neural Layout is composed of 42039 random unrolled quantum circuits operating on 5 qubits and characterized by a maximum of 10 CNOT gates.

From each circuit, twenty-two features have been extracted (see Section 3.1.0.III). Moreover, besides circuits' features, the dataset also contains a collection of features related to the processor where the above circuits are mapped on, namely IBM Q Burlington, which have been collected by using calibration data provided by IBM[7]. Each instance belonging to the dataset is then related to an output label encoding the best circuit mapping computed by using well-known algorithms available in IBM Qiskit, namely *Dense Layout* and *Noise Adaptive Layout*. In this context, the best mapping of a circuit is the one that generates a new circuit characterized by the smaller number of SWAP gates. The number of SWAP gates in a circuit is computed by means of two IBM Qiskit routing algorithms, named *Look ahead Swap* [8] and *Stochastic Swap* [9].

[7]https://qiskit.org/documentation/stubs/qiskit.providers.models.BackendProperties.html
[8]IBM Qiskit Lookahead Swap Algorithm
[9]IBM Qiskit Stochastic Swap Algorithm

Table 3.2: Final quantum circuit obtained using a naive mapping strategy on the left and the Neural Layout mapping approach on the right. The circuits must be read from the bottom. At the beginning of each line there is the mapping used. The integers refer to circuit qubits, while q_is are processor qubits.



Therefore, the process of labeling each circuit c randomly generated is outlined as follows:

1. Compute two initial mappings for the circuit c by using both *Dense Layout* and *Noise Adaptive Layout* approaches;
2. For each mapping computed at the previous step, compute the number of SWAPs needed to run the circuit c by using both *Look ahead Swap* and *Stochastic Swap* approaches;
3. Choose the mapping requiring the smaller number of SWAP gate executions.

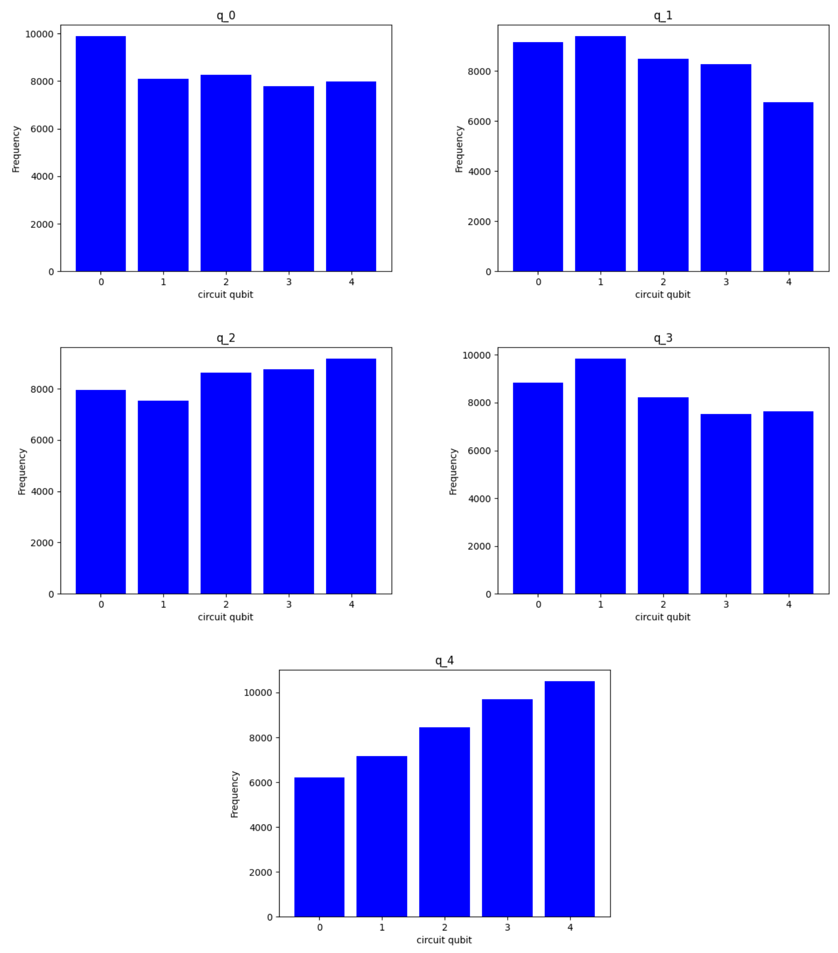


Figure 3.7: Histograms representing the target frequencies for each processor qubit of 42039 random quantum circuits used to train and test our model.

However, the above random generation of quantum circuits does not ensure a perfect balance of output classes in the dataset. Indeed, each histogram in Fig. 3.7 highlights the imbalance in output classes by showing how many times each qubit of the circuit has been mapped to its processor qubit by means of the above approach. Therefore, to develop an efficient predictor, two precautions have been taken. Firstly, a weighted average of the cost function with respect to the target frequencies for each output layer in the model was done. Secondly, appropriate dropout layers have been added to our model to face overfitting due to data imbalance, as shown in Section 3.1.0.IV.

Analysis of Experimental Results

The experimental results section has been organized into three parts. In the first part, a complete experimental setting of the optimal configuration of Neural Layout has been performed so as to identify the right number of hidden layers able to maximize the network accuracy. In the second part, the optimized Neural Layout has been compared to other well-known classifiers such as Random Forest, Support Vector Machine, and Logistic Regression Cross Validation, in order to validate its superiority in solving the problem under consideration with respect to other machine learning techniques. In the last part, the performance of the optimized Neural Layout has been compared to the performance yielded by state-of-the-art quantum circuit mapping algorithms provided by IBM Qiskit, both in terms of circuit mapping accuracy and running time, so as to prove that the proposed approach is ready to be used in real quantum compilers.

Table 3.3: Test and Training Accuracy of the four Neural Layout configurations.

Configuration	*Test Accuracy*	*Training Accuracy*
Configuration 1	0.7041	0.8471
Configuration 2	0.7548	0.8867
Configuration 3	0.7188	0.8295
Configuration 4	**0.7645**	0.8752

Experimental setting of Neural Layout

This experimental session is aimed at identifying the best configuration of Neural Layout. Here, four different configurations, based on the use or not of the Ξ operator and a dropout layer have been considered:

1. Neural Layout composed of an input layer of 53 units, a dense layer of 264 units (D(264)), a dense layer of 1024 units (D(1024)), a dropout layer, and five output slots, without Ξ operator and dropout layer in slot 4;

2. Neural Layout composed of an input layer of 53 units, a dense layer of 264 units (D(264)), a dense layer of 1024 units (D(1024)), a dropout layer, and five output slots extended with Ξ operator;

3. Neural Layout composed of an input layer of 53 units, a dense layer of 264 units (D(264)), a dense layer of 1024 units (D(1024)), a dropout layer, and five output slots extended with a dropout layer in Slot 4 to face overfitting due to the imbalance of the dataset;

4. Neural Layout composed of an input layer of 53 units, a dense layer of 264 units (D(264)), a dense layer of 1024 units (D(1024)), a dropout layer, and five output slots extended with both Ξ operator and a dropout layer in Slot 4 to face overfitting due to the imbalance of the dataset;

Each of the above configurations has been trained and tested by using the above dataset composed of 42039 5-qubit random quantum circuits split in training(80%) - test(10%) - validation(10%) sets. Each configuration has been trained for 150 epochs using Adam Optimizer with a learning rate equal to 0.0005. The experiments were run on a classical computer equipped with an Intel i9 processor and 128 Gb of RAM. The results obtained in terms of accuracy are shown in Table 3.3. The best accuracy on the test set was obtained by configuration 4. It should be emphasized that by using the repair operator Ξ accuracy increases significantly, while the exploitation of a dropout layer in slot 4 results in a percentage increase in test accuracy and a simultaneous decrease in training accuracy, which can be interpreted as a reduction of overfitting phenomenon.

Once the best configuration has been identified, further analysis is performed by using confusion matrices to evaluate the performance of this configuration in correctly identifying the i-th qubit of the output vector, with $i \in \{0, 1, 2, 3, 4\}$ (see Fig. 3.8). The accuracy obtained by the best configuration in identifying the i-th qubit of the output vector is reported in Table 3.4. Here, the slight difference between the values is due to the aforementioned imbalance present in the dataset.

In the next sections, the identified best model Neural Layout will be compared with other machine learning classifiers and with other deterministic quantum circuit mapping algorithms already used in real quantum compilers.

Table 3.4: Accuracy of individual output slots related to Neural Layout extended with Ξ operator and a dropout layer for slot 4.

	Slot 0	Slot 1	Slot 2	Slot 3	Slot 4
Test Accuracy	0.85	0.85	0.84	0.88	0.92

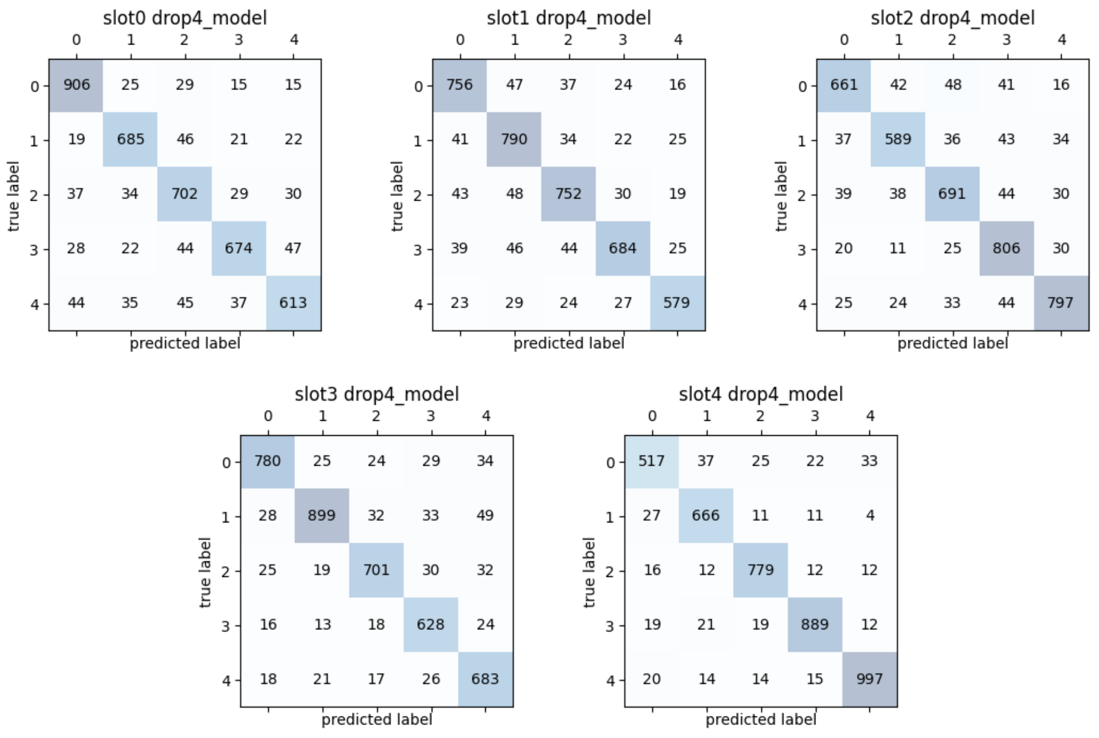


Figure 3.8: Confusion matrices related to the different output slots for Neural Layout.

Comparing Neural Layout and Machine Learning Techniques for Circuit Mapping

In this section, well-known machine learning classifiers, such as Random Forest (RF), Support Vector Machine (SVM) and Logistic Regression (LR-CV) are compared with Neural Layout with respect to their capabilities in solving the quantum circuit mapping problem modeled by a classification task. These classifiers can be easily adapted to perform the multi-output classification that the proposed mapping approach requires. In order to allow these machine learning techniques to generate feasible solution for the problem, Ξ operator has been applied to their output. For each classifier different combinations of hyperparameters have been tried in order to identify a suitable model to solve the quantum circuit mapping problem. All these combinations are summarized in Table 3.5: Random Forest has been tested with several numbers of estimators and two different splitting criteria for the construction of de-

Table 3.5: Hyperparameter setting for machine learning techniques.

Classifier	**Hyperparameters**	
SVM	**Kernel Function**	**C Value**
	Rbf	[0.01, 0.1, 1.0, 10.0, 20.0, 50.0, 100.0, 500.0]
	Poly	[10.0, 20.0, 50.0, 100.0, 500.0]
LR - CV	**CV**	**C Value**
	5	[10.0, 20.0, 50.0, 100.0, 500.0]
RF	**Criterion**	**N Estimators**
	Gini	[100, 500, 1000]
	Entropy	[100, 500, 1000]

cision trees, namely Gini and Entropy criterion; furthermore, SVM has been tested with two different kernel functions; LR-CV has been tested by using several values of the regularization parameter c. All these techniques have been trained and tested by using the dataset prepared in Section 3.1.0.IV and, as shown in Fig. 3.9, performance yielded by these machine learning techniques in terms of test accuracy is worse than the best configuration of Neural Layout identified in the previous section

As already done for Neural Layout, a further verification of the performance for these classifiers has been made by calculating the test accuracy for the individual output slots. The obtained results are shown in Table 3.6. This table is a further confirmation of how Neural Layout is far more suitable to perform the operation of mapping quantum circuits than conventional machine learning models. Indeed, the best classifier, Random Forest with Gini criterion, has an average accuracy value for individual slots that is more than 10% lower than Neural Layout, reflecting an overall accuracy value that is more than a 15% lower than that provided by Neural Layout.

Comparing Neural Layout and IBM Qiskit Mapping Algorithms

In the last step of the evaluation process, the performance of the optimal configuration of Neural Layout have been compared with two state-of-the-art quantum circuit

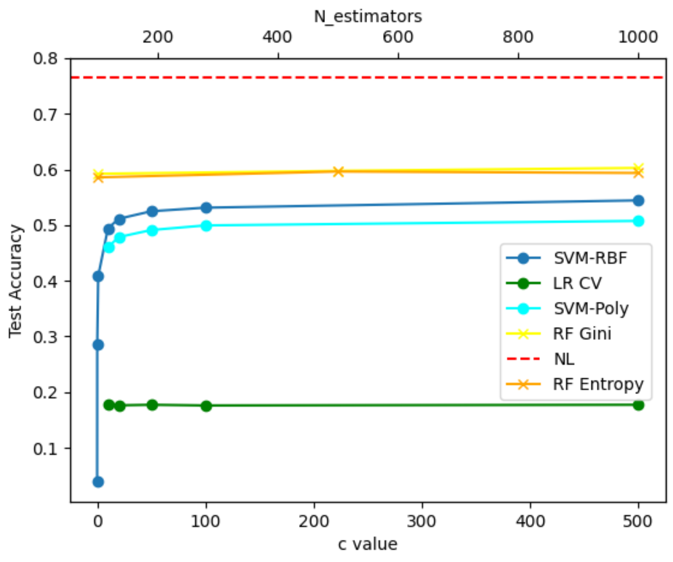


Figure 3.9: Comparing test accuracy of Neural Layout (NL) and other machine learning techniques.

Table 3.6: Test Accuracy of output Slots for each classic classifier.

Test Accuracy					
Classifier	**Slot 0**	**Slot 1**	**Slot 2**	**Slot 3**	**Slot 4**
SVM (Rbf)	0.69	0.74	0.70	0.73	0.82
SVM (Poly)	0.67	0.72	0.67	0.72	0.80
LR	0.43	0.60	0.30	0.47	0.66
RF (Gini)	0.72	0.76	0.75	0.75	0.84
RF (Entropy)	0.71	0.76	0.75	0.75	0.83

mapping algorithms, namely Dense Layout and Noise Adaptive Layout, provided by IBM Qiskit, both in terms of quality of the mappings and running times. For this purpose 1000 random quantum circuits were generated and unrolled in terms of the basic gates. The total number of CNOTs belonging to the unrolled circuits varies between 0 and 50. These unrolled circuits have been mapped on the IBM Q Burlington processor, by using two circuit mappings: the one computed by Neural Layout and the best circuit mapping returned by the execution of the two aforementioned algorithms provided by IBM Qiskit. For each of these two mappings, the depth of the final circuit was calculated by using both routing algorithms made available by Qiskit: Look ahead Swap (LAS) and Stochastic Swap (SS). The plots in Fig. 3.10

show the results about the quality of the mappings proposed by Neural Layout, as the routing algorithm used varies. On the x axis there is the number of CNOTs in the unrolled circuit, on the y there is the percentage of final circuits in which the initial mapping provided by Neural Layout returns a circuit characterized by a depth less than or equal to that of the best Qiksit mapping. The plots in blue and red refer to the different routing methods used to obtain the final circuits: in blue the required SWAP insertions were computed with the LAS routing algorithm, while in red they were computed with SS routing algorithm. In green there is the percentage of random quantum circuits mapped by Neural Layout which have a depth less than or equal to the one obtained by the best mapping provided by Qiskit using at least one of the two routing algorithms. The random quantum circuits generated were 1000, with a number of CNOTs after the unrolling, less or equal to 50. Table 3.7 shows the average percentage values and relative standard deviation for the three cases above considered. It should be also emphasized that although the network has been trained through unrolled circuits with a maximum of 10 CNOTs, it manages to keep its performance almost constant even as the number of CNOTs within the circuits increases.

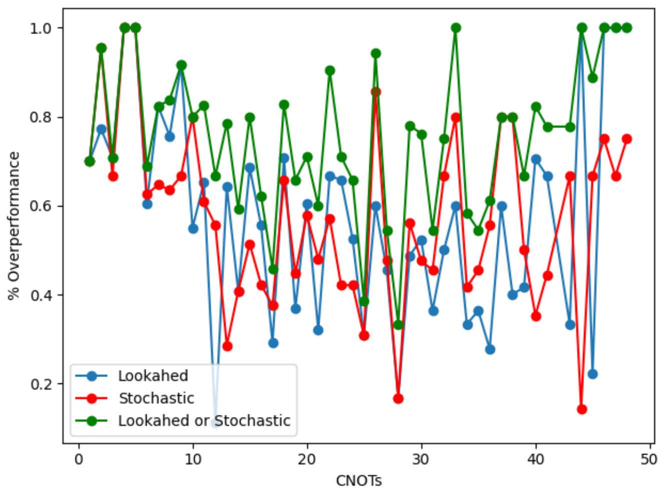


Figure 3.10: Percentage of final circuits mapped by DNN resulting in a depth less than or equal to the depth resulted after the Qiskit best map.

Table 3.7: Mean values and relative standard deviations of the percentages of 1000 random circuits mapped via DNN resulted in a final circuits less or equal deep to the final circuits obtained mapping the random circuit via Qiskit algorithms depending on the routing algorithm used.

Routing Algorithm	**Mean**	**StDev**
LAS	0.582	0.239
SS	0.578	0.195
LAS or SS	0.757	0.168

As further proof of the suitability of the proposed approach, a comparative analysis of running times of Neural Layout, Dense Layout and Noise Adaptive Layout has been performed. This comparison was made by generating random quantum circuits of different depths and mapping these on the IBM Q Burlington processor by using the three above algorithms. For each depth value, 10 random circuits were generated and the mean value of the mapping time for each depth has been collected. These values are shown in the Fig. 3.11 according to the mapping algorithm used.

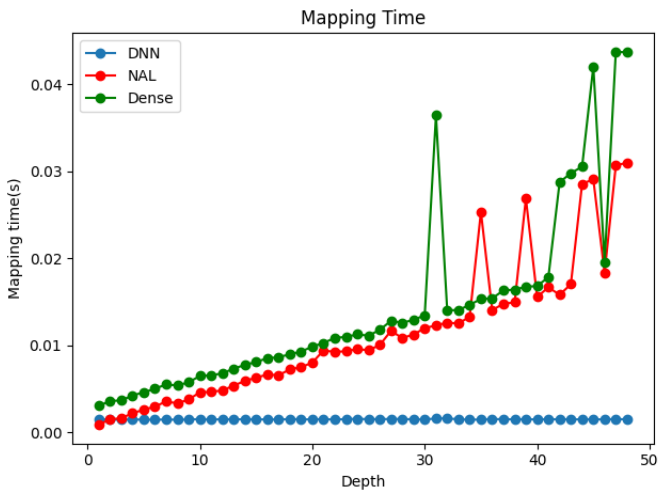


Figure 3.11: Mapping time in second of each mapping algorithm as the depth of the circuit to map varies. A point in these plots is the average mapping time value of 10 circuits mapped.

As highlighted in the figure, the computational time taken by Neural Layout remains constant as the depth of the circuits increases. On the contrary, the times taken by Qiskit algorithms grow linearly with the depth. This result, combined with a good accuracy of Neural Layout, encourages the use of this mapping approach for larger circuits and processors. Indeed, the constant execution time means that the mapping of quantum circuits via the Neural Layer can be a strong candidate to solve the problem of scalability that occurs in current mapping algorithms when applied large circuits to large processors.

3.1.0.V Summary

This chapter introduced Neural Layout, the very first approach aimed at facing the quantum circuit mapping problem by means of machine learning techniques. In order to achieve this goal, the circuit mapping problem has been modeled as a conventional classification task and, successively, a deep neural network, characterized by a proper output layer, has been integrated with a repair operator capable of moving values computed by the neural network towards feasibility regions and allowing the proposed approach to work correctly. The performance yielded by Neural Layout have been compared to that obtained by other well-known machine learning techniques when applied to the circuit mapping problem and, in all cases, Neural Layout proved to be to most efficient one. Moreover, Neural Layout has been compared to two state-of-the-art algorithms for quantum circuit mapping belonging to IBM Qiskit and, in this case, Neural Layout showed similar performance in terms of mapping accuracy, but a considerable speedup in terms of running time, making the proposed approach appropriate to be used in real quantum computing environments, such as IBM Q Experience.

Considering the results obtained by Neural Layout both in terms of quality and runtime of the mappings, it can be further developed to significantly impact the current state-of-the-art for the quantum circuit mapping algorithms. Indeed, the proposed DNN model has to be considered as a proof that machine learning can be used to address quantum circuit mapping problem opportunely modeled as a classification task, but further investigations on the optimization of the proposed model will be conducted in future studies. These results lay the foundation to a new mapping paradigm and despite this approach has been tested on *ibmq_burlington* processor, the procedure to build and test models is completely independent from the processor used. Therefore, practically implementation of Neural Layout for other processors requires only the construction of proper datasets using the workflow proposed in section 3.1.0.IV. Moreover, in [13]we released a collection of datasets for other IBM Quantum processor. Furthermore, the described way to collect data can be integrated with other deterministic mapping algorithms such as *SABRE Layout*[107] or better algorithms which will be developed in future, so as to increase the quality of the training data for classification models. The practical application of NL can be further supported by means of *online-training* techniques of the prediction models, which could be re-trained daily thanks to the calibration data provided simultaneously by IBM for each quantum processors.

In the future, other important studies will be conducted on the possibility to embed NL in a recursive framework, where unsupervised machine learning techniques will be used to partition n-qubit circuits ($n >> 5$) in a collection of equivalent k-qubits circuits ($k \leq 5$) to be mapped to m-qubit quantum processors ($m >> 5$) so as to enable an efficient quantum circuit mapping on future generation of quantum computers.

Conclusion

This thesis investigated the synergy between Quantum Computing and Computational Intelligence with a dichotomic approach: on the one hand, new quantum-enhanced computational intelligence algorithms were proposed, and, on the other hand, classical computational intelligence algorithms have been exploited to improve the current status of quantum computers. In detail, new quantum genetic operators and fuzzy inference engines were developed in the first research line. In the former field, the goal was to propose quantum genetic operators able to be run on current NISQ devices and exploit quantum noise to enhance the exploration capability of classical genetic algorithms. The suitability of these approaches in finding better solutions than the classical counterparts has been confirmed almost in all the experimental studies carried out. In the latter field, it has been proved that both quantum annealers and quantum digital computers can carry out fuzzy inference. Moreover, the QFIE algorithm proposed for quantum digital computers is able to achieve an exponential speed-up in computing fuzzy rules over the classical FRBS. Remarkably, all the QFIEs proposed have been tested both by means of simulations and execution on real quantum machines, and a QFIE-based FRBS has been exploited for the very first time to control an actual particle accelerator at CERN.

The application scenario of the above algorithms is enormous. After all, any classical genetic algorithm and any classical fuzzy inference engine can potentially be replaced by the quantum counterpart introduced in this thesis. For example, in dynamic and volatile financial markets, it is crucial to use genetic algorithms capable of finding excellent solutions in very fast search times. The proposed quantum genetic algorithms are therefore perfect candidates to improve this aspect. Moreover, to date many decision-making tasks performed by AI are performed by machine learning algorithms. However, if one thinks of sensitive decision-making contexts such as medical or industrial, where the interpretability of the AI models used is almost as important as efficiency, almost all the machine learning models are black boxes where one may be wary of trusting them. At the same time, the more interpretable classical FRBSs cannot be used because of the poor ability to handle the large number of rules that controlling these environments would require. However, QFIE could revolutionize these application scenarios since on the one hand it would preserve the interpretability of classical FRBSs, and on the other hand it would exploit the exponential computational advantage in computing the large number of fuzzy rules.

Despite these results and these potential impacts, a critical reader might doubt the relevance of the proposed work, criticizing in particular the complexity of the problems used as benchmarks to validate these innovative quantum algorithms that indeed, can still be solved by classical counterparts. It should be made clear to such a reader that the limitations of the experiments carried out in this thesis are due solely to the current state of NISQ quantum computers: the limited number of qubits, the high error rates, and the relatively short qubits' coherence times don't enable the reliable execution of complex algorithms, which obviously cannot either be classically simulated over a certain size (otherwise what would be the point of quantum computers?). Therefore, the algorithms proposed in this thesis that are among the first (and in the case of fuzzy systems the very first) quantum computational intelligence algorithms will be able to replace their classical counterparts as soon as the hardware allows. But, considering that from punched-card computers today, we daily use smartphones or calculators that were inconceivable until a few years ago, this step into the future is very likely according to the most popular opinion. In this direction, Chapter 3 investigates the possibility of using classical CI algorithms to speed up this technological process, with a particular focus on the problem of quantum compiling. The research reported in Chapter 3 proposes to exploit machine learning and in particular neural networks to solve the circuit mapping problem on quantum chips. This study, together with the other accomplishments that companies and academies around the world are achieving, will allow the results obtained in the benchmark problems studied in this thesis to be extended into problems that classically could not be solved.

If the reader is still critical, it should be pointed out that the world is making a revolution toward quantum technology not only in computing. As proven by initiatives of major world governments, such as the European Union's Quantum Flaghsiph (`https://qt.eu`) or the U.S. National Quantum Initiative (`https://www.quantum.gov`) quantum technology is set to heavily change the lives of even ordinary citizens. Quantum sensing, metrology, and communication will produce scenarios where data and protocols will be different from the classical data and protocols we know. As a result, the way we process information will also have to change. To intelligently compute this new kind of information a new form of quantum computational intelligence is required. This thesis tries to be a first step in that direction.

Code Availability Statements

Scan the QR Code for accessing the GitHub Repository to implement Genetic Algorithms equipped with QMO and QGS.

Scan the QR code to open the QFIE Python library. There, you can read about the documentation and look at several tutorials.

Bibliography

[1] Qiskit: An open-source framework for quantum computing, 2021.

[2] Giovanni Acampora, Vittorio Cataudella, Pratibha Raghupati Hegde, Procolo Lucignano, Gianluca Passarelli, and Autilia Vitiello. An evolutionary strategy for finding effective quantum 2-body hamiltonians of p-body interacting systems. *Quantum Machine Intelligence*, 1(3):113–122, 2019.

[3] Giovanni Acampora, Angela Chiatto, Luigi Coraggio, Giovanni De Gregorio, Roberto Schiattarella, and Autilia Vitiello. Genetic algorithms for constructing effective nuclear shell-model hamiltonians. In *2023 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2023.

[4] Giovanni Acampora, Angela Chiatto, Stefano De Luca, Roberta Di Pace, Alfredo Massa, Roberto Schiattarella, and Autilia Vitiello. Application of quantum genetic algorithms to network signal setting design. In *2023 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2023.

[5] Giovanni Acampora, Ferdinando Di Martino, Alfredo Massa, Roberto Schiattarella, and Autilia Vitiello. D-nisq: a reference model for distributed noisy intermediate-scale quantum computers. *Information Fusion*, 89:16–28, 2023.

[6] Giovanni Acampora, Michele Grossi, Micheal Schenk, and Roberto Schiattarella. Quantum fuzzy inference engine for particle accelerators control. *Submitted to IEEE Transactions on Quantum Engineering.*

[7] Giovanni Acampora, Michele Grossi, and Roberto Schiattarella. A comparison of quantum computer architectures in running fuzzy inference engines. In *2023 IEEE International Conference on Fuzzy Systems (FUZZ)*, pages 1–6, 2023.

[8] Giovanni Acampora, Michele Grossi, and Autilia Vitiello. Genetic algorithms for error mitigation in quantum measurement. In *2021 IEEE congress on evolutionary computation (CEC)*, pages 1826–1832. IEEE, 2021.

[9] Giovanni Acampora, Federico Luongo, and Autilia Vitiello. Quantum implementation of fuzzy systems through grovers algorithm. In *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8. IEEE, 2018.

[10] Giovanni Acampora, Alfredo Massa, Roberto Schiattarella, and Autilia Vitiello. Distributing fuzzy inference engines on quantum computers. In *2023 IEEE International Conference on Fuzzy Systems (FUZZ)*, pages 1–6, 2023.

[11] Giovanni Acampora and Roberto Schiattarella. Deep neural networks for quantum circuit mapping. *Neural Computing and Applications*, 33(20):13723–13743, 2021.

[12] Giovanni Acampora and Roberto Schiattarella. On the effect of quantum noise in quantum genetic algorithms. In *2023 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2023.

[13] Giovanni Acampora, Roberto Schiattarella, and Alfredo Troiano. A dataset for quantum circuit mapping. *Data in Brief*, 39:107526, 2021.

[14] Giovanni Acampora, Roberto Schiattarella, and Autilia Vitiello. Quantum genetic selection: Using a quantum computer to select individuals in genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO ’21, page 219220, New York, NY, USA, 2021. Association for Computing Machinery.

[15] Giovanni Acampora, Roberto Schiattarella, and Autilia Vitiello. On the implementation of fuzzy inference engines on quantum computers. *IEEE Transactions on Fuzzy Systems*, 2022.

[16] Giovanni Acampora, Roberto Schiattarella, and Autilia Vitiello. Quantum mating operator: A new approach to evolve chromosomes in genetic algorithms. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2022.

[17] Giovanni Acampora, Roberto Schiattarella, and Autilia Vitiello. Using quantum amplitude amplification in genetic algorithms. *Expert Systems with Applications*, 209:118203, 2022.

[18] Giovanni Acampora and Autilia Vitiello. Error mitigation in quantum measurement through fuzzy c-means clustering. In *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2021.

[19] Giovanni Acampora and Autilia Vitiello. Implementing evolutionary optimization on actual quantum processors. *Information Sciences*, 575:542–562, 2021.

[20] Erik Adli, Arun Ahuja, O Apsimon, Robert Apsimon, A-M Bachmann, D Barrientos, Fabian Batsch, Jeremie Bauche, VK Berglyd Olsen, M Bernardini, et al. Acceleration of electrons in the plasma wakefield of a proton bunch. *Nature*, 561(7723):363–367, 2018.

[21] Diederik Aerts, Thomas Durt, and Bruno Van Bogaert. A physical example of quantum fuzzy sets and the classical limit. *Tatra Mt. Math. Publ*, 1:5–15, 1993.

[22] R Agustsson, P Carriere, O Chimalpopoca, V Dolgashev, MA Gusarova, SV Kutsaev, and A Yu Smirnov. Experimental studies of a high-gradient x-band welded hard-copper split accelerating structure. *Journal of Physics D: Applied Physics*, 55(14):145001, 2022.

[23] Dorit Aharonov, Wim Van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM review*, 50(4):755–787, 2008.

[24] Mohammad AlFailakawi, Imtiaz Ahmad, and Suha Hamdan. Lnn reversible circuit realization using fast harmony search based heuristic. In *Asia-Pacific Conference on Computer Science and Electrical Engineering*, 2014.

[25] Basemah Alshemali and Jugal Kalita. Improving the reliability of deep neural networks in nlp: A review. *Knowledge-Based Systems*, 191:105210, 2020.

[26] Farzan Aminian and Mehran Aminian. Fault diagnosis of analog circuits using bayesian neural networks with wavelet transform as preprocessor. *Journal of electronic testing*, 17(1):29–36, 2001.

[27] Farzan Aminian, Mehran Aminian, and HW Collins. Analog fault diagnosis of actual circuits using neural networks. *IEEE Transactions on Instrumentation and Measurement*, 51(3):544–550, 2002.

[28] M. Aminian and F. Aminian. A modular fault-diagnostic system for analog electronic circuits using neural networks with wavelet transform as a preprocessor. *IEEE Transactions on Instrumentation and Measurement*, 56(5):1546–1554, 2007.

[29] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[30] Anderson Ávila, Murilo Schmalfuss, Renata Reiser, and Vladik Kreinovich. Fuzzy xor classes from quantum computing. In *International Conference on Artificial Intelligence and Soft Computing*, pages 305–317. Springer, 2015.

[31] Safial Islam Ayon, Md Milon Islam, and Md Rahat Hossain. Coronary artery heart disease prediction: a comparative study of computational intelligence techniques. *IETE Journal of Research*, 68(4):2488–2507, 2022.

[32] Thomas Back. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the first IEEE conference on evolutionary computation. IEEE World Congress on Computational Intelligence*, pages 57–62. IEEE, 1994.

[33] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[34] James E Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, volume 206, pages 14–21, 1987.

[35] Enrique Ballinas and Oscar Montiel. Hybrid quantum genetic algorithm for the 0-1 knapsack problem in the ibm qiskit simulator. *Computación y Sistemas*, 26(2), 2022.

[36] Jagdish Chand Bansal and Kusum Deep. A modified binary particle swarm optimization for knapsack problems. *Applied Mathematics and Computation*, 218(22):11042–11061, 2012.

[37] Jagdish Chand Bansal, Pramod Kumar Singh, and Nikhil R Pal. *Evolutionary and swarm intelligence algorithms*. Springer, 2019.

[38] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[39] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20, 1993.

[40] Anirban Bhattacharjee, Chandan Bandyopadhyay, Robert Wille, Rolf Drechsler, and Hafizur Rahaman. A novel approach for nearest neighbor realization of 2d quantum circuits. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 305–310. IEEE, 2018.

[41] Debjyoti Bhattacharjee and Anupam Chattopadhyay. Depth-optimal quantum circuit placement for arbitrary topologies. *arXiv preprint arXiv:1703.08540*, 2017.

[42] Sergey Blinov, B Wu, and C Monroe. Comparison of cloud-based ion trap and superconducting quantum computer architectures. *AVS Quantum Science*, 3(3):033801, 2021.

[43] Kyle EC Booth, Minh Do, J Christopher Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. *arXiv preprint arXiv:1803.06775*, 2018.

[44] Max Born and Vladimir Fock. Beweis des adiabatensatzes. *Zeitschrift für Physik*, 51(3-4):165–180, 1928.

[45] Adi Botea, Akihiro Kishimoto, and Radu Marinescu. On the complexity of quantum circuit compilation. In *Eleventh Annual Symposium on Combinatorial Search*, 2018.

[46] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[47] Teresa Brecht, Wolfgang Pfaff, Chen Wang, Yiwen Chu, Luigi Frunzio, Michel H Devoret, and Robert J Schoelkopf. Multilayer microwave integrated quantum

circuits for scalable quantum computing. *npj Quantum Information*, 2(1):1–4, 2016.

[48] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, and John Schulman. Jie tang and wojciech zaremba. openai gym. *arXiv preprint arXiv*, 1606, 2016.

[49] Marcantonio Catelani and Marco Gori. On the application of neural networks to fault diagnosis of electronic analog circuits. *Measurement*, 17(2):73–80, 1996.

[50] Gangotree Chakma and Shaan Awasthi. Resource optimization for circuit simulation using machine learning. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4900–4905. IEEE, 2018.

[51] Amlan Chakrabarti, Susmita Sur-Kolay, and Ayan Chaudhury. Linear nearest neighbor synthesis of reversible circuits by graph partitioning. *arXiv preprint arXiv:1112.0564*, 2011.

[52] Mriganka Chakraborty. Artificial neural network for performance modeling and optimization of cmos analog circuits. *arXiv preprint arXiv:1212.0215*, 2012.

[53] Long Chen, Shaobo Lin, Xiankai Lu, Dongpu Cao, Hangbin Wu, Chi Guo, Chun Liu, and Fei-Yue Wang. Deep neural network based vehicle and pedestrian detection for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 22(6):3234–3246, 2021.

[54] Roshani Choudhary and Sanyam Shukla. A clustering based ensemble of weighted kernelized extreme learning machine for class imbalance learning. *Expert Systems with Applications*, 164:114041, 2021.

[55] Y Chung, G Decker, and K Evans. Closed orbit correction using singular value decomposition of the response matrix. In *Proceedings of International Conference on Particle Accelerators*, pages 2263–2265. IEEE, 1993.

[56] Pinar Civicioglu and Erkan Besdok. Bezier search differential evolution algorithm for numerical function optimization: A comparative study with crmlsp, mvo, wa, shade and lshade. *Expert Systems with Applications*, 165:113875, 2021.

[57] Elias F. Combarro and Samuel Gonzalez Castillo. *A Practical Guide to Quantum Machine Learning and Quantum Optimization.* <packt>, UK, March 2023.

[58] William J Conover and Ronald L Iman. Rank transformations as a bridge between parametric and nonparametric statistics. *The American Statistician*, 35(3):124–129, 1981.

[59] Toby S Cubitt, David Perez-Garcia, and Michael M Wolf. Undecidability of the spectral gap. *Nature*, 528(7581):207–211, 2015.

[60] Gian Luigi D'Alessandro, Johannes Bernhard, Alexander Gerbershagen, Laurent Gatignon, Markus Brugger, Stephen M Gibson, Francesco M Velotti, Niels Doble, Dipanwita Banerjee, and Bastien Rae. Jacow: Target bypass beam optics for future high intensity fixed target experiments in the cern north area. *JACoW IPAC*, 2021:3046–3048, 2021.

[61] Prasanna Date, Davis Arthur, and Lauren Pusey-Nazzaro. Qubo formulations for training machine learning models. *Scientific Reports*, 11(1):1–10, 2021.

[62] Kenneth De Jong. Evolutionary computation: a unified approach. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 185–199, 2016.

[63] Kalyanmoy Deb, Ram Bhushan Agrawal, et al. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.

[64] Shantanu Debnath, Norbert M Linke, Caroline Figgatt, Kevin A Landsman, Kevin Wright, and Christopher Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63–66, 2016.

[65] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[66] Francesco Di Colandrea, Lorenzo Amato, Roberto Schiattarella, Alexandre Dauphin, and Filippo Cardano. Retrieving unitary polarization transformations via optimized quantum tomography. *arXiv preprint arXiv:2210.17288*, 2022.

[67] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M Shir, and Thomas Bäck. Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing*, 88:106027, 2020.

[68] Didier Dubois and Henri Prade. Fuzzy sets in approximate reasoning, part 1: Inference with possibility distributions. *Fuzzy sets and systems*, 40(1):143–202, 1991.

[69] Agoston E Eiben, Zbigniew Michalewicz, Marc Schoenauer, and James E Smith. Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 19–46. Springer, 2007.

[70] Agoston E Eiben, P-E Raue, and Zs Ruttkay. Genetic algorithms with multi-parent recombination. In *International conference on parallel problem solving from nature*, pages 78–87. Springer, 1994.

[71] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing.* Springer, 2015.

[72] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[73] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[74] Dillion M Fox, Kim M Branson, and Ross C Walker. mrna codon optimization with quantum computers. *PloS one*, 16(10):e0259101, 2021.

[75] Giovanni Gallavotti. *Statistical mechanics: A short treatise*. Springer Science & Business Media, 1999.

[76] Salvador Garcia, Joaquin Derrac, Jose Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE transactions on pattern analysis and machine intelligence*, 34(3):417–435, 2012.

[77] Juan Carlos Garcia-Escartin and Pedro Chamorro-Posada. Equivalent quantum circuits. *arXiv preprint arXiv:1110.2998*, 2011.

[78] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier, 1991.

[79] John Grefenstette, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, volume 160, pages 160–168. Lawrence Erlbaum, 1985.

[80] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[81] Long GuiLu, Zhang WeiLin, Li YanSong, and Niu Li. Arbitrary phase rotation of the marked state cannot be used for grover's quantum search algorithm. *Communications in Theoretical Physics*, 32(3):335, 1999.

[82] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

[83] Jun He and Guangming Lin. Average convergence rate of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 20(2):316–321, 2015.

[84] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[85] Min Hou, Shibin Zhang, and Jinyue Xia. Quantum fuzzy k-means algorithm based on fuzzy theory. In *International Conference on Adaptive and Intelligent Systems*, pages 348–356. Springer, 2022.

[86] Scott Houchin J. Pendulum: Controlling an inverted pendulum using fuzzy logic. 1991.

[87] Peter Høyer. Arbitrary phases in quantum amplitude amplification. *Physical Review A*, 62(5):052304, 2000.

[88] Yo-Ping Huang, Pritpal Singh, Wen-Lin Kuo, and Hung-Chi Chu. A type-2 fuzzy clustering and quantum optimization approach for crops image segmentation. *International Journal of Fuzzy Systems*, 23(3):615–629, 2021.

[89] Travis S Humble, Himanshu Thapliyal, Edgard Munoz-Coreas, Fahd A Mohiyaddin, and Ryan S Bennink. Quantum computing circuits and devices. *IEEE Design & Test*, 36(3):69–94, 2019.

[90] Abid Hussain and Yousaf Shad Muhammad. Trade-off between exploration and exploitation with genetic algorithm using a novel selection operator. *Complex & intelligent systems*, pages 1–14, 2019.

[91] Shiro Ishikawa and Kohshi Kikuchi. Quantum fuzzy logic and time. *Journal of Applied Mathematics and Physics*, 9(11):2609–2622, 2021.

[92] Rene Jager. *Fuzzy logic in control.* Rene Jager, 1995.

[93] Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia. *Programming quantum computers: essential algorithms and code samples.* OReilly Media, Incorporated, 2019.

[94] Eric R Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia. *Programming Quantum Computers: essential algorithms and code samples.* " O'Reilly Media, Inc.", 2019.

[95] Ahmed Kafafy, Ahmed Bounekkar, and Stéphane Bonnevay. Hybrid metaheuristics based on moea/d for 0/1 multiobjective knapsack problems: A comparative study. In *2012 IEEE congress on evolutionary computation*, pages 1–8. IEEE, 2012.

[96] Phillip Kaye, Raymond Laflamme, Michele Mosca, et al. *An introduction to quantum computing.* Oxford university press, 2007.

[97] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[98] Morten Kjaergaard, Mollie E Schwartz, Jochen Braumüller, Philip Krantz, Joel I-J Wang, Simon Gustavsson, and William D Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11:369–395, 2020.

[99] Abhoy Kole, Kamalika Datta, and Indranil Sengupta. A heuristic for linear nearest neighbor realization of quantum circuits by swap gate insertion using n-gate lookahead. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(1):62–72, 2016.

[100] Abhoy Kole, Kamalika Datta, and Indranil Sengupta. A new heuristic for n-dimensional nearest neighbor realization of a quantum circuit. *IEEE Transactions*

on Computer-Aided Design of Integrated Circuits and Systems, 37(1):182–192, 2017.

[101] Zdenko Kovacic and Stjepan Bogdan. *Fuzzy controller design: theory and applications*. CRC press, 2018.

[102] Janusz Kusyk, Samah M Saeed, and Muharrem Umit Uyar. Survey on quantum circuit compilation for noisy intermediate-scale quantum computers: Artificial intelligence to heuristics. *IEEE Transactions on Quantum Engineering*, 2:1–16, 2021.

[103] Rafael Lahoz-Beltra. Quantum genetic algorithms for computer scientists. *Computers*, 5(4):24, 2016.

[104] P Martin Larsen. Industrial applications of fuzzy logic control. *International Journal of Man-Machine Studies*, 12(1):3–10, 1980.

[105] Roberto Leporini, Cesarino Bertini, and Filippo Carone Fabiani. Fuzzy representation of finite-valued quantum gates. *Soft Computing*, 24(14):10305–10313, 2020.

[106] B. Li and P. D. Franzon. Machine learning in physical design. In *2016 IEEE 25th Conference on Electrical Performance Of Electronic Packaging And Systems (EPEPS)*, pages 147–150, 2016.

[107] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019.

[108] Junyao Li and Zhinan Hao. A quantum probabilistic linguistic term framework to multi-attribute decision-making for battlefield situation assessment. *International Journal of Fuzzy Systems*, 24(1):495–507, 2022.

[109] Xun Li, Lishui Chen, and Yazhe Tang. Hard: Bit-split string matching using a heuristic algorithm to reduce memory demand. *Rom. J. Inf. Sci. Technol*, 23:T94–T105, 2020.

[110] Y. Li, Y. Lin, M. Madhusudan, A. Sharma, W. Xu, S. Sapatnekar, R. Harjani, and J. Hu. Exploring a machine learning approach to performance driven analog ic placement. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 24–29, 2020.

[111] Benjamin Lienhard, Antti Vepsäläinen, Luke CG Govia, Cole R Hoffer, Jack Y Qiu, Diego Ristè, Matthew Ware, David Kim, Roni Winik, Alexander Melville, et al. Deep-neural-network discrimination of multiplexed superconducting-qubit states. *Physical Review Applied*, 17(1):014024, 2022.

[112] Chia-Chun Lin, Susmita Sur-Kolay, and Niraj K Jha. Paqcs: Physical design-aware fault-tolerant quantum circuit synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(7):1221–1234, 2014.

[113] Norbert M Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

[114] Sean Luke. *Essentials of metaheuristics*, volume 2. Lulu Raleigh, 2013.

[115] Aaron Lye, Robert Wille, and Rolf Drechsler. Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits. In *The 20th Asia and South Pacific Design Automation Conference*, pages 178–183. IEEE, 2015.

[116] Andrea Malossini, Enrico Blanzieri, and Tommaso Calarco. Quantum genetic optimization. *IEEE Transactions on Evolutionary Computation*, 12(2):231–241, 2008.

[117] Ebrahim H Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the institution of electrical engineers*, volume 121, pages 1585–1588. IET, 1974.

[118] Dmitri Maslov, Sean M Falconer, and Michele Mosca. Quantum circuit placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(4):752–763, 2008.

[119] Elisha Siddiqui Matekole, Yao-Lung Leo Fang, and Meifeng Lin. Methods and results for quantum optimal pulse control on superconducting qubit systems. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 600–606, 2022.

[120] WS McCelloch and Walter Pitts. A logical calculus of the idea immanent in neural nets. *Bulletin ofMathematical Biophysics*, 5:115–133, 1943.

[121] Melanie Mitchell. *An introduction to genetic algorithms.* MIT press, 1998.

[122] Thomas B Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Computer vision and image understanding*, 81(3):231–268, 2001.

[123] Klaus Mølmer and Anders Sørensen. Multiparticle entanglement of hot trapped ions. *Physical Review Letters*, 82(9):1835, 1999.

[124] Christopher Monroe, Robert Raussendorf, Alex Ruthven, Kenneth R Brown, Peter Maunz, L-M Duan, and Jungsang Kim. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Physical Review A*, 89(2):022317, 2014.

[125] Gary J Mooney, Charles D Hill, and Lloyd CL Hollenberg. Entanglement in a 20-qubit superconducting quantum computer. *Scientific reports*, 9(1):1–8, 2019.

[126] Michele Mosca et al. Quantum searching, counting and amplitude amplification by eigenvector analysis. In *MFCS98 workshop on Randomized Algorithms*, pages 90–100, 1998.

[127] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1015–1029, 2019.

[128] Ajit Narayanan and Mark Moore. Quantum-inspired genetic algorithms. In *Proceedings of IEEE international conference on evolutionary computation*, pages 61–66. IEEE, 1996.

[129] Hendrik Poulsen Nautrup, Nicolas Delfosse, Vedran Dunjko, Hans J Briegel, and Nicolai Friis. Optimizing quantum error correction codes with reinforcement learning. *Quantum*, 3:215, 2019.

[130] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011.

[131] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[132] Angelo Oddi and Riccardo Rasconi. Greedy randomized search for scalable compilation of quantum circuits. In *International Conference on the Integration of*

Constraint Programming, Artificial Intelligence, and Operations Research, pages 446–461. Springer, 2018.

[133] Nissim Ofek, Andrei Petrenko, Reinier Heeres, Philip Reinhold, Zaki Leghtas, Brian Vlastakis, Yehan Liu, Luigi Frunzio, SM Girvin, Liang Jiang, et al. Extending the lifetime of a quantum bit with error correction in superconducting circuits. *Nature*, 536(7617):441–445, 2016.

[134] Ramon WJ Overwater, Masoud Babaie, and Fabio Sebastiano. Neural-network decoders for quantum error correction using surface codes: A space exploration of the hardware cost-performance tradeoffs. *IEEE Transactions on Quantum Engineering*, 3:1–19, 2022.

[135] Alexandru Paler. On the influence of initial qubit placement during nisq circuit compilation. In *International Workshop on Quantum Technology and Optimization Problems*, pages 207–217. Springer, 2019.

[136] Hari Mohan Pandey, Ankit Chaudhary, and Deepti Mehrotra. A comparative review of approaches to prevent premature convergence in ga. *Applied Soft Computing*, 24:1047–1077, 2014.

[137] Nirali Patel. Review on machine learning for analog circuit design. *International Journal of Engineering and Technical Research*, 9:1024–1028, 05 2020.

[138] David Pollard. *A user's guide to measure theoretic probability*. Number 8. Cambridge University Press, 2002.

[139] Amir Pourabdollah, Giovanni Acampora, and Roberto Schiattarella. Fuzzy logic on quantum annealers. *IEEE Transactions on Fuzzy Systems*, 30(8):3389–3394, 2021.

[140] Amir Pourabdollah, Giovanni Acampora, and Roberto Schiattarella. Implementing defuzzification operators on quantum annealers. In *2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2022.

[141] Amir Pourabdollah, Colin Wilmott, Roberto Schiattarella, and Giovanni Acampora. Fuzzy inference on quantum annealers. In *2023 IEEE International Conference on Fuzzy Systems (FUZZ)*, pages 1–6, 2023.

[142] Claudiu Pozna, Radu-Emil Precup, Erno Horvath, and Emil M Petriu. Hybrid particle filter-particle swarm optimization algorithm and application to fuzzy controlled servo systems. *IEEE Transactions on Fuzzy Systems*, 2022.

[143] Radu-Emil Precup, Radu-Codruţ David, Emil M Petriu, Stefan Preitl, and Adrian Sebastian Paul. Gravitational search algorithm-based tuning of fuzzy control systems with a reduced parametric sensitivity. In *Soft computing in industrial applications*, pages 141–150. Springer, 2011.

[144] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[145] Francesco Preti, Tommaso Calarco, and Felix Motzoi. Continuous quantum gate sets and pulse-class meta-optimization. *PRX Quantum*, 3(4):040311, 2022.

[146] Weiyi Qi et al. *IC Design Analysis, Optimization and Reuse via Machine Learning*. PhD thesis, North Carolina State University, 2017.

[147] Riccardo Rasconi and Angelo Oddi. An innovative genetic algorithm for the quantum circuit compilation problem. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7707–7714, 2019.

[148] Renata Reiser, Alexandre Lemke, Anderson Avila, Júlia Vieira, Maurício Pilla, and André Du Bois. Interpretations on quantum fuzzy computing: intuitionistic

fuzzy operations× quantum operators. *Electronic Notes in Theoretical Computer Science*, 324:135–150, 2016.

[149] Gerasimos G Rigatos and Spyros G Tzafestas. Parallelization of a fuzzy control algorithm using quantum computation. *IEEE Transactions on Fuzzy Systems*, 10(4):451–460, 2002.

[150] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[151] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[152] Rosa Joao P. S., Daniel J. D. Guerra, Nuno C. G. Horta, Ricardo Martins, and Lourenco Nuno C. C. *Using artificial neural networks for analog integrated circuit design automation*. Springer, 2020.

[153] Mehdi Saeedi, Robert Wille, and Rolf Drechsler. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quantum Information Processing*, 10(3):355–377, 2011.

[154] Harvey M Salkin and Cornelis A De Kluyver. The knapsack problem: a survey. *Naval Research Logistics Quarterly*, 22(1):127–144, 1975.

[155] Michael Schenk, Elías F Combarro, Michele Grossi, Verena Kain, Kevin Shing Bruce Li, Mircea-Marian Popa, and Sofia Vallecorsa. Hybrid actor-critic algorithm for quantum reinforcement learning at cern beam lines. *arXiv preprint arXiv:2209.11044*, 2022.

[156] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2013.

[157] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Qubit placement to minimize communication overhead in 2d quantum architectures. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 495–500. IEEE, 2014.

[158] Vivek V Shende, Stephen S Bullock, and Igor L Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010, 2006.

[159] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):14841509, Oct 1997.

[160] Ritu Ranjan Shrivastwa, Kamalika Datta, and Indranil Sengupta. Fast qubit placement in 2d architecture using nearest neighbor realization. In *2015 ieee international symposium on nanoelectronic and information systems*, pages 95–100. IEEE, 2015.

[161] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[162] William M Spears and Kenneth A De Jong. An analysis of multi-point crossover. In *Foundations of genetic algorithms*, volume 1, pages 301–315. Elsevier, 1991.

[163] Hugh Spence. Automatic analog fault simulation. In *Conference Record. AUTOTESTCON'96*, pages 17–22. IEEE, 1996.

[164] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[165] Gilbert Syswerda et al. Uniform crossover in genetic algorithms. In *ICGA*, volume 3, pages 2–9, 1989.

[166] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on systems, man, and cybernetics*, pages 116–132, 1985.

[167] Garry Wei-Han Tan, Keng-Boon Ooi, Lai-Ying Leong, and Binshan Lin. Predicting the drivers of behavioral intention to use mobile learning: A hybrid sem-neural networks approach. *Computers in Human Behavior*, 36:198–213, 2014.

[168] Ankit Thakkar and Kinjal Chaudhari. A comprehensive survey on deep neural networks for stock market: The need, challenges, and future directions. *Expert Systems with Applications*, 177:114800, 2021.

[169] Matthew P Thompson, Jeff D Hamann, and John Sessions. Selection and penalty strategies for genetic algorithms designed to solve spatial forest planning problems. *International Journal of Forestry Research*, 2009, 2009.

[170] Anant J Umbarkar and Pranali D Sheth. Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6(1), 2015.

[171] Christine L Valenzuela. A simple evolutionary algorithm for multi-objective optimization (seamo). In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 1, pages 717–722. IEEE, 2002.

[172] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology*, 3(2):025004, 2018.

[173] Davide Venturelli, Minh Do, Eleanor Gilbert Rieffel, and Jeremy Frank. Temporal planning for compilation of quantum approximate optimization circuits. In *IJCAI*, pages 4440–4446, 2017.

[174] Lidiane Visintin, Adriano Maron, Renata Reiser, and Vladik Kreinovich. Aggregation operations from quantum computing. In *2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8. IEEE, 2013.

[175] Feng Wang and Yuan-xiang Li. Analog circuit design automation using neural network-based two-level genetic programming. In *2006 International Conference on Machine Learning and Cybernetics*, pages 2087–2092. IEEE, 2006.

[176] L.-X. Wang. Fuzzy systems are universal approximators. In *[1992 Proceedings] IEEE International Conference on Fuzzy Systems*, pages 1163–1170, 1992.

[177] Li-Xin Wang, Jerry M Mendel, et al. Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. *IEEE transactions on Neural Networks*, 3(5):807–814, 1992.

[178] Darrell Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, page 116121, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[179] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[180] Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits. In *2016 21st Asia and South Pacific design automation conference (ASP-DAC)*, pages 292–297. IEEE, 2016.

[181] Robert Wille, Aaron Lye, and Rolf Drechsler. Optimal swap gate insertion for nearest neighbor quantum circuits. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 489–494. IEEE, 2014.

[182] David J Wineland, Christopher Monroe, Wayne M Itano, Dietrich Leibfried, Brian E King, and Dawn M Meekhof. Experimental issues in coherent quantum-state manipulation of trapped atomic ions. *Journal of research of the National Institute of Standards and Technology*, 103(3):259, 1998.

[183] Xin-She Yang. Chapter 6 - genetic algorithms. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms (Second Edition)*, pages 91–100. Academic Press, second edition edition, 2021.

[184] J-S Yih and Pinaki Mazumder. A neural network design for circuit partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(12):1265–1271, 1990.

[185] EL Yu and Ponnuthurai N Suganthan. Ensemble of niching algorithms. *Information Sciences*, 180(15):2815–2833, 2010.

[186] L.A. Zadeh. Fuzzy logic. *Computer*, 21(4):83–93, 1988.

[187] Lofti A Zadeh. Fuzzy sets,information and control, 1965, vol. 8, 338–353. *Fuzzy Algorithms,Information and Control*, 12:94–102, 1968.

[188] Lotfi A Zadeh. Fuzzy sets. In *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh*, pages 394–432. World Scientific, 1996.

[189] Christof Zalka. A grover-based quantum search of optimal order for an unknown number of marked elements. *arXiv preprint quant-ph/9902049*, 1999.

[190] Iuliu Alexandru Zamfirache, Radu-Emil Precup, Raul-Cristian Roman, and Emil M Petriu. Policy iteration reinforcement learning-based control using a grey wolf optimizer algorithm. *Information Sciences*, 585:162–175, 2022.

[191] Shah Zeb, Aamir Mahmood, Syed Ali Hassan, MD Jalil Piran, Mikael Gidlund, and Mohsen Guizani. Industrial digital twins at the nexus of nextg wireless

networks and computational intelligence: A survey. *Journal of Network and Computer Applications*, 200:103309, 2022.

[192] Gexiang Zhang, Weidong Jin, and Laizhao Hu. A novel parallel quantum genetic algorithm. In *Proceedings of the fourth international conference on parallel and distributed computing, applications and technologies*, pages 693–697. IEEE, 2003.

[193] Yazhou Zhang, Yaochen Liu, Qiuchi Li, Prayag Tiwari, Benyou Wang, Yuhua Li, Hari Mohan Pandey, Peng Zhang, and Dawei Song. Cfn: a complex-valued fuzzy network for sarcasm detection in conversations. *IEEE Transactions on Fuzzy Systems*, 29(12):3696–3710, 2021.

[194] Yujun Zheng, Xueqin Lu, Minxia Zhang, and Shengyong Chen. *Optimization Problems and Algorithms*, pages 1–25. Springer Singapore, Singapore, 2019.