Department of Information Engineering and Computer Science

Doctoral Programme in
Information and Communication Technology

Department of Physics

Transdisciplinary Program in
Quantum Science and Technologies

Final Dissertation

Hybrid classical-quantum algorithms for optimization and machine learning

Advisor
Enrico Blanzieri

Student
Enrico Zardini

Co-Advisors
Davide Pastorello
Valter Moretti

Academic year 2022/2023

Acknowledgements

First of all, I would like to thank my supervisors, namely, Professors Enrico Blanzieri, Davide Pastorello, and Valter Moretti, for guiding me in this journey. I would also like to thank Professor Gabriele Cavallaro for hosting me at Forschungszentrum Jülich during my research period abroad and the reviewers of this thesis for their precious feedback.

Then, I would like to thank my family, namely, my mother Marina, my father Michele, and my brother Luca, for always supporting and motivating me. In addition, I would like to express my gratitude to my grandmother Pia and all my relatives.

Last but not least, I would like to thank my friends and colleagues for being present in these years.

This thesis was supported by Q@TN, the joint lab between University of Trento, FBK-Fondazione Bruno Kessler, INFN-National Institute for Nuclear Physics and CNR-National Research Council. In addition, the author gratefully acknowledges the Jülich Supercomputing Center (`https://www.fz-juelich.de/ias/jsc`) for providing computing time on D-Wave quantum annealers and, specifically, on the D-Wave Advantage™ System JUPSI through the Jülich UNified Infrastructure for Quantum computing (JUNIQ). Lastly, the author gratefully acknowledges the Italian Ministry of University and Research (MUR), which, under the initiative "Dipartimenti di Eccellenza 2018-2022 (Legge 232/2016)", has provided computational resources used in the experiments.

Contents

Abstract

Quantum computing is a form of computation that exploits quantum mechanical phenomena for information processing, with promising applications (among others) in optimization and machine learning. Indeed, quantum machine learning is currently one of the most popular directions of research in quantum computing, offering solutions with an at-least-theoretical advantage compared to the classical counterparts. Nevertheless, the quantum devices available in the current Noisy Intermediate-Scale Quantum (NISQ) era are limited in the number of qubits and significantly affected by noise. An interesting alternative to the current prototypes of general-purpose quantum devices is represented by quantum annealers, specific-purpose quantum machines implementing the heuristic search for solving optimization problems known as quantum annealing. However, despite the higher number of qubits, the current quantum annealers are characterised by very sparse topologies. These practical issues have led to the development of hybrid classical-quantum schemes, aiming at leveraging the strengths of both paradigms while circumventing some of the limitations of the available devices. In this thesis, several hybrid classical-quantum algorithms for optimization and machine learning are introduced and/or empirically assessed, as the empirical evaluation is a fundamental part of algorithmic research. The quantum computing models taken into account are both quantum annealing and circuit-based universal quantum computing. The results obtained have shown the effectiveness of most of the proposed approaches.

Keywords: hybrid classical-quantum computing, quantum annealing, quantum machine learning, optimization, locality, empirical evaluation

1 Introduction

Quantum computing is a type of computation leveraging quantum mechanical phenomena for information processing. In particular, quantum computing has the potentiality to efficiently solve optimization and machine learning problems, which are often computationally intensive. However, in the current Noisy Intermediate-Scale Quantum (NISQ) era [87], the quantum computers available in the market [45, 91] are limited in the number of qubits and significantly affected by noise, restricting the problems that can be addressed in practice.

An interesting alternative to universal quantum computing is represented by quantum annealing. In detail, quantum annealing is a heuristic search aimed at solving optimization problems by probabilistically identifying the low-energy states of a quantum system [59], and quantum annealers are non-universal specific-purpose quantum devices that implement quantum annealing. Compared to the available general-purpose quantum computers, the quantum annealers provided by D-Wave [49] are characterised by a higher number of qubits (thousands versus hundreds). This has fostered expectations that, for specific problem domains, quantum annealers could potentially outperform the classical counterparts. A promising application is, for instance, the one described by Bian et al. in their work [8]. Nevertheless, the topology of these machines is far from being complete (namely, it is very sparse), and quantum annealers are also subject to noise.

In the area of quantum computing, one of the most popular directions of research is represented by Quantum Machine Learning (QML). Indeed, in the last decade, numerous interesting QML algorithms have been introduced and theoretically characterised, with occasional empirical validation. Notable examples are the quantum support vector machine introduced by Rebentrost et al. [89], distance-based classifiers as the one presented by Schuld et al. [98], and quantum neural networks, whose performance have been analysed by Abbas et al. [1]. However, quantum annealing has also been taken into account in the development of quantum machine learning algorithms [74, 113, 118]. In particular, the combination of quantum computation and machine learning offers solutions characterised by an at-least-theoretical advantage compared to the classical counterparts. Moreover, QML seems a good way to leverage the existing NISQ devices for addressing real-world problems.

The desire of practically using quantum computing has given rise to hybrid classical-quantum approaches, delegating part of the computation to classical devices. Indeed, the current quantum computing architectures are still immature, and hybrid schemes can effectively circumvent some of their limitations. Additionally, it is reasonable to assume that the interplay between classical and quantum procedures allows leveraging the strengths of both paradigms, despite the inefficient interface between them. Specifically, hybrid classical-quantum approaches have been presented for both universal quantum devices [69] and quantum annealing architectures [4]. Moreover, in the field of QML, this represents a compelling alternative to the development of quantum algorithms capable of fully solving machine learning tasks but requiring ideal devices.

In this thesis, several hybrid classical-quantum algorithms for optimization and machine learning are introduced and/or empirically evaluated. Indeed, the empirical evaluation is an essential part of algorithmic research and, although established benchmarks do not yet exist for QML, a systematic evaluation is fundamental in order to progress. The thesis is structured as follows. Chapters 2 and 3 provide useful background information for understanding the subsequent chapters. Specifically, Chapter 2 deals with miscellaneous topics, while Chapter 3 deals with quantum machine learning only. Regarding the novel content, Part I, which includes Chapters 4 to 6, is devoted to quantum annealing algorithms, while Part II, which includes Chapters 7 and 8, is devoted to algorithms for (circuit-based) universal quantum devices. In detail, Chapter 4, which is based on the article "Quantum annealing learning search implementations" [12], presents two implementations and the empirical evaluation of QALS [81], a hybrid algorithm for tackling optimization problems that cannot be directly mapped on a quantum annealer. The article in question was motivated by the absence of an empirical assessment of the algorithm in the literature. Instead, Chapter 5, which is based on the article “Reconstructing Bayesian networks on a quantum annealer” [130], presents the implementation and the empirical evaluation of the QUBO formulation proposed by O’Gorman et al. [76] for reconstructing Bayesian networks on the same architecture; a *divide et impera* approach for tackling larger problem instances is also introduced and assessed in the same chapter. This article, as well, was motivated by the absence of an empirical assessment of the QUBO formulation in the literature. In Chapter 6, which is based on the article “Local Binary and Multiclass SVMs Trained on a Quantum Annealer” [129], the local application of support vector machines trained on a quantum annealer is introduced and empirically evaluated. In this case, the entirely classical counterpart had already proven successful [9, 104], and the considered quantum-trained models [26, 118] suffered from the limited connectivity of the available quantum annealers, motivating the article in question. Concerning Chapter 7, which is based on the article “Implementation and empirical evaluation of a quantum machine learning pipeline for local classification” [128], it introduces the application of a quantum locality technique, such as a quantum k-nearest neighbors algorithm, as a preliminary step of other quantum machine learning models; the empirical assessment of the approach is also illustrated. This article was motivated by the absence of quantum local classification pipelines in the literature, a successful approach in the classical realm (as said for the previous chapter) that may also allow saving qubits in the quantum one. Lastly, Chapter 8, which is based on the article “A quantum k-nearest neighbors algorithm based on the Euclidean distance estimation” [127], introduces a novel quantum adaptation of the k-nearest neighbors algorithm (based on the Euclidean distance) and its empirical evaluation. The article in question was motivated by the absence of a quantum version of the algorithm utilizing the Euclidean distance distance metric in the literature. The main results presented in these chapters are summarised in Chapter 9, which also briefly describes some possible future directions of research.

2 Background

This chapter provides some background information about quantum computing and the topics covered in the first part of this work (Part I). In particular, this chapter is a reworked version of different parts of the background sections of various articles [12, 128–130].

2.1 Quantum Computing

Quantum computing is a form of computation that leverages quantum phenomena, such as state superposition and entanglement. This field represents a significant application of quantum information theory, delivering algorithms to efficiently solve problems that are challenging for classical computers [75]. In this work, two different kinds of quantum computation have been exploited, namely, quantum annealing and quantum circuits.

2.1.1 Quantum Annealing

Quantum Annealing (QA) is a heuristic search method for solving optimization problems [59]. In this approach, the solution to a given problem corresponds to the ground state, being the least energetic physical state, of an n-qubit system. The system's energy is described by a problem Hamiltonian H_P, which is a $2^n \times 2^n$ Hermitian matrix. In particular, the annealing process involves the quantum system's time evolution towards the ground state of the problem Hamiltonian. More in detail, let us consider the time-dependent Hamiltonian

$$H(t) = \Gamma(t) H_D + H_P, \tag{2.1}$$

where H_P denotes the problem Hamiltonian and H_D represents the transverse field Hamiltonian. Specifically, H_D provides the kinetic term, enabling the exploration of the solution landscape through quantum fluctuations. Instead, Γ represents a decreasing function that reduces the kinetic term, guiding the system towards the global minimum of the problem (represented by H_P).

QA can be physically implemented by taking into account a quantum spin glass, which is a network of qubits organized on the vertices of a graph $\langle V, E\rangle$, where $|V| = n$, and the edges E correspond to the couplings among these qubits. The problem Hamiltonian is defined as

$$H_P = H(\Theta) = \sum_{i \in V} \theta_i \sigma_z^{(i)} + \sum_{(i,j)\in E} \theta_{ij} \sigma_z^{(i)} \sigma_z^{(j)}, \tag{2.2}$$

with the real coefficients θ_i, θ_{ij} being organized into the matrix Θ. In detail, $H(\Theta)$ is an operator on the n-qubit Hilbert space $\mathsf{H} = (\mathbb{C}^2)^{\otimes n}$, while $\sigma_z^{(i)}$ operates as the Pauli matrix

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.3}$$

on the i-th tensor factor and as the 2×2 identity matrix on the other tensor factors. Concerning the coefficient matrix Θ, it is an $n \times n$ symmetric square matrix that contains real values (known as *weights*) and is defined as

$$\Theta_{ij} := \begin{cases} \theta_i, & i = j, \\ \theta_{ij}, & (i,j) \in E, \\ 0, & (i,j) \notin E, \end{cases} \tag{2.4}$$

where θ_i physically corresponds to the local field on the i-th qubit, and θ_{ij} to the coupling between the qubits i and j. Specifically, the Pauli matrix σ_z has two eigenvalues, $\{-1, 1\}$, corresponding to the binary states of each qubit, i.e., *spin down* and *spin up*. Consequently, the spectrum of eigenvalues of the problem Hamiltonian (Equation 2.2) includes all the potential values of the energy function of the well-known *Ising model*:

$$\mathsf{E}(\Theta, \mathbf{z}) = \sum_{i \in V} \theta_i z_i + \sum_{(i,j) \in E} \theta_{ij} z_i z_j, \quad \mathbf{z} = (z_1, ..., z_n) \in \{-1, 1\}^{|V|}. \tag{2.5}$$

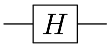
In practice, the annealing procedure, often referred to as *cooling*, guides the system towards the ground state of $H(\Theta)$. This state corresponds to the spin configuration that encodes the solution to the problem, namely,

$$\mathbf{z}^* = \underset{\mathbf{z} \in \{-1,1\}^{|V|}}{\text{argmin}} \ \mathsf{E}(\Theta, \mathbf{z}). \tag{2.6}$$

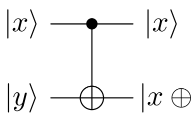
When given a problem, the annealer is initialized using appropriate weights Θ, and the binary variables $z_i \in \{-1, 1\}$ are physically realized through measurements conducted on the qubits positioned at the vertices V. In order to address a general optimization problem via QA, it is essential to determine a suitable *encoding* of the objective function in relation to the cost function (2.5), a task that is generally challenging.

2.1.2 Quantum Circuit Model

Among the different models of universal quantum computation, the most common one is the quantum circuit model. In this framework, the primary element of quantum computation is the qubit, a two-level physical system whose state is described by a unit vector $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$ in a two-dimensional complex Hilbert space, where $|0\rangle$ and $|1\rangle$ constitute an orthonormal basis. Here, $|\ \rangle$ represents a *ket* in the Dirac notation, used for denoting quantum states, and $|0\rangle$ and $|1\rangle$ correspond to the vectors of the standard basis in $\mathbb{C}^2$. The absolute squares of the complex amplitudes α and β represent the probabilities of measuring the qubit in states 0 and 1, respectively, satisfying the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. Following a measurement process, the qubit's state collapses to either $|0\rangle$ or $|1\rangle$, depending on the observed outcome. Moreover, the time evolution of isolated quantum systems, like qubits, is described by unitary operators, which are referred to as quantum gates in the realm of quantum computation. For instance, the Hadamard gate, which acts on a single qubit, can be defined through the equations $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$, where $|\pm\rangle = 1/\sqrt{2} \cdot (|0\rangle \pm |1\rangle)$. Essentially, the Hadamard gate creates a superposition state; its matrix representation and circuital symbol are

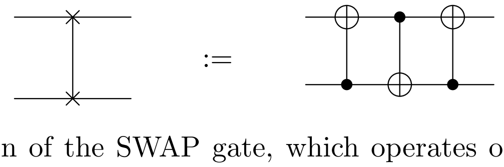
$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$ and .

Another fundamental quantum gate is the Controlled-NOT (CNOT) gate, which acts on two qubits and works as follows with respect to the computational basis:



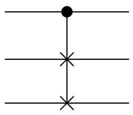
,

where $x, y \in \{0, 1\}$ and $\oplus$ is the sum modulo 2. In practical terms, the CNOT gate flips the state of the target qubit only if the control qubit is in state $|1\rangle$. Notably, by utilizing three CNOT gates in combination, it becomes feasible to construct the SWAP gate, a 2-qubit gate designed to exchange the input qubits, whose circuital definition is



.

Instead, the controlled version of the SWAP gate, which operates on three qubits, is referred to as the Fredkin gate. Its classical counterpart is universal for classical reversible computation, and its circuital symbol is



.

According to a fundamental axiom of quantum mechanics, a system of n qubits is mathematically described in the space $(\mathbb{C}^2)^{\otimes n}$. Consequently, the space available for representing data increases exponentially with the number of qubits. This exponential growth constitutes one of the key advantages of quantum computation. In practical terms, quantum algorithms are crafted by combining the available quantum gates in order to generate a quantum state encoding the solution to a problem of interest. Then, the outcome is obtained by measuring the output state in the computational basis. Given the probabilistic nature of the results of quantum computation, quantum algorithms typically need to be repeated multiple times to provide meaningful outcomes. A significant demonstration of the efficiency of quantum computation lies in Shor's algorithm [107], whose time complexity for solving the integer factoring problem, a task generally suspected to be beyond $\boldsymbol{P}$, is polynomial.

2.2 Quadratic Unconstrained Binary Optimization

Quadratic Unconstrained Binary Optimization (QUBO) problems are optimization problems expressed in the form

$$\underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{x}^T Q \mathbf{x}, \tag{2.7}$$

with $\mathbf{x}$ being a binary vector, and Q being an upper triangular (or symmetric[1]) matrix composed of real values. Specifically, let $\mathbf{x}$ be an $n \times 1$ vector and Q an $n \times n$ upper triangular matrix. Then, it is possible to reformulate the problem as

$$\begin{aligned} \mathbf{x}^T Q \mathbf{x} &= \sum_{i=1}^{n} q_{ii} x_i^2 + \sum_{i=1}^{n} \sum_{j=i+1}^{n} q_{ij} x_i x_j \\ &= \sum_{i=1}^{n} q_{ii} x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} q_{ij} x_i x_j, \end{aligned} \tag{2.8}$$

where $x_i^2 = x_i$ due to $x_i \in \mathbb{B} = \{0, 1\}$. Basically, the main diagonal of Q contains the linear coefficients (q_{ii}), while the other cells of the matrix contain the quadratic coefficients (q_{ij}). Although QUBO problems are inherently unconstrained, it is possible to introduce constraints as penalties. Glover et al. [38] provide several examples of this approach.

The relevance of the QUBO formulation consists in being computationally equivalent to the Ising model, which is the physical model on which annealers are based. The only distinction consists in the domain of the variables: $\{0, 1\}$ for the QUBO formulation and $\{-1, +1\}$ for the Ising one. Consequently, a simple conversion allows the usage of quantum annealing to solve problems expressed as QUBO, such as optimization problems on graphs, clustering problems, set partitioning problems, or sequencing and ordering problems [38].

2.2.1 D-Wave Annealers, Embedding, and Hybrid Solvers

D-Wave Systems [49] is a Canadian company specialized in producing quantum annealers, namely, physical machines that implement the quantum annealing process. Currently, their flagship model is the D-Wave Advantage, which adopts the *Pegasus* topology. This model is characterised by 5640 qubits, with each qubit connected to 15 other qubits. While a greater number of qubits enables the processing of larger problems, the most significant aspect is the connectivity, as it determines the complexity of problems that can be effectively represented and solved.

A crucial step, in order to exploit quantum annealing for solving QUBO problems, consists in mapping the problem variables to the Quantum Processing Unit (QPU) qubits. However, due to the

[1] If the matrix Q is symmetric, it is sufficient to sum the corresponding elements outside the main diagonal in order to obtain an upper triangular matrix.

sparseness of the annealer topology, a direct representation is often not feasible. The solution lies in chaining together multiple physical qubits to operate as a single logical qubit. In this way, the connectivity of the annealer graph is enhanced, although at the expense of reducing the number of available qubits and, consequently, limiting the size of the problems that can be effectively represented. This entire process is commonly referred to as embedding or *minor embedding*, in the glossary of D-Wave [52, 53]. Specifically, D-Wave's Ocean library offers the *EmbeddingComposite* class [50] to automate the minor embedding of the QUBO matrices.

In addition, as an alternative to simulated and quantum annealing, D-Wave provides the so-called *hybrid* annealing (just *Hybrid* from now on) [51], a framework that enables the simultaneous execution of multiple solvers, whether classical or quantum. In detail, a branch could correspond to a classical technique like Tabu Search or to a workflow involving a *decomposer* - *sampler* - *composer* structure. In the latter case, the decomposer divides the given problem into subproblems, each solved by the sampler. The local solutions obtained are then recomposed by the composer to produce the final solution. Notably, the sampler component could be Simulated Annealing (SA), quantum annealing, other classical techniques, or more complex methods.

2.3 Quantum Annealing Learning Search

Quantum Annealing Learning Search (QALS) is a guided meta-heuristic approach specifically devised to tackle optimization problems that cannot be directly mapped onto the architecture of a quantum annealer. The core idea of this approach, initially proposed in [81] and further elaborated in [82], consists in initializing the quantum annealer by summing a *tabu matrix* S to the weight matrix Θ. The role of the S matrix is to penalize previously explored solutions, thus preventing redundant searches in the solution space. Suppose that we have a set of k solutions $\{\mathbf{z}_j\}_{j=1,\dots,k}$ to be penalized. The matrix S is built as follows:

$$S = \sum_{j=1}^{k} (\mathbf{z}_j \mathbf{z}_j^T - I + \operatorname{diag} \mathbf{z}_j), \tag{2.9}$$

with I being the identity matrix of size n, and $\operatorname{diag} \mathbf{z}_j$ being the diagonal matrix resulting from the vector $\mathbf{z}_j$. By design, the tabu matrix S imposes *energetic penalties* on the solutions ${\mathbf{z}_j}_{j=1,\dots,k}$ in the spectrum of the Hamiltonian $H(\Theta + S)$, which corresponds to the energy function $\mathbf{z} \mapsto \mathsf{E}(\Theta + S, \mathbf{z})$.

Basically, QALS operates through an iterative process involving the generation of candidate solutions via QA and subsequent probabilistic acceptance or rejection. New candidate solutions are created by perturbing the weights according to the approach presented below, while rejected solutions contribute to the tabu matrix S. In addition, similarly to SA, the algorithm allows for suboptimal acceptance of solutions and a decreasing temperature parameter is used to control the weight perturbation. Concerning the problem representation into the annealer, the matrix Q representing the objective quadratic function f_Q is deformed by means of S and mapped onto the annealer architecture in a piecewise manner, which means that only specific elements of the QUBO matrix are chosen at each iteration. More precisely, the mapping μ employed for solving QUBO problems is defined as follows:

$$\mu[f_Q](\mathbf{z}) = \mathsf{E}(P^T(Q + \lambda S)P \circ \mathcal{A}, P\mathbf{z}), \tag{2.10}$$

where P denotes a permutation matrix of order n and P^T represents its transpose, λ is a scaling factor that modulates the contribution of the tabu matrix S (initially set to zero), $\mathcal{A}$ is the adjacency matrix of the topology graph of the quantum annealer, and $\circ$ corresponds to the Hadamard product. In practice, μ maps some elements of Q, deformed by S and selected by P, into the weights. It is also worth highlighting that, in QALS, the QUBO problem variables are defined in the $\{-1, +1\}$ domain.

The QALS scheme is outlined in Algorithm 1. In particular, the tabu-based encodings are generated according to Equation (2.10). To achieve this, an auxiliary function $P \mapsto g(P, p)$, which alters a permutation by selecting elements for shuffling with probability p, is exploited. In practice, the function g is responsible for generating the permutations that induce the encodings into the annealer architecture (Algorithm 1, lines 5 and 25). In addition, the mapping process takes into account the

```
Data: Matrix $Q$ of order $n$ encoding a QUBO problem, annealer adjacency matrix $\mathcal{A}$ of order $n$
Input: Energy function of the annealer $\mathsf{E}(\Theta, \mathbf{z})$, permutation modification function $g(P, p)$, solution
       modification function $h(\mathbf{z})$, minimum probability $0 < p_\delta < 0.5$ of permutation modification,
       probability decreasing rate $\eta > 0$, candidate perturbation probability $q > 0$, number $N$ of
       iterations at constant $p$, initial balancing factor $\lambda_0 > 0$, number of annealer runs $k \geq 1$,
       termination parameters $i_{max}$, $N_{max}$, $d_{min}$
Result: $\mathbf{z}^*$ vector with $n$ elements in $\{-1, 1\}$ solution of the QUBO problem
function $f_Q(\mathbf{z})$:
    return $\mathbf{z}^T Q \mathbf{z}$ ;
$P \leftarrow I_n$;
$p \leftarrow 1$;
$P_1 \leftarrow g(P, 1); P_2 \leftarrow g(P, 1)$;    // generate two permutation matrices perturbing the identity
$\Theta_1 \leftarrow P_1^T Q P_1 \circ \mathcal{A}$; $\Theta_2 \leftarrow P_2^T Q P_2 \circ \mathcal{A}$;    // weights initialization
/* run the annealer $k$ times with weights $\Theta_1$ and $\Theta_2$ */
$\mathbf{z}_1 \leftarrow P_1^T \widehat{\operatorname{argmin}}_{\mathbf{z}}(\mathsf{E}(\Theta_1, \mathbf{z}))$; $\mathbf{z}_2 \leftarrow P_2^T \widehat{\operatorname{argmin}}_{\mathbf{z}}(\mathsf{E}(\Theta_2, \mathbf{z}))$;  // estimate energy argmin, $P_1^T$ and $P_2^T$
  map back the variables
$f_1 \leftarrow f_Q(\mathbf{z}_1); f_2 \leftarrow f_Q(\mathbf{z}_2)$;    // evaluate $f_Q$
/* use the best to initialize $\mathbf{z}^*$ and $P^*$; use the worst to initialize $\mathbf{z}'$ */
if $f_1 < f_2$ then
    $\mathbf{z}^* \leftarrow \mathbf{z}_1$; $f^* \leftarrow f_1$; $P^* \leftarrow P_1$ $\mathbf{z}' \leftarrow \mathbf{z}_2$;
else
    $\mathbf{z}^* \leftarrow \mathbf{z}_2$ ; $f^* \leftarrow f_2$; $P^* \leftarrow P_2$; $\mathbf{z}' \leftarrow \mathbf{z}_1$;
end
if $f_1 \neq f_2$ then
    $S \leftarrow \mathbf{z}' \otimes \mathbf{z}' - I_n + diag(\mathbf{z}')$;    // use $\mathbf{z}'$ to initialize the tabu matrix $S$
else
    $S \leftarrow 0$;    // otherwise set all the elements of $S$ to zero
end
$e \leftarrow 0$; $d \leftarrow 0$; $i \leftarrow 0$; $\lambda \leftarrow \lambda_0$;
repeat
    $Q' \leftarrow Q + \lambda S$;    // scale and add the tabu matrix
    if $N$ divides $i$ then
        $p \leftarrow p - (p - p_\delta)\eta$;
    end
    $P \leftarrow g(P^*, p)$;    // modify permutation $P^*$
    $\Theta' \leftarrow P^T Q' P \circ \mathcal{A}$;    // weights initialization
    /* run the annealer $k$ times with weights $\Theta'$ */
    $\mathbf{z}' \leftarrow P^T \widehat{\operatorname{argmin}}_{\mathbf{z}}(\mathsf{E}(\Theta', \mathbf{z}))$;    // estimate energy argmin, $P^T$ maps back the variables
    with probability $q$  $\mathbf{z}' \leftarrow h(\mathbf{z}', p)$;    // possibly perturb the candidate
    if $\mathbf{z}' \neq \mathbf{z}^*$ then
        $f' \leftarrow f_Q(\mathbf{z}')$;    // evaluate $f_Q$
        if $f' < f^*$ then
            $swap(\mathbf{z}', \mathbf{z}^*)$; $f^* \leftarrow f'$; $P^* \leftarrow P$; $e \leftarrow 0$; $d \leftarrow 0$;    // $\mathbf{z}'$ is better
            $S \leftarrow S + \mathbf{z}' \otimes \mathbf{z}' - I_n + diag(\mathbf{z}')$;    // use $\mathbf{z}'$ to update the tabu matrix $S$
        else
            $d \leftarrow d + 1$;
            with probability $(p - p_\delta)^{(f' - f^*)}$ $swap(\mathbf{z}', \mathbf{z}^*)$; $f^* \leftarrow f'$; $P^* \leftarrow P$; $e \leftarrow 0$;
        end
        update the balancing factor $\lambda$ with $\lambda \leq \lambda_0$;
    else
        $e \leftarrow e + 1$;
    end
    $i \leftarrow i + 1$;
until $i = i_{max}$ or $(e + d \geq N_{max}$ and $d < d_{min})$;
return $\mathbf{z}^*$;
```

Algorithm 1: Quantum Annealing Learning Search for QUBO problems (taken from [12]).

actual annealer topology, which is represented by the graph matrix $\mathcal{A}$ (Algorithm 1, lines 6 and 26). The permutations are also exploited to map the solutions obtained from the annealer back to the original problem's solution space (Algorithm 1, lines 7 and 27) and to define the optimal map μ^*. Regarding the function g, when $p = 1$, the generated permutation is entirely random. For $0 < p < 1$, the permutation partially resembles the input one. Instead, if $p = 0$, the resulting permutation would match the input one. However, this scenario never occurs since the probability of an element being shuffled gradually decreases to a value $0 < p_\delta < 0.5$ with rate η (Algorithm 1, lines 22-24).

The additive interaction of the tabu matrix S, scaled by a balancing factor λ, with the QUBO matrix Q (Algorithm 1, line 21) guides the quantum annealing search with an energy profile consistent with Equation (2.10). The consequence is that the algorithm does not search just for solutions of subproblems. Indeed, S contains information about the already visited solutions with objective function values higher than f^* (Algorithm 1, lines 15 and 33). Additionally, the balancing factor λ, initialized to λ_0, is decreased throughout the iterations (Algorithm 1, line 38), with the purpose of preventing the tabu matrix S from overshadowing the information about f_Q carried by Q. In general, λ should depend on the number of bad candidates penalized by S.

It is also worth highlighting that the solution returned by the annealer (Algorithm 1, line 27) could be perturbed by the function $h(\mathbf{z}', p)$, which flips any entry of $\mathbf{z}'$ with probability p (Algorithm 1, line 28). These perturbations, which occur with probability q, are necessary in order to guarantee the convergence [81]. Furthermore, suboptimal solutions (Algorithm 1, line 36) are accepted with probability $(p - p_\delta)^{(f' - f^*)}$. By comparison with the acceptance rule of SA, it is possible to notice that the parameter p is related to the temperature parameter of SA by the relationship $T = -\ln^{-1}(p - p_\delta)$. Hence, $T \to 0$ as $p \to p_\delta$.

Eventually, the iterative process outlined in Algorithm 1 terminates either upon convergence or when the specified number of iterations has been achieved. In particular, line 19 of Algorithm 1 establishes three distinct counters for controlling the convergence: e counts the number of consecutive occurrences of the current best solution (Algorithm 1, line 40); d keeps track of the number of times the current best solution and the new solution are different, yet the current solution is better (Algorithm 1, line 35); the variable i represents the current number of iterations. The values of these counters are compared against the input thresholds in the termination condition (Algorithm 1, line 43).

2.4 Bayesian Network Structure Learning

A Bayesian network (BN) is a directed acyclic graph (DAG) illustrating the conditional dependencies of a collection of random variables [84]. In this representation, the nodes correspond to the variables, while the edges represent the conditional dependencies between them. Furthermore, each node is coupled with the conditional probability distribution of the node given its parent nodes.

The procedure presented by O'Gorman et al. [76] addresses the task of Bayesian Network Structure Learning (BNSL), a task that has received much attention [109] and consists in identifying the Bayesian network that most likely has generated a given dataset. In particular, O'Gorman et al. have devised a novel QUBO formulation of the BNSL problem that is compatible with the quantum annealing hardware, including the necessary lower bounds for penalties. It is worth noting that this problem is NP-Complete [18] and that a polynomial speedup is expected by O'Gorman et al. when using their quantum-annealing-based approach. Obviously, this is not the sole application of quantum computing in the context of Bayesian networks. For instance, Ozols et al. have proposed a quantum counterpart to the classical rejection sampling algorithm employed in Bayesian network inference [77], while Borujeni et al. have introduced a systematic technique for designing a quantum circuit that represents a generic discrete BN [13]. However, this section focuses on the BNSL task.

More in detail, a Bayesian network can be represented as a pair (B_s, B_p), where B_s denotes a Directed Acyclic Graph (DAG) and B_p represents the corresponding set of conditional probabilities. Given a database $D = \{\mathbf{x}_i | 1 \leq i \leq N\}$, where $\mathbf{x}_i$ represents the state of all variables, the goal is to identify the network structure that maximizes the posterior probability distribution $p(B_s|D)$. However, exploiting Bayes' theorem, which establishes the proportionality between $p(B_s|D)$ and $p(D|B_s)$, the problem can be reformulated as the maximization of $p(D|B_s)$, which is equal to

$$p(D|B_s) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})}, \tag{2.11}$$

where Γ denotes the gamma function, q_i corresponds to the number of joint states associated with the parent set of the i-th random variable, r_i represents the number of states of the i-th random variable, N_{ijk} stands for the number of occurrences in D where the i-th random variable is in its k-th state and its parent set is in its j-th state, α_{ijk} is the hyperparameter of the assumed Dirichlet prior for the node's conditional probability distribution, N_{ij} and α_{ij} denote the sums of the corresponding parameter values over all possible k values.

2.4.1 O'Gorman's QUBO Formulation

In their study [76], O'Gorman et al. introduce a Hamiltonian function tailored for the BNSL problem. Given the Hamiltonian, the QUBO matrix can be built in a straightforward way, mapping the coefficients of the variables into the matrix entries. Specifically, the BNSL Hamiltonian is composed of three key components: the score Hamiltonian (H_{score}), which evaluates the quality of the solution graph; the max Hamiltonian (H_{max}), which penalises solutions including nodes with a number of parents exceeding a certain threshold m, dictated by resource constraints; the cycle Hamiltonian (H_{cycle}), further split into the consistency Hamiltonian ($H_{consist}$) and the transitivity Hamiltonian (H_{trans}), which penalise solutions that contain cycles. Consequently, the BNSL Hamiltonian (H) is given by

$$H(\mathbf{d}, \mathbf{y}, \mathbf{r}) = H_{score}(\mathbf{d}) + H_{max}(\mathbf{d}, \mathbf{y}) + H_{cycle}(\mathbf{d}, \mathbf{r}), \tag{2.12}$$

where $\mathbf{d}$ denotes the $n(n-1)$ bits employed to represent the presence or absence of edges between nodes, while $\mathbf{y}$ and $\mathbf{r}$ are auxiliary variables used to encode the constraints.

The score Hamiltonian (H_{score}) is computed independently for each variable, and the components are then summed together. Specifically, the score Hamiltonian for the i-th variable is defined as

$$H^{(i)}_{score}(\mathbf{d}_i) = \sum_{\substack{J \subset \{1 \dots n\} \setminus \{i\} \\ |J| \le m}} \left(w_i(J) \prod_{j \in J} d_{ji} \right), \tag{2.13}$$

where $\mathbf{d}_i$ is the vector of bits (d_{ji}) representing edges directed towards the considered node, m denotes the maximum allowed size for the parent set, and w_i is calculated as

$$w_i(J) = \sum_{l=0}^{|J|} (-1)^{|J|-l} \sum_{\substack{K \subset J \\ |K| = l}} s_i(K), \tag{2.14}$$

where s_i is a score value obtained from Eq. (2.11) with the introduction of a logarithm for numerical efficiency. In detail, s_i is given by

$$s_i(\Pi_i(B_s)) = -\log \left(\prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})} \right), \tag{2.15}$$

where $\Pi_i(B_s)$ represents the parent set of the i-th node. In practical terms, the sum of the s_i values corresponds to $-\log p(D|B_s)$.

Analogously, the max Hamiltonian is calculated independently for every variable as

$$H^{(i)}_{max}(\mathbf{d}_i, \mathbf{y}_i) = \delta^{(i)}_{max}(m - d_i - y_i)^2, \tag{2.16}$$

where $\delta^{(i)}_{max} > 0$ corresponds to the penalty weight, d_i denotes the in-degree of the i-th node (that is equal to $\sum_{1 \le j \le n \cap j \ne i} d_{ji}$), and $y_i \in \mathbb{Z}$ is a slack variable (encoded through binary expansion in $\mathbf{y}_i$

employing μ bits[2]) that cancels $H^{(i)}_{max}$'s contribution if the constraint is satisfied. Indeed, $H^{(i)}_{max}$ is zero if the node taken into account has at most m parents, otherwise it brings a positive penalty.

Eventually, the cycle Hamiltonian is defined as the sum of two terms:

$$H_{cycle}(\mathbf{d}, \mathbf{r}) = H_{trans}(\mathbf{r}) + H_{consist}(\mathbf{d}, \mathbf{r}), \tag{2.17}$$

with $\mathbf{r}$ representing $n(n-1)/2$ auxiliary boolean variables that are used to encode a topological order (r_{ij} is 1 if the i-th node precedes the j-th node, 0 otherwise). In particular, the transitivity Hamiltonian penalises the cycles of length three in the r_{ij} values, and is calculated independently for each 3-set of variables as

$$H^{(ijk)}_{trans}(r_{ij}, r_{jk}, r_{ik}) = \delta^{(ijk)}_{trans}\left(r_{ik} + r_{ij}r_{jk} - r_{ij}r_{ik} - r_{jk}r_{ik}\right), \tag{2.18}$$

with $\delta^{(ijk)}_{trans}$ being the positive penalty that is added if the i-th, j-th and k-th variables form a 3-cycle. As in the other cases, the $H^{(ijk)}_{trans}$ components are summed together to obtain the full H_{trans}. Regarding the consistency Hamiltonian, it penalises the solutions for which the topological order encoded in $\mathbf{r}$ is not consistent with the graph structure encoded in $\mathbf{d}$. In practical terms, it disadvantages the solutions for which $r_{ij} = 1$ and $d_{ji} = 1$, or $r_{ij} = 0$ and $d_{ij} = 1$. This Hamiltonian is computed independently for each pair of variables as

$$H^{(ij)}_{consist}(d_{ij}, d_{ji}, r_{ij}) = \delta^{(ij)}_{consist}(d_{ji}r_{ij} + d_{ij} - d_{ij}r_{ij}), \tag{2.19}$$

with $\delta^{(ij)}_{consist}$ being the positive penalty introduced in the case of an inconsistency. It is also worth noticing that the order of the superscript indices in $\delta^{(ijk)}_{trans}$ and $\delta^{(ij)}_{consist}$ does not matter, since the set of variables is always the same.

In practice, the QUBO formulation of the BNSL problem involves $n(n-1)$ binary variables (d_{ij}) representing the graph structure, $n\mu = n\lceil \log_2(m+1) \rceil$ binary slack variables (y_{il}) associated with the maximum parent constraint, and $n(n-1)/2$ binary variables (r_{ij}) related to the absence of cycles constraint. Thus, the QUBO encoding of n Bayesian variables necessitates $\mathcal{O}(n^2)$ binary variables. However, since H_{score} includes multiplications with m factors, if $m \geq 3$, further steps and slack variables are required to transform the problem into a quadratic form. For instance, to reduce a BNSL problem with $m = 3$ into a quadratic form, $n\lfloor \frac{(n-2)^2}{4} \rfloor$ binary slack variables are required [76], raising the total number of binary variables to $\mathcal{O}(n^3)$.

Regarding the penalty values, O'Gorman et al. have provided the following lower bounds (and demonstrated their sufficiency):

$$\delta^{(i)}_{max} > \max_{j \neq i} \Delta_{ji},\ 1 \leq i \leq n, \tag{2.20}$$

$$\delta^{(ij)}_{consist} > (n-2) \max_{k \notin \{i,j\}} \delta^{(ijk)}_{trans},\ 1 \leq i < j \leq n, \tag{2.21}$$

$$\delta^{(ijk)}_{trans} = \delta_{trans} > \max_{\substack{1 \leq i', j' \leq n \\ i' \neq j'}} \Delta_{i'j'},\ 1 \leq i < j < k \leq n, \tag{2.22}$$

with Δ_{ji} being an estimate of the largest increase in score caused by the insertion of an arc from the j-th to the i-th node. For $m = 2$, it is given by

$$\Delta_{ji} = \max\{0, \Delta'_{ji}\}, \tag{2.23}$$

$$\Delta'_{ji} = -w_i(\{j\}) - \sum_{\substack{1 \leq k \leq n \\ k \neq i,j}} \min\{0, w_i(\{j,k\})\}. \tag{2.24}$$

Conversely, for $m \geq 3$, determining Δ_{ji} is an intractable optimization problem.

[2] $y_i = \sum_{l=1}^{\mu} 2^{l-1} y_{il}$, with $\mu = \lceil \log_2(m+1) \rceil$

2.5 Quantum-Trained Support Vector Machines

A Support Vector Machine (SVM) is a supervised machine learning algorithm for binary classification tasks [22]. Essentially, an SVM aims at finding the optimal hyperplane separating the data samples belonging to different classes. The term support vectors refers to the data points lying close to the decision boundary and contributing to the prediction of new labels. Actually, SVMs are not limited to linearly separable problems. Indeed, with the introduction of kernel functions computing the similarity of data points in higher-dimensional feature spaces without explicitly mapping them, SVMs become able to manage complex decision boundaries and the complexity of the problem does not increase (this is the so-called kernel trick) [97]. For instance, the Gaussian kernel function is defined as

$$k(\mathbf{x}_m, \mathbf{x}_n) = e^{-\gamma \|\mathbf{x}_m - \mathbf{x}_n\|^2}, \tag{2.25}$$

where $\mathbf{x}_m$ and $\mathbf{x}_n$ are two input data points, and $\gamma > 0$ is the kernel width. Moreover, different formulations of the SVM learning problem exist, and also extensions to multiclass classification and regression tasks. Concerning the quantum counterparts, a fully-quantum SVM for binary classification has been proposed by Rebentrost et al. [89]. Instead, Havlíček et al. have introduced both a quantum SVM based on a variational circuit and a quantum kernel estimator [43]. However, in this work, the focus is on classical SVMs trained on a quantum annealer. In detail, the quantum-trained models for binary and multiclass classification tasks have been taken into account here (a version for regression also exists [78]). The QUBO formulations of the corresponding learning problems are provided below.

2.5.1 Quantum Binary Support Vector Machine (QBSVM)

The quantum-trained SVM for binary classification proposed by Willsch et al. [118], and denoted here as QBSVM, is based on the dual formulation of the SVM. Specifically, given a dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=0,\ldots,N-1}$, with $\mathbf{x}_n \in \mathbb{R}^d$ being a d-dimensional feature vector and $y_n \in \{-1, +1\}$ being the corresponding class label, the dual formulation of the SVM learning problem is

$$\begin{gathered} \min_{\boldsymbol{\alpha} \in \mathbb{R}^N} \frac{1}{2} \sum_{n,m} \alpha_n \alpha_m y_n y_m k(\mathbf{x}_n, \mathbf{x}_m) - \sum_n \alpha_n \\ \text{subject to} \quad 0 \leq \alpha_n \leq A \quad \text{and} \quad \sum_n \alpha_n y_n = 0, \end{gathered} \tag{2.26}$$

where $\boldsymbol{\alpha}$ is the vector of coefficients to be found, k is the kernel function, and A is a regularization factor[3]. Once the α_n coefficients have been determined (the support vectors are the ones with $\alpha_n \neq 0$), the label prediction for a test instance $\mathbf{x}$ is obtained by taking the sign of

$$f(\mathbf{x}) = \sum_n \alpha_n y_n k(\mathbf{x}_n, \mathbf{x}) + b,$$

where b can be calculated as

$$b = \frac{\sum_n \alpha_n (A - \alpha_n) \left[y_n - \sum_m \alpha_m y_m k(\mathbf{x}_m, \mathbf{x}_n)\right]}{\sum_n \alpha_n (A - \alpha_n)}. \tag{2.27}$$

Since the solutions to the learning problem illustrated in Equation (2.26) are real-valued coefficients, a binary encoding is required for solving it with a quantum annealer. In detail, the encoding proposed by Willsch et al. is the following:

$$\alpha_n = \sum_{k=0}^{K-1} B^k a_{Kn+k},$$

where K is the number of binary variables utilized for encoding α_n, B is the base exploited for the encoding, and a_{Kn+k} are binary variables. Given this encoding, the QUBO formulation of the SVM

[3] The regularization factor of the SVM is usually denoted as C. To avoid ambiguity with the number of classes in the QMSVM formulation (see Section 2.5.2), C has been replaced here with A.

learning problem turns out to be

$$\min_{\boldsymbol{a}\in\mathbb{R}^{NK}} \frac{1}{2}\sum_{n,m,k,j} a_{Kn+k}a_{Km+j}B^{k+j}y_n y_m k(\mathbf{x}_n,\mathbf{x}_m) - \sum_{n,k} B^k a_{Kn+k} + \frac{\xi}{2}\left(\sum_{n,k} B^k a_{Kn+k} y_n\right)^2 =$$

$$= \min_{\boldsymbol{a}\in\mathbb{R}^{NK}} \sum_{n,m=0}^{N-1}\sum_{k,j=0}^{K-1} a_{Kn+k}\ Q_{Kn+k,Km+j}\ a_{Km+j},$$

where ξ is a multiplier needed to represent the second constraint in Equation (2.26), and Q is the $NK \times NK$ QUBO matrix. Hence, Q is defined as

$$Q_{Kn+k,Km+j} = \frac{1}{2}B^{k+j}y_n y_m\left[k(\mathbf{x}_n,\mathbf{x}_m)+\xi\right] - \delta_{nm}\delta_{kj}B^k,$$

with δ representing the Kronecker delta function. Regarding the first constraint in Equation (2.26), it is automatically satisfied by the choice of the encoding. Indeed, $a_n \geq 0$ by definition and $A = \sum_{k=0}^{K-1} B^k$. Instead, the bias b can be computed using Equation (2.27) or adjusted afterwards.

The just described QUBO formulation might produce matrices that cannot be embedded in the available quantum annealers. To address this issue, Willsch et al. have proposed an interesting approach. In particular, the full dataset is split into L disjoint slices. Then, for each slice, the S best solutions found by the annealer are combined by averaging their decision functions. In practice, the average α_n coefficients and biases are computed for each slice. Eventually, the full decision function is defined as

$$F(x) = \frac{1}{L}\sum_{l=0}^{L-1}\left[\sum_{n=0}^{N-1}\overline{\alpha}_n^{(l)}y_n^{(l)}k(\mathbf{x}_n^{(l)},\mathbf{x}) + \overline{b}^{(l)}\right],$$

where the superscript l denotes the slice, $\overline{\alpha}_n^{(l)}$ is the n-th mean coefficient for the l-th slice, and $\overline{b}^{(l)}$ is the mean bias for the l-th slice. Actually, averaging the S best solution could be advantageous also in cases where the dataset split is not necessary.

2.5.2 Quantum Multiclass Support Vector Machine (QMSVM)

The quantum-trained SVM for multiclass classification proposed by Delilbasic et al. [26], and denoted as QMSVM in the original paper, is based on the Crammer-Singer (CS) SVM [21], a single step SVM for multiclass classification. In detail, given a dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=0,\ldots,N-1}$, with $\mathbf{x}_n \in \mathbb{R}^d$ being a d-dimensional feature vector and $y_n \in \{0,\ldots,C-1\}$ being the corresponding label, the learning problem of the CS SVM is

$$\min_T \frac{1}{2}\sum_{n_1,n_2=0}^{N-1} k(\mathbf{x}_{n_1},\mathbf{x}_{n_2})\sum_{c=0}^{C-1}\tau_{n_1c}\tau_{n_2c} - \beta\sum_{n=0}^{N-1}\sum_{c=0}^{C-1}\delta_{cy_n}\tau_{nc}$$
$$\text{subject to} \quad \sum_{c=0}^{C-1}\tau_{nc} = 0 \ \ \forall n \quad \text{and} \quad \tau_{nc} \leq 0 \ \ \forall n, \forall c \neq y_n, \tag{2.28}$$

where $T = (\tau_{nc})_{0\leq n\leq N-1,\ 0\leq c\leq C-1}$ represents the matrix of the NC coefficients to be determined, with $\tau_{nc} \in [-1,+1]$, δ denotes the Kronecker delta, and β is a regularization term. Once the τ_{nc} coefficients have been found, the label prediction for a test instance $\mathbf{x}$ is given by

$$y = \operatorname*{argmax}_{0\leq c\leq C-1}\sum_{n=0}^{N-1}\tau_{nc}k(\mathbf{x}_n,\mathbf{x}). \tag{2.29}$$

In order to solve the learning problem of Equation (2.28) using a quantum annealer, a binary encoding of the τ_{nc} variables is required. Specifically, Delilbasic et al. have proposed the following encoding based on the uniform sampling of the $[-1,+1]$ interval:

$$\tau_{nc} = -1 + \frac{2}{2^K - 1} \sum_{k=0}^{K-1} 2^k a_{nCK+cK+k},$$

where K is the number of binary variables used to encode τ_{nc}, and $a_{nCK+cK+k}$ are binary variables. Given this encoding, the QUBO formulation of the CS SVM learning problem turns out to be

$$\min_{\boldsymbol{a} \in \mathbb{R}^{NCK}} \sum_{n_1, n_2, c_1, c_2, k_1, k_2} a_{n_1CK+c_1K+k_1}\ Q_{n_1CK+c_1K+k_1, n_2CK+c_2K+k_2}\ a_{n_2CK+c_2K+k_2},$$

where the $NCK \times NCK$ QUBO matrix Q is defined as

$$\begin{aligned} Q_{n_1CK+c_1K+k_1, n_2CK+c_2K+k_2} &= \\ &= \delta_{n_1n_2}\delta_{c_1c_2}\delta_{k_1k_2}\frac{2^{k_1+1}}{2^K-1}\left(-\sum_{n_3=0}^{N-1} k(\mathbf{x}_{n_1}, \mathbf{x}_{n_3}) - \delta_{c_1y_{n_1}}(\beta+\mu) - 2C\mu + \mu\right) + \\ &\qquad + \delta_{c_1c_2}\frac{2^{k_1+k_2+1}}{(2^K-1)^2}k(\mathbf{x}_{n_1}, \mathbf{x}_{n_2}) + \delta_{n_1n_2}\frac{2^{k_1+k_2+2}\mu}{(2^K-1)^2}, \end{aligned}$$

with μ being the penalty weight associated with the constraints of Equation (2.28).

Lastly, Delilbasic et al. have proposed a method for taking advantage of the multiple solutions returned by the annealer. In particular, given the S best solutions found by the annealer, each of them is tested on a validation set (which can coincide with the training set). Then, a weighted average is performed. In detail, the weights of the solutions achieving an accuracy above a certain threshold are given by a softmax function applied to the values $\mathit{multiplier} \cdot \mathit{accuracy}_s$, where *multiplier* is a real value and $\mathit{accuracy}_s$ is the accuracy obtained by the s-th solution. Instead, the weights of the other solutions are set to zero. The resulting $\overline{\tau}_{nc}$ mean variables are used in Equation (2.29), in the place of τ_{nc}, to predict the new labels.

2.6 Local Support Vector Machines

Reducing the number of samples provided as input to a classical machine learning model by means of a locality technique has already proven to be a successful strategy, providing performance enhancements compared to the base model. Let us consider the most straightforward locality technique, i.e., the k-Nearest Neighbors (k-NN) algorithm, which selects the data samples closest to the target instance based on a given metric (more details about the k-NN algorithm can be found in Section 3.3). In addition, let us restrict ourselves to the SVM as the base classification model. In 2006, Blanzieri and Melgani have proposed and empirically evaluated the kNNSVM [9], i.e., a local SVM trained on the data samples selected by a k-NN model, obtaining good results. Specifically, the k-NN algorithm must return the nearest neighbors with respect to the transformed feature space where the SVM operates, which could be an issue when using the kernel trick. However, in the case of RBF kernels (such as the Gaussian kernel) and polynomial kernels with degree 1, the Euclidean distance can be directly used as the distance metric for the k-NN [9]. Furthermore, local SVMs have been theoretically characterised by, for instance, Hable [42] and Meister and Steinwart [70]. Nevertheless, despite the accuracy improvement and the reduced training time per model (due to the lower number of samples used for training), the kNNSVM classifier requires to train an SVM on the k-neighborhood of each test sample (unless all nearest neighbors belong to the same class), which represents a serious bottleneck in terms of execution time. To solve this issue, Segata and Blanzieri have developed the approach presented below.

2.6.1 Fast Local Kernel Support Vector Machine (FaLK-SVM)

Fast Local Kernel Support Vector Machine (FaLK-SVM) [104] improves the execution time of the kNNSVM classifier [9] by taking advantage of a data structure proposed by Beygelzimer et al. for efficient nearest-neighbor operations, namely, the cover tree [6]. In detail, the cover tree is a data structure with the following properties: the root is a randomly selected sample; if a data sample appears in a level, it is present in all lower levels; the distance between the samples belonging to the i-th level (levels are decreasingly indexed) is greater than b^i, with $b > 1$; each node in the i-th level has a parent node such that the distance between the corresponding samples is smaller than b^{i+1}.

In practice, the idea consists in covering the training set using a set of local SVM models, and predicting the test instance label using the most appropriate (pre-trained) local model. In more detail, the training of the FaLK-SVM model works as follows: a cover tree is built on the training dataset; the centres of the local SVMs are chosen by means of the cover tree, which enables the efficient retrieval of data points that are far from each other, limiting the local models overlapping; the local SVM models are trained as usual (if a local training set includes only one class, the training is avoided). In particular, the selection procedure ends when each training sample belongs to the k'-neighborhood of at least one centre, with $k' < k$ being a fixed hyperparameter controlling the redundancy of the local models. Additionally, during the training phase, the association between each training point and the centre for which the neighbor ranking of the considered training point is the smallest is pre-computed. In this way, at evaluation time, it is sufficient to find the test-instance nearest neighbor in the training set and run the local model with which that data point is associated. Regarding the time complexity, the training phase has a worst-case complexity of $O(kN \times \max(\log N, k^2))$, where k is the number of nearest neighbors selected and N is the number of training points, while the prediction of a new label has a complexity of $O(\max(\log N, k))$.

Actually, in the same article, two variants of FaLK-SVM have been also introduced. The former, which has not been considered in this work, is FaLK-SVMc. Specifically, FaLK-SVMc has proven to be faster than FaLK-SVM, but also less accurate. The difference lies in the policy used for selecting the local model at prediction time. Instead, the latter, which is the one that has been used in this work, is denoted as FaLK-SVMl. Essentially, FaLK-SVMl includes a grid-search model selection procedure that is executed before the training of FaLK-SVM. In practice, each combination of local model parameters is evaluated using a custom κ-fold cross-validation on m local models, whose centres are randomly selected. In this custom validation procedure, the split into folds is applied to the k' nearest neighbors of the model centre; the remaining $k - k'$ points of the k-neighborhood are added to the training set of each κ-fold iteration. In the end, the parameter configuration maximizing the average accuracy of the m models is chosen and used for all local models.

3 Quantum Machine Learning on Circuit Model

This chapter introduces the field of quantum machine learning, the most common data encoding schemes, and the models and works from the literature taken into account. In particular, this chapter is a reworked version of different parts of the background sections of various articles [127, 128].

3.1 A Brief Overview

In essence, Machine Learning (ML) is the automation of methodologies designed to extract information from gathered data. When data analysis techniques are executed on conventional digital computers, it falls under the realm of classical ML. Instead, if quantum machines are employed, it pertains to quantum ML. The first quantum versions of ML algorithms were introduced approximately twenty years ago [100, 111]. However, the surge in interest has occurred only in the last decade thanks to the development of the first operational prototypes of quantum machines by companies like IBM [45], Rigetti [91], and D-Wave Systems [49], and the publication of several intriguing results [7, 30, 43, 89]. Concerning practical advantages with respect to classical ML, quantum subroutines have been embedded into ML frameworks, enabling, for instance, the efficient calculation of distances in the feature space [98], with advantages in classification and clustering tasks. In addition, Grover-based subroutines have been employed to locate items within unsorted databases [115], offering a quadratic speedup compared to exhaustive searches. Such techniques find applications, for instance, in the realm of pattern recognition. In general, there is a growing interest in hybrid approaches, where quantum co-processors efficiently tackle specific subproblems within more complex learning schemes.

The drive behind developing novel QML approaches is summarised in the reasoning provided by Biamonte et al. [7]: given the challenge of simulating (even small) quantum systems with classical computers, it is reasonable to assume that (even small) quantum processors could find data structures that are challenging to unveil classically. Consequently, QML emerges as a promising avenue towards meaningful applications of the small-scale quantum machines available today and in the near future. On the other hand, with strong assumptions of universality, large scale, and fault tolerance, it is conceivable to formulate numerous QML algorithms that surpass their classical counterparts. This holds significance for understanding the foundations of quantum computing and demonstrating the true potential of quantum computers. However, in the pursuit of advancing quantum technologies in the immediate future, it is useful to consider the limitations of current quantum hardware while exploring innovative QML approaches.

From a mathematical perspective, there exists a compelling motivation for developing ML algorithms tailored for quantum machines, given by a formal analogy between quantum mechanics and ML. Both domains heavily rely on matrix operations within high-dimensional vector spaces. In practice, the Hilbert spaces used to describe physical quantum systems can serve as feature spaces for data representations. In this context, linear algebraic operations find physical realization in the time evolution of quantum states. For instance, in the circuit model of quantum computation, this evolution is described as the action of quantum gates. Furthermore, the representation of data within quantum states offers advantages in terms of space resources. These advantages stem from the exponential growth of the dimensionality of the Hilbert space of a multi-qubit system with respect to the number of qubits. Consequently, the controlled dynamics of a small number of qubits towards a target state may correspond to applying intricate linear algebraic operations on the considered feature space.

To conclude this brief introduction to QML, it is worth highlighting that QML is probably the most promising way for discovering effective applications of the existing small-scale quantum computers. In addition, beyond just quantum speedup, which is not the sole advantage, other dimensions of merit should be taken into account. These include enhanced accuracy in prediction, increased expressive power, superior generalization capabilities, and the ability to avoid plateaus in training.

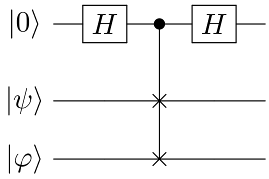
3.2 Data Encoding and SWAP Test

A fundamental concept in QML is *quantum encoding*, which denotes any method that translates classical data (such as a list of symbols) into quantum states. In detail, efficiently loading substantial volumes of data into quantum architectures represents a significant challenge in the current state of QML. Indeed, the state preparation necessary for running various QML algorithms can be executed efficiently only under the strong assumption of the availability of a Quantum Random Access Memory (QRAM) [37]. To delve deeper, consider an n-qubit register and let $\{|i\rangle\}_{i=0,\dots,2^n-1}$ be a fixed orthonormal basis of the corresponding Hilbert space, known as the *computational basis*. The most straightforward quantum encoding method is *basis encoding*, wherein bit strings of length n are encoded into the states constituting the computational basis. Thus, n qubits are employed to encode n bits of classical information, presenting intriguing quantum possibilities, such as creating superpositions of data and enabling non-classical correlations through entanglement. Alternatively, a more efficient quantum encoding method in terms of space resources is *amplitude encoding*. In this case, a normalized complex vector $\mathbf{x} \in \mathbb{C}^{2^n}$ representing a data instance is encoded into the coordinates (or amplitudes) of a quantum state with respect to the computational basis, namely,

$$|\psi\rangle = \sum_{i=0}^{2^n-1} x_i \, |i\rangle \qquad (n\text{-qubit state}).$$

The amplitude encoding harnesses the exponential storage capability of a quantum memory. However, the direct retrieval of the stored data is not feasible. This constraint arises because the amplitudes cannot be observed. Indeed, only the probabilities $|x_i|^2$ can be estimated.

It is also worth introducing a simple example of quantum processing that finds practical utility in QML, namely, the SWAP test [15]. The corresponding quantum circuit is depicted as follows:



,

where $|\psi\rangle$ and $|\varphi\rangle$ are n-qubit states. In detail, the SWAP gate, which operates on the quantum states $|\psi\rangle$ and $|\varphi\rangle$, is controlled by a qubit initially prepared in state $|0\rangle$ (this can be realized using n Fredkin gates). Through a straightforward calculation, the probability of measuring 0 in the first qubit is given by $\mathbb{P}(0) = \frac{1}{2}\left(1 + |\langle\psi|\varphi\rangle|^2\right)$, where $\langle\psi|\varphi\rangle$ represents the inner product between the states $|\psi\rangle$ and $|\varphi\rangle$ in the Dirac notation. Moreover, estimating $\mathbb{P}(0)$ with an error margin of ϵ necessitates approximately $O(\epsilon^{-2})$ repetitions, as dictated by the binomial proportion confidence interval for a Bernoulli trial. In practice, the SWAP test allows the efficient calculation of the fidelity of the quantum states $|\psi\rangle$ and $|\varphi\rangle$. For two pure quantum states, the fidelity is defined as

$$\mathcal{F}(|\psi\rangle, |\varphi\rangle) = |\langle\psi|\varphi\rangle|^2 = (\cos(\psi,\varphi) \cdot \parallel \psi \parallel \cdot \parallel \varphi \parallel)^2 = \cos^2(\psi,\varphi), \tag{3.1}$$

with $\cos(\psi,\varphi)$ being the cosine similarity between $|\psi\rangle$ and $|\varphi\rangle$, and the norms of $|\psi\rangle$ and $|\varphi\rangle$ being 1 by definition. Therefore, by representing data vectors into the amplitudes of $|\psi\rangle$ and $|\varphi\rangle$, it is possible to estimate their dot product/cosine similarity via the SWAP test.

3.3 Quantum k-Nearest Neighbors

The k-NN [34] is a classification algorithm involving three main steps: computing the distance to the training elements; identifying the k nearest neighbors, which are the k elements closest to the test instance; predicting the class label by majority voting. Several quantum variants employing different distance measures have been suggested, and a shared characteristic among them is the usage of a superposition state to execute parallel operations (quantum parallelism), like the simultaneous computation of the distances values.

To begin with, quantum k-NN algorithms using the Hamming distance, which needs binary features, have been developed by Schuld et al. [99], Wiśniewska and Sawerwain [120], Ruan et al. [95], Zhou et al. [131], and Li et al. [65]. In the first two studies, the Hamming distances are calculated by encoding the sums of the qubits differences (differences computed by means of controlled-NOT gates) into the amplitudes of the quantum states through a unitary operation (an approach proposed first by Trugenberger [111]). After that, the test instance is directly classified by measuring, without the explicit selection of the nearest neighbors. Conversely, the other works utilize Kaye's incrementation circuit [61] to get the distance values in basis encoding. After that, Ruan et al. [95] pick the training data samples with distances below a specified threshold through an OR gate and a projection operation to directly classify the test element, Zhou et al. [131] utilize Dürr's minimization algorithm [31] to identify the k minimum distance values, and Li et al. [65] employ a new quantum search algorithm inspired by a binary search to find the minimum.

In the realm of non-binary features, distance measures related to vector angles, such as the cosine distance, are extensively used. For example, a quantum k-NN algorithm based on a measure of this type has been exploited by Dang et al. [23] and Wang et al. [114] for image classification tasks. In detail, the SWAP test [15] without measurements is employed to calculate the distances, whose values are subsequently encoded in the qubits states via the amplitude estimation algorithm [14]. Eventually, the nearest neighbors are identified using Dürr's algorithm. This approach has been introduced first by Wiebe et al. [115], even though for retrieving only the nearest neighbor. Instead, Afham et al. [2] and Ma et al. [68] have devised a simpler variant that iterates SWAP tests and measurements to estimate a value directly proportional to the squared cosine similarity with the training instances. Notably, this model supports the parallel processing of multiple test instances [68]. In addition, Afham et al. have recently presented an alternative variant [5] that takes advantage of the SWAP test, a quantum analog-to-digital conversion algorithm [72], and a variation of Dürr's algorithm. Hence, it does not differ significantly from previously cited works.

For non-binary features, other widely used distance measures are the Euclidean, Mahalanobis, and polar distances. The Euclidean distance is treated separately in Section 3.5. Concerning the others, Gao et al. [35] have introduced a quantum k-NN employing the Mahalanobis distance, whereas Feng et al. [33] have proposed a variant using the polar distance, which combines angle and module length information through a tunable parameter. Specifically, the Mahalanobis distance is calculated using the phase estimation algorithm [19], combined with Hamiltonian simulation [90], and a controlled rotation; instead, the computation of the polar distance is performed via a SWAP test without measurements and two Toffoli gates (one of which extended). Subsequently, in both cases, the distance values are encoded into the qubits states by means of the amplitude estimation algorithm (or its coherent version, introduced by Wiebe et al. [115]). Eventually, the nearest neighbors are obtained by applying Dürr's algorithm (or an algorithm based on it, introduced by Miyamoto et al. [73]). To conclude this overview on quantum k-NN algorithms, it is worth mentioning the variant (based on a quantum sorting subroutine) introduced by Quezada et al. [88]. Specifically, it needs a metric operator calculating distances and encoding them into qubits states, an oracle identifying sorted sequences, and Grover's algorithm [41]. Similarly to other works, the test instance is directly classified without the explicit identification of the k nearest neighbors.

3.3.1 A Quantum k-NN in Detail

Let us present in detail the quantum k-NN algorithm introduced by Afham et al. [2]. To this end, let us consider the dataset $\{\mathbf{x}_i\}_{i=0,\ldots,N-1}$, where $\mathbf{x}_i \in \mathbb{R}^d$, the test data instance $\mathbf{x} \in \mathbb{R}^d$, and the fidelity, namely, the squared cosine similarity (refer to Equation 3.1), as the similarity measure. Within the amplitude encoding, the cosine similarity between $\mathbf{x}_i$ and $\mathbf{x}$ corresponds to the inner product $\langle \mathbf{x}_i | \mathbf{x} \rangle$ between their respective quantum states. Additionally, without loss of generality, let N and d be powers of 2. Now, let us take into account an index register consisting of $\log_2 N$ qubits, where the indices of the training instances are stored using the basis encoding, two n-qubit registers ($n = \log_2 d$), where the data features are encoded into the amplitudes of the quantum states, and an ancillary qubit. These four registers are initialized in the following state:

$$\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle\,|\mathbf{x}_i\rangle\;|\mathbf{x}\rangle\;|0\rangle\in\mathsf{H}_{index}\otimes\mathsf{H}_n\otimes\mathsf{H}_n\otimes\mathsf{H}_a.$$

It is worth noticing that the superposition of the training data and the test instance are stored into two distinct registers. Then, a SWAP test is executed on the two n-qubit registers, with the ancillary qubit as the control qubit, leading to the state

$$|\Psi\rangle=\frac{1}{2\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle\left[\left(|\mathbf{x}_i\rangle|\mathbf{x}\rangle+|\mathbf{x}\rangle|\mathbf{x}_i\rangle\right)|0\rangle+\left(|\mathbf{x}_i\rangle|\mathbf{x}\rangle-|\mathbf{x}\rangle|\mathbf{x}_i\rangle\right)|1\rangle\right].$$

The probability of obtaining the outcome $\alpha\in\{0,1\}$ through a measurement process applied to the ancillary qubit is given by

$$\mathbb{P}(\alpha)=\frac{1}{2}+(-1)^{\alpha}\frac{1}{2N}\sum_{i=0}^{N-1}|\langle\mathbf{x}_i|\mathbf{x}\rangle|^2,$$

and the corresponding post-measurement state is

$$|\Psi_\alpha\rangle=\frac{\sum_{i=0}^{N-1}|i\rangle\left(|\mathbf{x}_i\rangle|\mathbf{x}\rangle+(-1)^{\alpha}|\mathbf{x}\rangle|\mathbf{x}_i\rangle\right)}{\sqrt{2\left(N+(-1)^{\alpha}\sum_{i=0}^{N-1}|\langle\mathbf{x}_i|\mathbf{x}\rangle|^2\right)}}\;|\alpha\rangle\,.$$

After measuring the ancillary qubit state (α), the probability of getting the outcome i by measuring the index register is

$$\mathbb{P}(i|\alpha)=\frac{1+(-1)^{\alpha}|\langle\mathbf{x}|\mathbf{x}_i\rangle|^2}{N+(-1)^{\alpha}\sum_{i=0}^{N-1}|\langle\mathbf{x}|\mathbf{x}_i\rangle|^2}.$$

Therefore,

$$\mathbb{Q}(i):=\mathbb{P}(i|0)-\mathbb{P}(i|1)=\frac{2(|\langle\mathbf{x}|\mathbf{x}_i\rangle|^2-C)}{N(1-C^2)},\tag{3.2}$$

where $C=\frac{1}{N}\sum_i|\langle\mathbf{x}|\mathbf{x}_i\rangle|^2$ is a constant value. In practical terms, Equation (3.2) is proportional to the squared cosine similarity $|\langle\mathbf{x}_i|\mathbf{x}\rangle|^2$ between $\mathbf{x}_i$ and $\mathbf{x}$. Hence, sampling from the index register allows finding the indices of the vectors closest to $\mathbf{x}$ (they have the highest $\mathbb{Q}$ values). However, since $\mathbb{Q}$ is proportional to the square of the cosine similarity, the sign of each data feature must be consistent in order to avoid extracting also the instances most dissimilar to $\mathbf{x}$.

3.4 Quantum Cosine Binary Classifier

Recently, Pastorello and Blanzieri have introduced a quantum binary classifier that relies on the cosine similarity metric [79]. The classifier has a simple iterative structure, which involves the preparation of a superposition state in which the training and test features are encoded as amplitudes, a SWAP test that acts on single qubit states, and a final measurement. In this way, a probability value proportional to a weighted label assignment, with the weights given by the cosine similarity, is estimated.

In detail, let us consider a training set $X=\{\mathbf{x}_i,y_i\}_{i=0,...,N-1}$, where $\mathbf{x}_i\in\mathbb{R}^d$ and $y_i\in\{-1,1\}\,\forall i\in\{0,...,N-1\}$. Hence, X comprises N data instances characterised by d real features and two-valued labels. Additionally, let $\mathbf{x}\in\mathbb{R}^d$ be a novel (test) data instance that needs to be classified as -1 or 1. Then, let us consider the following (classical) classification model:

$$y(\mathbf{x}):=\operatorname{sgn}\left(\sum_{i=0}^{N-1}y_i\cos(\mathbf{x}_i,\mathbf{x})\right),\tag{3.3}$$

with $\cos(\mathbf{x}_i,\mathbf{x}):=\frac{\mathbf{x}_i\cdot\mathbf{x}}{\|\mathbf{x}_i\|\|\mathbf{x}\|}$ being the cosine similarity between the training vector $\mathbf{x}_i$ and $\mathbf{x}$. In this model, every training vector plays a role in predicting the new label, with the contributions being weighted by the cosine similarity to the test instance. Now, let us take into account a $\log_2 N$-qubit register for encoding the indices of the training data vectors, an n-qubit register ($n = \log_2 d$) for storing the data features using the amplitude encoding, and a single qubit for representing the labels according to $b_i = \frac{1-y_i}{2} \in \{0,1\}$. At this point, let us consider the state

$$|X\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle |\mathbf{x}_i\rangle |b_i\rangle \in \mathsf{H}_{index} \otimes \mathsf{H}_n \otimes \mathsf{H}_l, \tag{3.4}$$

where H_l is the Hilbert space of the label qubit. The above state encodes the training set X, including features and labels, as a quantum superposition. Additionally, within the same registers, let us consider the state

$$|\psi_{\mathbf{x}}\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle |\mathbf{x}\rangle |-\rangle \in \mathsf{H}_{index} \otimes \mathsf{H}_n \otimes \mathsf{H}_l, \tag{3.5}$$

with the label qubit being in state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. As a result, the test label is initially in a quantum superposition of the two possible classes. In order to allow the states (3.4) and (3.5) to coexist within the same registers, let us introduce an ancillary qubit (a). The initial state is then the following:

$$\frac{1}{\sqrt{2}} \left(|X\rangle |0\rangle + |\psi_{\mathbf{x}}\rangle |1\rangle\right) \in \mathsf{H}_{index} \otimes \mathsf{H}_n \otimes \mathsf{H}_l \otimes \mathsf{H}_a. \tag{3.6}$$

After preparing the initial state, let us perform a SWAP test on the qubit a and a second ancillary qubit (b) initialized in $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. The control qubit is another ancillary qubit (c) initialized in $|0\rangle$. The probability of measuring 1 on the control qubit after the SWAP test is

$$\mathbb{P}(1) = \frac{1}{4}(1 - \langle X|\psi_{\mathbf{x}}\rangle),$$

which is strictly related to the classification model taken into account. Indeed,

$$\langle X|\psi_{\mathbf{x}}\rangle = \frac{1}{N\sqrt{2}} \sum_{i=0}^{N-1} y_i \cos(\mathbf{x}_i, \mathbf{x}).$$

As a consequence, given $\mathbb{P}(1)$ or an estimate of it, the label of $\mathbf{x}$ can be predicted (according to 3.3) as

$$y(\mathbf{x}) = \operatorname{sgn}\left[1 - 4\,\mathbb{P}(1)\right]. \tag{3.7}$$

3.5 Quantum Euclidean Distance

The Euclidean distance is a widely used distance metric in machine learning. Here, the definition of its squared version is supplied. Specifically, let us consider two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$. The squared Euclidean distance between them, denoted as $d^2(\mathbf{u}, \mathbf{v})$, is given by

$$d^2(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|^2 = \|\mathbf{u}\|^2 - 2\langle \mathbf{u}, \mathbf{v}\rangle + \|\mathbf{v}\|^2, \tag{3.8}$$

with $\langle \mathbf{u}, \mathbf{v}\rangle$ being the scalar product between $\mathbf{u}$ and $\mathbf{v}$.

The aforementioned distance metric has been exploited also in the field of QML. For example, Lloyd et al. [66] have introduced a quantum procedure for estimating the squared Euclidean distance between a data point and a cluster centroid, namely, the mean of the elements within a data group. In particular, this algorithm is based on the SWAP test, which is performed on the index registers, and does not need input vectors with unit norms. An analogous approach has been employed by Sarma et al. [96] to develop a hybrid k–means clustering algorithm, where the centroids are classically computed, and by Getachew et al. [36] for a hybrid version of the k-medians algorithm. In addition, Yu et al. [123] have introduced three quantum procedures for the estimation of different similarity measurements, all based on the squared Euclidean distance, on sets of data. These procedures do not need unit-norm input vectors, leverage the quantum interference resulting from the change of basis, and employ the amplitude estimation algorithm for determining the similarity measures. Eventually, it is worth recalling the quantum binary classifier proposed by Schuld et al. [98]. Specifically, the quantum circuit of the classifier comprises a Hadamard gate (required for the quantum interference), a conditional measurement, and a final measurement. The repeated execution of the circuit allows estimating a probability value related to the squared Euclidean distances for each of the two classes. However, only input vectors with unit norms are taken into account.

Concerning quantum k-NN models, to the best of the author's knowledge, the sole variant available in the literature relying on the Euclidean distance has been proposed by Fastovets et al. [32]. The variant in question leverages the procedure introduced by Lloyd et al. [66] for estimating the pairwise distance values and employs Dürr's minimization algorithm for identifying the k nearest neighbors. A notable drawback of this method consists in the necessity of multiple iterations of each of the two steps, as both of them include a final measurement. Additionally, Dürr's algorithm needs an oracle, namely, a black-box function, to be executed. Actually, the nearest neighbor algorithm introduced by Wiebe et al. [115] accepts also the Euclidean distance as the distance metric. Nevertheless, its workflow (described in Section 3.3) is considerably complex to be realized. Eventually, the calculation of the single linkage, i.e., a set similarity measure taken into account by Yu et al. [123], could be considered as a generalisation of the nearest neighbor search. Nonetheless, their quantum algorithm utilizes the reciprocals of the input vectors, leading (theoretically) to the loss of the original distance relationships.

Part I

Quantum Annealing

4 QALS Empirical Evaluation

This chapter is a reworked version of the article "Quantum annealing learning search implementations" [12], which was motivated by the absence of an empirical evaluation of QALS (see Section 2.3) in the original work by Pastorello and Blanzieri [81]. In practice, in this chapter, two implementations of QALS and their empirical evaluation on two different real-world problems are presented. Specifically, the implementations are in C++ and Python, respectively. The former aims at improving the performance of the classical part of the algorithm, while the latter is characterised by a smooth interaction with the quantum annealers provided by D-Wave. Concerning the problems considered, they are the Number Partitioning Problem, which has a natural QUBO representation [63], and the Travelling Salesman Problem, which has constraints that must be included as penalties in the QUBO formulation. The algorithm has been compared with classical competitors and with the tools provided by D-Wave. The results have demonstrated that QALS can tackle bigger problems with respect to the standard D-Wave solutions.

4.1 Implementations

Two implementations of QALS, in C++ and in Python, have been developed. In both cases, some optimizations with respect to the original mathematical formulation of QALS have been required. In addition, in contrast to the theoretical formulation presented in Section 2.3, the QUBO problem variables are defined in the $\{0, 1\}$ domain, which is the standard domain for QUBO problems. The pseudocode, which is valid for both implementations, is provided in Algorithm 2.

First of all, in order to avoid the usage of the quantum annealer while testing the correctness of the implementations (the machine time at our disposal was limited), the call to the annealer has been substituted with an exhaustive search on the same equation minimized by D-Wave. It is important to notice that the minimization of the equation in question does not simulate the actual behavior of a quantum annealer, since it is a probabilistic machine.

After the correctness tests, it was evident that the execution was notably slow. The main cause was the calculation of the product of three matrices, i.e., the transposed permutation matrix P^T, the QUBO matrix Q, and the permutation matrix P. Indeed, the computation of the product P^TQP could require up to 100 seconds. Given that the permutation matrix P is a sparse matrix, a sparse matrix representation has been adopted first. In practice, only the non-zero values are stored as triplets (*row*, *column*, *value*). In this way, the execution time for this calculation was reduced to approximately 1.5 seconds, but further optimizations were feasible.

To address this efficiency issue, both the representation of P and the way to permute the matrix Q have been modified. Specifically, P has been substituted with a permutation vector *perm*. Let P_i denote the i-th row in P, then:

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad perm = [3, 0, 4, 1, 2]$$

$$\begin{aligned} &perm[0] = 3 \rightarrow P_0 \text{ has the value 1 in column 3} \\ &perm[1] = 0 \rightarrow P_1 \text{ has the value 1 in column 0} \\ &perm[2] = 4 \rightarrow P_2 \text{ has the value 1 in column 4} \\ &perm[3] = 1 \rightarrow P_3 \text{ has the value 1 in column 1} \\ &perm[4] = 2 \rightarrow P_4 \text{ has the value 1 in column 2.} \end{aligned}$$

```
Data: Matrix $Q$ of order $n$ encoding a QUBO problem, annealer adjacency matrix $\mathcal{A}$ of order $n$
Input: Permutation modification and mapping function $g$, minimum probability $p_\delta$ of permutation
       modification, probability decreasing rate $\eta$, candidate perturbation probability $q$, number $N$ of
       iterations at constant $p$, scaling factor $\lambda_0$, number of annealer runs $k$, termination parameters
       $i_{max}$, $N_{max}$, $d_{min}$
Result: $\mathbf{x}^*$ vector with $n$ elements in $\{0,1\}$ solution of the QUBO problem
function $f_Q(Q,\mathbf{x})$:
    return $\mathbf{x}^TQ\mathbf{x}$;
$m = 0_n$;
for $i \leftarrow 0$ to $n$ do
    $m[i] \leftarrow i$;
end
$p \leftarrow 1$;
$\Theta_1, m_1 \leftarrow g(Q, \mathcal{A}, m, p)$;
$\Theta_2, m_2 \leftarrow g(Q, \mathcal{A}, m, p)$;
$\mathbf{x}_1 \leftarrow mapback(annealer(\Theta_1, k), m_1)$;
$\mathbf{x}_2 \leftarrow mapback(annealer(\Theta_2, k), m_2)$;
$f_1 \leftarrow f_Q(Q, \mathbf{x}_1); f_2 \leftarrow f_Q(Q, \mathbf{x}_2)$;
if $f_1 < f_2$ then
    $\mathbf{x}^* \leftarrow \mathbf{x}_1$; $f^* \leftarrow f_1$; $m^* \leftarrow m_1$; $\mathbf{x}' \leftarrow \mathbf{x}_2$;
else
    $\mathbf{x}^* \leftarrow \mathbf{x}_2$; $f^* \leftarrow f_2$; $m^* \leftarrow m_2$; $\mathbf{x}' \leftarrow \mathbf{x}_1$;
end
if $f_1 \neq f_2$ then
    $S \leftarrow \mathbf{x}' \otimes \mathbf{x}' - I_n + diag(\mathbf{x}')$;
else
    $S \leftarrow 0_{n\times n}$;
end
$e \leftarrow 0$; $d \leftarrow 0$; $i \leftarrow 0$; $\lambda \leftarrow \lambda_0$;
repeat
    $Q' \leftarrow Q + \lambda S$;
    if $N$ divides $i$ then
        $p \leftarrow p - \eta(p - p_\delta)$;
    end
    $\Theta', m \leftarrow g(Q', \mathcal{A}, m^*, p)$;
    $\mathbf{x}' \leftarrow mapback(annealer(\Theta', k), m)$;
    with probability $q$: $\mathbf{x}' \to h(\mathbf{x}', p)$;
    if $\mathbf{x}' \neq \mathbf{x}^*$ then
        $f' \leftarrow f_Q(Q, \mathbf{x}')$;
        if $f' < f^*$ then
            $swap(\mathbf{x}', \mathbf{x}^*)$; $f^* \leftarrow f'$; $m^* \leftarrow m$; $e \leftarrow 0$; $d \leftarrow 0$;
            $S \leftarrow S + \mathbf{x}' \otimes \mathbf{x}' - I_n + diag(\mathbf{x}')$;
        else
            $d \leftarrow d + 1$;
            with probability $(p - p_\delta)^{(f'-f^*)}$:  $swap(\mathbf{x}', \mathbf{x}^*)$; $f^* \leftarrow f'$; $m^* \leftarrow m$; $e \leftarrow 0$;
        end
        $\lambda \leftarrow \min\left\{\lambda_0, \frac{\lambda_0}{2+i-e}\right\}$;
    else
        $e \leftarrow e + 1$;
    end
    $i \leftarrow i + 1$;
until $i = i_{max}$ or $(e + d \geq N_{max}$ and $d < d_{min})$;
return $\mathbf{x}^*$;
```

Algorithm 2: Implementation of QALS (taken from [12]).

While the permutation of the matrix P has a computational complexity of $O(n^2)$, the permutation of the vector *perm* has a complexity of $O(n \log n)$, which is determined by $O(n)$ accesses to the map m (each one with complexity $O(\log n)$). The usage of a hashmap would further reduce the time complexity to $O(n)$ on average.

After the change of representation, the way to permute only the elements that must be mapped into the D-Wave topology has been investigated. Let us take into account an example matrix Q (that is not a QUBO matrix), with the permutation matrix and vector from the previous example:

$$Q = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad perm = [3, 0, 4, 1, 2].$$

The matrix resulting from the permutation (P^TQP) is

$$M = \begin{bmatrix} 7 & 9 & 10 & 6 & 8 \\ 17 & 19 & 20 & 16 & 18 \\ 22 & 24 & 25 & 21 & 23 \\ 2 & 4 & 5 & 1 & 3 \\ 12 & 14 & 15 & 11 & 13 \end{bmatrix}.$$

Now, let us examine the positions of the entries after the permutation. The objective is to find the element m_{ij} without generating the entire matrix M. In practical terms, the indices of the values i and j in the permutation vector *perm* should be exploited to find the correct entry in the Q matrix. For example, to find the value for the entry $m_{1,2}$, the indices of the values "1" and "2" in the permutation vector must be identified: the index of "1" is "3"; the index of "2" is "4". The first index designates the row, whereas the second one designates the column. Consequently, the entry $m_{1,2}$ corresponds to the entry $q_{3,4}$, which is equal to 20.

Searching for the index of a specific value in the permutation vector has a complexity of $O(n)$. In order to generate the matrix Θ, a total of $O(n)$ searches is necessary, as D-Wave's Pegasus topology A is characterised by $O(n)$ edges (with n being the number of nodes in the considered Pegasus' subgraph [55]). Hence, the overall complexity is $O(n^2)$. Nevertheless, considering that the goal is to find the index of multiple values, a more efficient approach consists in inverting indices and values. This operation can be performed by executing a $O(n)$ algorithm once at the beginning. As a consequence, the complexity of the search operation becomes $O(1)$. Let us examine the *inverse* vector for the previous example:

$$perm = [3, 0, 4, 1, 2] \qquad inverse = [1, 3, 4, 0, 2]$$

$$\begin{aligned} inverse[0] &= 1 \rightarrow perm \text{ has the value 0 in position 1} \\ inverse[1] &= 3 \rightarrow perm \text{ has the value 1 in position 3} \\ inverse[2] &= 4 \rightarrow perm \text{ has the value 2 in position 4} \\ inverse[3] &= 0 \rightarrow perm \text{ has the value 3 in position 0} \\ inverse[4] &= 2 \rightarrow perm \text{ has the value 4 in position 2.} \end{aligned}$$

After obtaining an estimate of the solution from the quantum annealer, the solution variables must be mapped back to the original unpermuted problem space. Due to the change of representation of P, the P^Tz operation is no longer valid. However, by leveraging the inverted permutation vector *inverse*, the original values can be retrieved as $x\_back[i] \leftarrow x[inverse[i]]$. The pseudocode for this process is given in Algorithm 3.

4.1.1 C++ Implementation

In developing the C++ implementation of QALS, two main issues have been encountered, namely, the lack of native D-Wave APIs and the selection of a suitable random number generator.

```
Input: Solution vector $x$, permutation vector $perm$
Result: $perm^{-1}(x)$
$inverse \leftarrow compute\_inverse(perm)$;        // computes the inverse of $perm$
$x\_back \leftarrow new\ int[n]$;
for $i \leftarrow 0$ to $n - 1$ do
    $x\_back[i] \leftarrow x[inverse[i]]$;    // $x\_back[i]$ takes the value contained in $x[inverse[i]]$,
     so that the values are mapped back to the original unpermuted space
end
return $x\_back$
```

Algorithm 3: Map back variables (*map_back*(x, *perm*) function) (taken from [12]).

4.1.1.1 Lack of Native APIs

At the time of writing, D-Wave offers only Python APIs to interact with the QPUs. As a consequence, the retrieval of the current QPU topology, the submission of problems, and the acquisition of the corresponding results are not feasible for a C++ program. To address these issues, two approaches have been considered.

The first approach involves embedding a Python function in C++ [28]. This can be achieved by including the *Python.h* header and, after generating all necessary *PyObjects*, running the Python function that encapsulates the calls to the annealer. However, the Leap IDE, where the experiments have been executed, does not supply the *Python.h* header.

In the second approach, the one that has been employed, the C++ executable creates a child Python process through a fork operation at the startup. The role of this process is twofold, namely, to provide the parent process with the current QPU topology and to submit the problems to the QPU solver. Specifically, the Python process first obtains the current topology and transmits it to the C++ process through an anonymous pipe. This step is essential since the quantum annealer might have inactive nodes, which must be taken into account during the embedding. After the reception of the topology, the parent process begins forwarding problems to the child process. In detail, the parent process transmits the row index, the column index, and the corresponding value for each entry in the Θ matrix. Eventually, it sends a message containing a "#" character to signal that all entries of the matrix have been transmitted. In the meanwhile, the child process consistently reads the input pipe, storing all the information. Upon receiving the "#" character, it submits the problem to the QPU, fetches the solution, and transmits it to the C++ process. Subsequently, it resumes reading the input pipe for new instructions.

Algorithms 4 and 5 provide the pseudocode for two functions, i.e., *init_child* and *send_to_annealer*, that implement the C++ side of the approach just described. Specifically, the *init_child* function generates the array of arguments to be transmitted to the Python process (using *dup2*), redirects the standard input and output to pipes, and substitutes the executable code with the Python code. *READ* (=0) and *WRITE* (=1) denote positions in the fd array and are used to identify the sides of the pipe. In detail, $fd[READ]$ is allocated for child reading, $fd[WRITE]$ for parent writing, $fd[READ + 2]$ for parent reading, and $fd[WRITE + 2]$ for child writing. Instead, *send_to_annealer* is responsible for transmitting each row-column-value triplet to the Python process and retrieving the solution from the pipe.

```
Input: Number of measurements for each problem $k$ ($num\_reads$)
$args \leftarrow \{"python", "solver.py", to\_string(k)\}$;
$dup2(fd[READ], STDIN\_FILENO)$;            // replace the standard input with an
 anonymous pipe
$dup2(fd[WRITE + 2], STDOUT\_FILENO)$;    // replace the standard output with an
 anonymous pipe
$close\_pipes(fd)$;
$execvp(args[0], args)$;                    // replace the child's executable code
```

Algorithm 4: Initialization of the Python child process (*init_child*(k) function) (taken from [12]).

```
Input: Weights $\Theta$ ($\theta_{edge}$ refers to the weight associated to the edge edge)
Result: Vector with minimum estimated energy $x$
% iterate over all edges, including pairs in the form $(i, i)$;
foreach edge in $\Theta$ do
    $row \leftarrow edge.u()$;                                    // store vertex $u$
    $col \leftarrow edge.v()$;                                    // store vertex $v$
    $val \leftarrow \theta_{edge}$;            // store weight associated to edge $(u, v)$
    $write(fd[WRITE], row)$;                                      // send row index
    $write(fd[WRITE], col)$;                                   // send column index
    $write(fd[WRITE], val)$;                                          // send value
end
$write(fd[WRITE], "\#")$;                             // notify end of transmission
for $i \leftarrow 0$ to $n - 1$ do
    $x[i] \leftarrow read(fd[READ + 2])$;
end
return $x$
```

Algorithm 5: Send Θ to the Python process and retrieve the estimated solution x ($send\_to\_annealer(\Theta)$ function) (taken from [12]).

Concerning the time required by the message exchange, it corresponds to approximately one second when using 5000 variables, which is nearly the limit of the Pegasus architecture. This value has been obtained by measuring the time elapsed between the initial message sent by the C++ process and the final message received by the same process, without actually invoking the annealer (the response has been generated and sent by the Python process). While this duration is not exaggerated, it may nullify the benefit of using C ++ over Python.

4.1.1.2 Random Number Generation

One of the key components of QALS is the shuffling of the map m (see Algorithm 1). In particular, the permutations of m have been generated as shown in Algorithm 6. The algorithm in question relies on the shuffling of the map's keys (line 6). Ideally, the shuffle algorithm should be capable of generating all the $n!$ permutations of the *keys* vector. In this work, the Fisher-Yates shuffle algorithm, which produces unbiased permutations, has been used.

The Fisher-Yates algorithm necessitates a random number generator. The first option that has been taken into account is the *rand()* function provided by C++. In detail, the *rand()* function provides values between 0 and *RAND_MAX*. Hence, if the generated numbers are restricted to the desired range by means of the modulo operation, the values do not necessarily have equal probability. Let us consider an example [112]. Specifically, let us assume that the desired range is $[0, 2]$ and

```
   Input: Map $m$ ($m_k$ is the value of $m$ for key $k$)
   Result: Permuted map $m$
 1 $shuffled \leftarrow map()$;
 2 $keys \leftarrow new\ int[n]$;
 3 foreach $k$ in $m.keys$ do
 4     $keys.append(k)$;                              // create a vector of keys
 5 end
 6 $shuffle\_vector(keys)$;                          // shuffle the vector of keys
 7 $it \leftarrow keys.begin()$;                                       // iterator
 8 for $pair$ in $m$ do
 9     $shuffled_{pair.key} \leftarrow m_{*it}$;
10     $it.next()$;
11 end
```

Algorithm 6: Shuffle map (*shuffle(m)* function) (taken from [12]).

$RAND\_MAX = 10$, thus:

- if *rand()* yields a value in $\{0, 3, 6, 9\}$, $rand()\%3 = 0$. Hence, $P(0) = \frac{4}{11}$;
- if *rand()* yields a value in $\{1, 4, 7, 10\}$, $rand()\%3 = 1$. Hence, $P(1) = \frac{4}{11}$;
- if *rand()* yields a value in $\{2, 5, 8\}$, $rand()\%3 = 2$. Hence, $P(2) = \frac{3}{11}$.

In practical terms, the probability of the numbers between 0 and 2 is not the same. Another issue was represented by the length of the random number generator period. For instance, to generate all possible permutations of a deck of 52 cards (52!), a period of at least 52! is necessary. Typically, the period of *rand()* is 2^{32}, although it can vary depending on the implementation. Given that QALS involves the permutation of vectors with more than 5000 variables, *rand()* was not a good choice.

Another solution that has been considered is the generation of random numbers by means of D-Wave's quantum annealers. Given that quantum annealers are "trusted" quantum machines, the generated numbers would be truly random. In particular, the idea consists of the following steps: submitting problems wherein each qubit has an equal probability of collapsing to either 0 or 1; decoding the resulting arrays of boolean values into one or more integer values. The pseudocode for this approach is provided in Algorithm 7. In essence, line 3 sets all the diagonal values in Θ to zero. Consequently, all 2^n possible states have an equal probability of being retrieved [56]. More in detail, any call to this algorithm allows generating $\lfloor n/k \rfloor$ integer numbers, where n represents the number of qubits used, and k denotes the number of bits per number. For instance, let us assume that the desired range of values is $[0, 127]$ and $n = 5000$. A single call would produce 714 integers ($\lfloor 5000/7 \rfloor$) in approximately 0.3 seconds. While this may seem reasonable, the cost is excessive:

- each QALS iteration requires to permute a vector whose maximum size is approximately 5436. Therefore, for each iteration, it is necessary to generate at most 5436 numbers. In a single run, with $n = 5436$ and $k = 13$, Algorithm 7 produces 418 numbers. As a consequence, for each iteration, the algorithm must be run 13 times;
- each run of Algorithm 7 takes approximately 0.3 seconds on average. Consequently, the overhead would be approximately 3.9 seconds for each QALS iteration.

This implies that the quantum annealer would be invoked 13 times more, reducing the number of possible experiments and increasing the overall cost. Clearly, this approach was not feasible.

In the end, the Mersenne Twister random number generator, which is characterised by a period of $2^{19937} - 1$, has been selected. Specifically, the Mersenne Twister is able to generate 64-bit floating point random numbers faster than the hardware-implemented Intel Secure Key [94]. In addition, while a large period does not ensure quality in random number generation, it allows the generation of longer sequences. Although the period in question is not large enough to cover all possible sequences, the Mersenne Twister represents the best tradeoff that has been found between efficiency and coverage of the permutation space.

```
Input: Amount of bits reserved per number $k$
Result: Vector of random integers $nums$
1 $\Theta \leftarrow \{\}$;                        // Python dictionary
2 for $i \leftarrow 0$ to $n - 1$ do
3     $\Theta[i][i] \leftarrow 0$;                 // initialize the diagonal with all zeros
4 end
5 $x \leftarrow sample\_qubo(\Theta)$;             // run the annealer with $\Theta$
6 $x \leftarrow x.first.sample.values()$;          // solution vector $x \in \{0,1\}^n$
7 $nums \leftarrow to\_decimal(x, k)$;   // extract $\lfloor n/k \rfloor$ decimal numbers, each one from 0 to
    $2^k - 1$
8 return $nums$
```

Algorithm 7: Generation of random numbers by quantum annealing ($gen(k)$ function) (taken from [12]).

4.1.2 Python Implementation

In developing the Python implementation, no particular issue has been encountered. Here, the embedding procedure and the submission of problems to the quantum annealer are described.

4.1.2.1 Embedding Procedure

In the QALS scheme, the embedding of the QUBO matrix into the annealer topology is a straightforward process. Indeed, each index of the QUBO matrix is associated with a *node* $\in V$, with V being the list of active nodes in the topology, and the matrix entries are mapped accordingly. Nevertheless, the topology graph is not complete, and some qubits are unavailable (only 97% of qubits are generally available). Hence, first of all, it is necessary to retrieve the list of working qubits and edges using a function provided by D-Wave. In practice, rather than D-Wave's *EmbeddingComposite* class, which is effective for matrices of size up to 196 × 196, a custom embedding algorithm is employed in QALS implementations. The pseudocode for the embedding algorithm, which is used also by the C++ implementation, is provided in Algorithm 8. The algorithm works as follows:

1. an empty dictionary is initialized using the necessary active nodes as keys;

2. each key (active node) is associated with the list of nodes that are endpoints of active edges outgoing from the considered node (these nodes must be keys of the same dictionary);

3. a support dictionary mapping each selected node to a QUBO matrix index is created;

4. the QUBO matrix is embedded by iterating on the nodes (corresponding to matrix rows) and their adjacent nodes (corresponding to matrix columns).

```
Input: Sampler $sampler$, QUBO problem $Q$, dimension of QUBO problem $n$
Result: Embedded QUBO problem $mapped$
$actives \leftarrow dict()$;
foreach $node \in sampler.nodelist$ do
    $actives[node] \leftarrow list()$;
    if $actives.keys().size() == n$ then
        break;
    end
end
foreach $edge \in sampler.edges$ do
    if $edge.node1 \in actives.keys()$ and $edge.node2 \in actives.keys()$ then
        $actives[edge.node1].append(edge.node2)$;
        $actives[edge.node2].append(edge.node1)$;
    end
end
$support_n \leftarrow dict()$;
$i \leftarrow 0$;
foreach $node \in actives.keys().ordered()$ do
    $support[node] = i$;
    $i \leftarrow i + 1$;
end
$mapped_{n \times n} \leftarrow 0$;
foreach $node \in actives.keys()$ do
    foreach $adjnode \in actives[node]$ do
        $mapped[node][adjnode] = Q[support[node]][support[adjnode]]$;
    end
end
return $mapped$;
```

Algorithm 8: Embedding the problem in the sampler topology (taken from [12]).

```
Input: Dictionary representing the embedding of the problem in the topology $\Theta$, sampler
       $sampler$, number of reads $k$
Result: List of $\{0,1\}^n$
$response = sampler.sample\_qubo(\Theta, num\_reads = k)$;
return $response.first.sample.values()$;
```

Algorithm 9: Communication with the annealer (problem submission) (taken from [12]).

4.1.2.2 Communication with the Annealer

D-Wave supplies different samplers for solving problems, and the signature of the sampling method is common to all samplers. The pseudocode of the algorithm used for submitting problems to the quantum annealer is provided in Algorithm 9. Its structure is really simple, but it is worth highlighting some aspects: Θ must be a dictionary, otherwise the problem will not be correctly processed by the sampler; k denotes the number of annealer reads (as in Algorithm 2). The k parameter is optional for both simulated and quantum annealing samplers, while hybrid samplers do not support it.

4.2 Empirical Evaluation

QALS has been empirically evaluated on two problems, namely, the Number Partitioning Problem (NPP) and the Travelling Salesman Problem (TSP). Specifically, for NPP, a single run has been executed due to the limited quantum annealing time available. Conversely, for TSP, multiple runs have been executed due to the less QPU usage per run. Here is reported a legend for the parameters of QALS:

- p_δ denotes the minimum probability of permutation modification;
- η represents the probability decreasing rate;
- q corresponds to the candidate perturbation probability;
- N is the number of iterations at constant probability;
- λ_0 denotes the initial balancing factor for the tabu matrix;
- k represents the number of annealer measurements for each problem;
- N_{max} represents the maximum number of consecutive times that QALS can find the current best solution or a solution that is not better than it before stopping;
- d_{min} corresponds to a further condition on the number of times that QALS can find a solution not better than the current best solution before stopping.

4.2.1 Number Partitioning Problem (NPP)

The Number Partitioning Problem (NPP) consists in dividing a set of numbers into two subsets so that the difference between the sum in the first subset and the sum in the second subset is minimized. The QUBO formulation that has been utilized in the experiments is the one provided by Glover et al. [38]. Specifically, let us consider a set of numbers $S = \{s_1, s_2, ..., s_n\}$. If s_i belongs to the first subset, $x_i = 1$; otherwise, $x_i = 0$. Therefore, the sum of the first subset values is equal to $sum_1 = \sum_{i=1}^{n} s_i x_i$, while the sum of the second subset values can be expressed as $sum_2 = \sum_{i=1}^{n} s_i - \sum_{i=1}^{n} s_i x_i$. Then, let c be $\sum_{i=1}^{n} s_i$. Hence, the difference between the two sums is equal to

$$diff = \sum_{i=1}^{n} s_i - 2\sum_{i=1}^{n} s_i x_i = c - 2\sum_{i=1}^{n} s_i x_i. \tag{4.1}$$

Rather than directly minimizing the difference, let us consider its squared value, which can be expressed as

$$diff^2 = \left\{ c - 2\sum_{i=1}^{n} s_i x_i \right\}^2 = c^2 + 4x^t Q x, \tag{4.2}$$

where Q is the QUBO matrix, whose entries are given by

$$Q_{ii} = s_i\,(s_i - c) \quad Q_{ij} = Q_{ji} = s_i s_j. \tag{4.3}$$

Since the constants (c^2 and 4) can be ignored, the QUBO problem can be expressed as

$$QUBO : min\ y = x^t Q x.$$

The pseudocode for the calculation of the QUBO matrix entries is provided in Algorithm 10.

```
Input: Set of numbers $s$
Result: Matrix $Q$ representing the QUBO formulation of the NPP problem defined by $s$
$n \leftarrow s.size()$;
$c \leftarrow sum(s)$; % sum over numbers in $s$
$Q \leftarrow new\ int[n][n]$;
for $i \leftarrow 0$ to $n-1$ do
    for $j \leftarrow 0$ to $n-1$ do
        if $i \neq j$ then
            $p \leftarrow s[i] \cdot s[j]$;
            $Q[i][j] \leftarrow p$;
            $Q[j][i] \leftarrow p$;
        else
            $Q[i][i] \leftarrow s[i] \cdot (s[i] - c)$;
        end
    end
end
return $Q$
```

Algorithm 10: Translation of NPP into QUBO (taken from [12]).

4.2.1.1 Classical Algorithms for NPP

Due to the NP-hard nature of NPP, an efficient algorithm that solves it does not exist (yet). However, various heuristics have been proposed. An example is the greedy heuristic, which consists in arranging the numbers in descending order and iteratively adding them to the subset with the smaller cumulative value. Another heuristic is the Karmarkar-Karp (KK) approach [60], which is the basis of an exact (exponential-time) algorithm presented by Korf [64] and evaluated by Pedroso and Kubo [86], namely, the Complete Karmarkar-Karp algorithm (CKK).

The CKK algorithm is based on a depth-first search of a binary tree. In this tree structure, the left branch represents the replacement of the two largest numbers at the present level with the absolute value of their difference, implying that these numbers are placed in different subsets. Conversely, the right branch represents their replacement them with their sum, implying that they are assigned to the same subset. Since the numbers are not assigned to a specific subset during the search, a linear time procedure must be run at the end of the search in order to retrieve the resulting subsets. The algorithm's worst-case complexity is exponential. However, the search terminates upon finding a perfect partition, namely, the last remaining number, representing the difference between the two subsets, is either 0 or 1. Additionally, if the biggest number is bigger than the sum of all the others, the branching for that sub-tree can be terminated, as all the other numbers can be assigned to the same subset. Finally, when dealing with four numbers or fewer, only the left branch is evaluated, as the KK heuristic is exact in this situation.

In the experiments presented here, the CKK implementation provided by Pedroso and Kubo [86] has been used as a comparison.

4.2.1.2 Experimental Setup and Results

The experiments have been executed sequentially on the Leap cloud, using the QALS parameters reported in Table 4.2. In this case, the C++ implementation has been used to obtain the results, while

Table 4.1: Tests performed on the Number Partitioning Problem (taken from [12]).

Dimension	Approach	Range	Sets Difference	Time (s)	# Iterations
500	QALS	100	93	2172	4000
	Hybrid		1	6	-
	Classical		1	0.003	-
	QALS	1000	292	2157	4000
	Hybrid		4	9	-
	Classical		0	0.005	-
	QALS	10000	2640	2151	4000
	Hybrid		36	12	-
	Classical		0	0.004	-
	QALS	1000000	475860	1992	4000
	Hybrid		2340912	14	-
	Classical		0	0.005	-
1200	QALS	1000	185	1327	2000
	Hybrid		1	23	-
	Classical		1	0.015	-
	QALS	10000	6337	1304	2000
	Hybrid		225	23	-
	Classical		1	0.017	-
	QALS	100000	145982	1270	2000
	Hybrid		186624	23	-
	Classical		0	0.024	-
	QALS	1000000	303833	1267	2000
	Hybrid		781440	23	-
	Classical		0	0.019	-
2500	QALS	1000	3108	2442	2000
	Hybrid		0	62	-
	Classical		0	0.072	
	QALS	10000	11681	2666	2000
	Hybrid		25	54	-
	Classical		1	0.090	-
	QALS	100000	160676	2354	2000
	Hybrid		6240	64	-
	Classical		1	0.089	-
	QALS	1000000	2731518	2341	2000
	Hybrid		1151232	65	-
	Classical		1	0.093	-
5436	QALS	1000	6209	9414	2000
	Hybrid		1	260	-
	Classical		1	0.318	-
	QALS	10000	528	9413	2000
	Hybrid		16	267	-
	Classical		0	0.437	-
	QALS	100000	4010004	9168	2000
	Hybrid		12112	263	-
	Classical		0	0.464	-
	QALS	1000000	5497085	9507	2000
	Hybrid		24576	260	-
	Classical		0	0.479	-

Table 4.2: Values used for QALS parameters in all NPP tests (taken from [12]).

p_δ	η	q	N	λ_0	k	N_{max}	d_{min}
0.1	0.01	0.2	10	1.5	10	100	70

the Python implementation has been employed to verify them. Regarding the annealing parameters, like the annealing time and schedule, the default values have been employed for QALS (the system that has been used is the Advantage 1.1) [48]. Instead, Hybrid internally manages these properties in an automatic way.

The obtained results are shown in Table 4.1. Specifically, *Range* denotes the upper boundary of the number generation interval ([1, *Range*]) and, therefore, the maximum possible s_i value. For example, if *Dimension* is 500 and *Range* is 100, a vector of 500 integers within the range $[1, 100]$ is employed. Conversely, *Sets Difference* represents the difference between the two resulting sets. In detail, QALS has not performed as well as expected. Indeed, Hybrid has turned out to be superior in terms of both results achieved and time required. In addition, the classical algorithm considered for comparison (the CKK algorithm) has exhibited notable speed and has consistently found the optimal solution. In practice, classical algorithms for NPP are efficient despite the problem being NP-Hard. QALS performance could be probably improved by increasing the number of annealer measurements ($k \gg 10$). Indeed, a larger output sample would increase the probability of finding a solution close to the optimum (the D-Wave quantum annealer is characterised by noise and temperature effects). However, the quantum annealing time at our disposal was too short.

4.2.2 Travelling Salesman Problem (TSP)

The Travelling Salesman Problem, commonly referred to as TSP, consists in determining the shortest route among n cities, given a list of cities and their pairwise distances (actually, a city is not required to be directly connected to all the others). In particular, each city must be visited exactly once and the route must conclude at the starting point. Therefore, TSP is equivalent to the task of identifying, within a graph $G = (V, E)$, the Hamiltonian cycle characterised by the minimum cumulative weight. This perspective is particularly significant as there exists a QUBO formulation for it [67]. In detail, the problem Hamiltonian for TSP is defined as

$$\begin{gathered}
H = H_A + H_B \\
H_A = A\sum_{i=1}^{n}\left(1-\sum_{j=1}^{n} x_{i,j}\right)^2 + A\sum_{j=1}^{n}\left(1-\sum_{i=1}^{n} x_{i,j}\right)^2 + A\sum_{(uv)\notin E}\sum_{j=1}^{n} x_{u,j}x_{v,j+1} \\
H_B = B\sum_{(uv)\in E} W_{uv}\sum_{j=1}^{n} x_{u,j}x_{v,j+1},
\end{gathered} \tag{4.4}$$

with $x_{i,j}$ being equal to 1 if the i-th node (city) is in the j-th position in the cycle (route), 0 otherwise, $x_{v,n+1} = x_{v,1}$, and A and B being positive constants ($A, B > 0$). In this context, H_A encodes the problem constraints, namely, each node must appear exactly once in the cycle (first term), there must be exactly one node in each cycle position (second term), and the order of nodes must be valid (third term). Conversely, H_B encodes the minimization of the total weight of the cycle (route length); specifically, $W_{u,v}$ represents the weight of the (u, v) edge. Eventually, to make the violation of the constraints energetically unfavourable, the relationship $0 < B(max(W_{uv})) < A$ must be satisfied. In the experiments presented here, B has been set to 1, while A was has been set to $n \times max(W_{uv})$.

Given the problem Hamiltonian H, a renumbering of the $x_{i,j}$ variables is necessary in order to construct the QUBO matrix Q. Specifically, the following renumbering scheme has been used here:

$$(x_{1,1},\ x_{1,2},\ \ldots\ x_{1,n},\ x_{2,1}\ \ldots\ x_{n,n}) \to (x_1,\ \ldots\ x_{n^2}). \tag{4.5}$$

Then, the Q_{ij} entry of Q, for $i, j \in \{1, \ldots n^2\}$, is given by the coefficient of $x_i x_j$ in H. The pseudocode for the computation of the QUBO matrix entries is provided in Algorithms 11 to 14.

```
Input: Distance matrix $D$
Result: Matrix $Q$ representing the QUBO formulation of the TSP problem defined by $D$
$n \leftarrow size(D, 0)$;                         // number of rows of D (square matrix)
$Q \leftarrow new\ int[n^2][n^2]$;
$all\_zeros(Q)$;
$A \leftarrow n \cdot max\_coeff(D)$;                          // penalty set according to [67]
$B \leftarrow 1$;                                        // multiplier set according to [67]
$Q \leftarrow add\_cost\_objective(Q, D, B)$;
$Q \leftarrow add\_time\_constraints(Q, A)$;
$Q \leftarrow add\_position\_constraints(Q, A)$;
return $Q$
```

Algorithm 11: Translation of TSP into QUBO [110] (taken from [12]).

```
Input: QUBO matrix $Q$, distance matrix $D$, multiplier $B$
Result: Matrix $Q$ including the cost objective
$n \leftarrow size(D, 0)$;                         // number of rows of D (square matrix)
for $t \leftarrow 0$ to $n - 1$ do
    for $i \leftarrow 0$ to $n - 1$ do
        for $j \leftarrow 0$ to $n - 1$ do
            $r \leftarrow t \cdot n + i$;
            $c \leftarrow (t + 1) \bmod n^2 + j$;
            $Q[r][c] \leftarrow B \cdot D[i][j]$;
        end
    end
end
return $Q$
```

Algorithm 12: *add_cost_objective* function (computation of H_B coefficients) (taken from [12]).

```
Input: QUBO matrix $Q$, constraint penalty $A$
Result: Matrix $Q$ including time constraints
$n \leftarrow size(D, 0)$;                         // number of rows of D (square matrix)
for $t \leftarrow 0$ to $n - 1$ do
    for $i \leftarrow 0$ to $n - 1$ do
        $r \leftarrow t \cdot n + i$;
        $Q[r][r] \leftarrow Q[r][r] - A$;
        for $j \leftarrow 0$ to $n - 1$ do
            if $i \neq j$ then
                $c \leftarrow t \cdot n + j$;
                $Q[r][c] \leftarrow 2 \cdot A$;
            end
        end
    end
end
return $Q$
```

Algorithm 13: *add_time_constraints* function (computation of part of the coefficients of H_A) (taken from [12]).

```
Input: QUBO Matrix $Q$, constraint penalty $A$
Result: Matrix $Q$ including position constraints
$n \leftarrow size(D, 0)$;                    // number of rows of D (square matrix)
for $i \leftarrow 0$ to $n - 1$ do
    for $t1 \leftarrow 0$ to $n - 1$ do
        $r \leftarrow t1 \cdot n + i$;
        $Q[r][r] \leftarrow Q[r][r] - A$;
        for $t2 \leftarrow 0$ to $n - 1$ do
            if $t1 \neq t2$ then
                $c \leftarrow t2 \cdot n + i$;
                $Q[r][c] \leftarrow 2 \cdot A$;
            end
        end
    end
end
return $Q$
```

Algorithm 14: *add_position_constraints* function (computation of part of the coefficients of H_A) (taken from [12]).

4.2.2.1 Solution Refinement Procedure

In the QUBO formulation of TSP, a solution is a binary vector of length n^2, where n denotes the number of nodes (cities). Alternatively, a solution can be seen as a list of n sub-sequences, each consisting of n locations. In particular, in a valid solution, each sub-sequence contains $n-1$ 0s and one 1; furthermore, the position of the value 1 within the i-th sub-sequence is unique with respect to all sub-sequences. For example, a valid solution for three cities is

$$x_1 = [0, 1, 0,\ 1, 0, 0,\ 0, 0, 1],$$

while an invalid solution is

$$x_2 = [1, 1, 0,\ 1, 0, 0,\ 0, 0, 0].$$

To convert a QUBO TSP solution into a classical TSP solution, each sub-sequence is replaced with the integer number representing the position of the value 1 inside that sub-sequence. For instance, the classical version of x_1 is

$$s_1 = [1, 0, 2].$$

Conversely, x_2 cannot be converted into a valid classical TSP solution unless it is refined, namely, modified a little bit.

To transform a non-valid QUBO TSP solution into a valid classical TSP solution while preserving the original solution as much as possible, the following procedure has been devised. Let x be the solution vector returned by the annealer, $f : \mathbb{B}^n \to \mathbb{N}^k$ be a function returning a vector of size k containing the position of all non-zero values in x_i (with x_i denoting the i-th sub-vector of length n in x), and A be the set $\{0, ..., n-1\}$. First of all, a solution vector s of length n is initialized with -1 values, namely, $s \in \{-1\}^n$. Let R be a vector of sets, with $R_i = \{f(x_i)\}$ being the set of possible solutions (nodes) for the i-th sub-sequence, based on the solution vector x. Therefore, $s_i = R_i[0] \iff sizeof(R_i) = 1$. Let K denote the set of unavailable nodes.

Definition 1 *An unavailable node is a node representing the solution for a specific position in the cycle (namely, no other 1 value exists in the corresponding sub-sequence) or a node designated as such after being randomly chosen from the available ones. In the subsequent example, the node 0 is unavailable.*

$$x_2 = [1, 1, 0,\ 1, 0, 0,\ 0, 0, 0]$$

In practical terms, $K = \{s_i \mid s_i \neq -1,\ i \in \{0, ..., n-1\}\}$. Then, let D_i be $R_i \setminus K$. The subsequent step involves computing D_i for the first i in $\{0, ..., n-1\}$ such that $sizeof(R_i) > 1$. If $sizeof(D_i) > 0$, a random value $d \in D_i$ is picked and the following actions are executed: $s_i = d$ and $K = \{K \cup d\}$, namely, the selected available node d is assigned to the i-th position of the solution vector, and the set of unavailable nodes is consequently updated. This process is then iterated for all subsequent i values that fulfill the condition on the size of R_i. After that, a vector idx_i is instantiated for each node i in $\{0, ..., n-1\}$. Specifically, $j \in idx_i \iff s_j = i$ (with $j \in \{0, ..., n-1\}$). Thus, idx_i comprises all the solution vector positions that contain the value i (duplicates may be present). If $sizeof(idx_i) > 1$ for a certain i, idx_i undergoes shuffling, and all s_j such that $j \in idx_i$ and $idx_i.indexof(j) > 0$ are set to -1. Essentially, for each idx_i, the s_j corresponding to the first index after the shuffling operation keeps the value i. In this way, duplicate nodes in the cycle are eliminated. At this point, all values in s are either unique or equal to -1. Then, let us define $L = A \setminus K$. In practice, L comprises all nodes that have not yet been assigned to a cycle position. Eventually, for each i in $\{0, ..., n-1\}$ such that $s_i = -1$, a random value $l \in L$ is selected, and the set of remaining nodes is updated consequently, namely, $s_i = l$ and $L = L \setminus \{l\}$. After this last step, all the positions have an assigned node, namely, $s_i \neq -1 \;\; \forall\, i \in \{0, ..., n-1\}$, and the solution is valid.

It is worth observing that this refinement procedure does not modify an already valid solution. In addition, in the case of non-valid solutions, it does not affect nodes whose position is unique within the cycle.

4.2.2.2 Experimental Setup and Results

The experiments have been executed sequentially on the Leap cloud, employing the QALS parameters reported in Table 4.3. In this case, the Python implementation has been used to obtain the results, whereas the C++ implementation has been employed to verify them. In addition, to ensure a fair comparison, an equivalent number of annealer measurements ($k = 5$) has been used for Embedding Composite. Concerning the annealing parameters, such as the annealing time and schedule, the default values have been utilized for both QALS and Embedding Composite (the system that has been used is Advantage 1.1) [48]. Instead, as mentioned earlier, Hybrid manages these properties in an automatic way. Eventually, it is worth mentioning that the brute force method whose results are reported here for comparison has been implemented in C++.

Table 4.3: Values used for QALS parameters in all TSP tests (taken from [12]).

p_δ	η	q	N	λ_0	k	N_{max}	d_{min}
0.1	0.2	0.2	5	1.5	5	100	70

In these experiments, QALS and Embedding Composite have never found solutions satisfying the H_A constraints. In particular, Embedding Composite does not ensure that either the optimal solution or a solution satisfying the constraints will be returned. Concerning QALS, given that only part of the entire problem (including the constraints) is mapped into the annealer topology at each iteration, the constraints are typically not entirely encoded inside the topology. Therefore, they are satisfied only if the optimal solution is found or in a limited number of other cases (these observations are based on the results obtained). In order to evaluate the performance of QALS and Embedding Composite, the refinement procedure presented in Section 4.2.2.1 has been applied to the solutions returned by them. The results reported here are those obtained after the execution of the refinement procedure.

The results are shown in Table 4.4. Specifically, μ denotes the average TSP cost across runs, σ represents the corresponding standard deviation, and the average time is expressed in seconds. Regarding the TSP cost, it corresponds to the route length (namely, the cycle cumulative weight) in the original problem space. It is also worth highlighting that, in these experiments, real numbers in the range $[0, 10]$ have been used as edge weights (city distances). In practice, in the smallest problem taken into account ($n = 10$), QALS has outperformed Embedding Composite but has been outperformed by both Hybrid, which has retrieved a solution close to the global minimum, and the brute force approach. In contrast, for $n = 12$, Embedding Composite has achieved better results than QALS. Concerning $n = 14$, the maximum dimension manageable by Embedding Composite, this method has found once again better solutions compared to QALS. For n values ranging from 32 to 72, only QALS and Hybrid could be applied, with Hybrid being faster and more effective than QALS. Eventually, only Hybrid has been able to address a problem of size $n = 74$. In particular, this should not be feasible, as the QUBO problem size exceeds the number of available qubits. Probably, in the case of incomplete graphs, Hybrid disregards non-connected edges, saving qubits. In addition, it is worth underscoring that the good results achieved by Hybrid are mainly related to the parallel execution of multiple solvers. In conclusion, the performance of both Embedding Composite and QALS could be probably improved by using a significantly higher number of annealer measurements ($k \gg 5$). As already said in Section 4.2.1.2, a larger output sample size would enhance the probability of getting a solution close to the optimum. However, the quantum annealing time at our disposal was not enough.

Table 4.4: Tests performed on the Travelling Salesman Problem. In particular, "E.C." stands for embedding composite, while "S.R." stands for solution refinement (taken from [12]).

TSP Size	QUBO Size	Approach	μ	σ	Avg. Time (s)	# Runs
10	100	Brute Force	35.20	0	48	10
		Hybrid	36.94	1.16	13.78	3
		E.C. with S.R.	55.13	3.22	94.66	10
		QALS with S.R.	49.95	3.02	64.83	10
12	144	Brute Force	26.21	0	56.8	3
		Hybrid	33.43	1.26	14.86	10
		E.C. with S.R.	52.91	6.5	132.96	10
		QALS with S.R.	54.77	4.53	265.3	10
14	196	Brute Force	33.94	0	10630	3
		Hybrid	49.48	1.99	15.7	3
		E.C. with S.R.	67.50	9.87	465	3
		QALS with S.R.	74.58	7.87	180	10
32	1024	Brute Force	-	-	-	-
		Hybrid	124.72	3.45	24.98	3
		E.C. with S.R.	-	-	-	-
		QALS with S.R.	157.99	10.23	588	10
64	4096	Brute Force	-	-	-	-
		Hybrid	288.87	5.82	23.25	3
		E.C. with S.R.	-	-	-	-
		QALS with S.R.	336.94	17.9	3570	10
72	5184	Brute Force	-	-	-	-
		Hybrid	331.24	16.39	37.36	3
		E.C. with S.R.	-	-	-	-
		QALS with S.R.	387.00	21.25	2528	3
74	5476	Brute Force	-	-	-	-
		Hybrid	344.19	2.8	38.65	3
		E.C. with S.R.	-	-	-	-
		QALS with S.R.	-	-	-	-

4.3 Discussion

In this chapter, two implementations of QALS, in C++ and in Python, and the empirical evaluation of the algorithm have been presented. The advantage of the C++ implementation lies in the higher efficiency of the classical part of the algorithm. Instead, the Python implementation is characterised by a smoother interaction with D-Wave's APIs, which are provided only in Python. In both cases, some changes with respect to the original pseudocode of the algorithm have been made, in order to improve the execution efficiency. Concerning the empirical evaluation, two optimization problems with complementary characteristics have been taken into account. In the case of NPP, which has a natural representation as QUBO, QALS has shown poor performance. Actually, the classical exact algorithm considered has outperformed both QALS and the hybrid solver provided by D-Wave, implying that the problem can be considered practically solved in most cases despite its NP-hard nature. Instead, in the case of TSP, which includes constraints that must be encoded as penalties, QALS has turned out to be able to address larger problems than Embedding Composite (the standard embedding procedure). Hence, the main objective of the algorithm has been empirically fulfilled. However, among the not-entirely classical methods tested, only D-Wave's hybrid solver has been able to provide solutions satisfying the problem constraints. In conclusion, QALS implementations could have practical potential on hard problems for which the QUBO representation cannot be directly embedded in the annealer topology. In addition, it is worth observing that D-Wave's hybrid solver represents an interesting baseline but cannot be considered as an effective competitor for drawing scientific conclusions, as it involves the parallel execution of multiple algorithms.

Concerning the code availability, the C++ implementation [71] and the Python implementation [11] are publicly available under the GPLv2 and MIT licences, respectively. Moreover, a refactored version of the Python implementation, supporting different tabu matrix types and including minor corrections, is available at `https://github.com/ZarHenry96/QALS-variants`.

5 Bayesian Networks Reconstruction

This chapter is a reworked version of the article “Reconstructing Bayesian networks on a quantum annealer” [130], which was motivated by the absence of an empirical evaluation of the quantum annealing algorithm for the reconstruction of Bayesian networks (see Section 2.4.1) in the work by O’Gorman et al. [76]. In practice, in this chapter, the implementation and the empirical evaluation of that algorithm are presented, with the purpose of assessing its applicability using the current quantum architectures. Given that the encoding of the problem and the consecutive embedding into the quantum architecture restrict the direct application of the algorithm to approximately 18 Bayesian variables (at the time of running the experiments), a divide et impera approach is also introduced. Both the original algorithm and this novel scheme have been evaluated on various problem instances characterized by an increasing number of variables, obtaining promising results.

5.1 O’Gorman’s Algorithm Implementation

O’Gorman’s algorithm has been implemented in Python, as D-Wave’s Ocean suite, which is essential for interacting with D-Wave’s quantum annealers, provides APIs only for this programming language. In this section, the implementation details, the complexity of the implementation, and a method to speed up the execution are presented.

5.1.1 QUBO Matrix Construction

The pseudocode for the implementation of O’Gorman’s algorithm is provided in Algorithm 15, which relies on Algorithms 16 to 18. Specifically, the input to the main algorithm consists of the number of Bayesian variables n, the number of states r_i for each variable, and the dataset of examples. The output corresponds to the QUBO matrix Q representing the given BNSL problem.

Before constructing the matrix, various intermediate values must be computed. First of all, for each Bayesian variable, all possible parent sets ($\mathit{\Pi}_i(B_s)$ in O’Gorman’s formulation) must be identified. The maximum number of parents m has been restricted to two for the reasons presented in Section 2.4.1. Therefore, the complexity of this step is $\mathcal{O}(n^3)$. It is also worth mentioning that the empty set is considered a valid parent set.

Then, the α_{ijk} hyperparameters of the Dirichlet priors are set to the uninformative value $1/(r_i \cdot q_i)$,

```
Input: number of Bayesian variables $n$, list $r = (r_i)_{i=1}^{n}$ with $r_i$ being the number of states of
       the $i^{th}$ variable, dataset $examples$
Result: QUBO matrix $Q$
/* calculation of the values needed to construct $Q$ */
$parentSets \leftarrow calcParentSets(n)$;
$\alpha \leftarrow calcAlpha(n, r, parentSets)$;
$s \leftarrow calcS(n, r, parentSets, \alpha, examples)$;                    // Algorithm 16
$w \leftarrow calcW(n, parentSets, s)$;                                       // Algorithm 17
$\Delta \leftarrow calcDelta(n, parentSets, w)$;                           // Eq. (2.23) and (2.24)
$\delta_{max} \leftarrow calcDeltaMax(n, \Delta)$;                                  // Eq. (2.20)
$\delta_{trans} \leftarrow calcDeltaTrans(n, \Delta)$;                              // Eq. (2.22)
$\delta_{consist} \leftarrow calcDeltaConsist(n, \delta_{trans})$;                  // Eq. (5.2)
/* construction of $Q$ */
$Q \leftarrow zeroMatrix()$;
$Q \leftarrow fillQ(Q, n, parentSets, w, \delta_{max}, \delta_{trans}, \delta_{consist})$;   // Algorithm 18
return $Q$;
```

Algorithm 15: *calcQUBOMatrix(n, r, examples)* (taken from [130]).

```
Input: number of Bayesian variables $n$, list of number of states $r$, list of parent sets
       $parentSets$, prior distributions hyperparameters $\alpha = (\alpha_{i\pi jk})$, dataset $examples$
Result: $s = (\{s_i(\pi)\ s.t.\ \pi \in parentSets[i]\})_{i=1}^{n}$ with $s_i(\pi)$ being the score for the Bayesian
        variable $i$ given the parent set $\pi$
function calcSi(i, π, r, α, examples):                                    // Eq. (5.1)
    $q_{i\pi} \leftarrow \prod_{p \in \pi} r_p$;                          // $q_{i\pi} = 1$ if $\pi = \emptyset$
    $sum \leftarrow 0$;
    for $j \leftarrow 1$ to $q_{i\pi}$ do
        $\alpha_{i\pi j} \leftarrow \sum_{k=1}^{r_i} \alpha_{i\pi jk}$;
        $N_{i\pi j} \leftarrow \sum_{k=1}^{r_i} calcNi\pi jk(examples, \pi, i, j, k, r)$;
        $sum \leftarrow sum + \ln\Gamma(\alpha_{i\pi j}) - \ln\Gamma(\alpha_{i\pi j} + N_{i\pi j})$;
        for $k \leftarrow 1$ to $r_i$ do
            $N_{i\pi jk} \leftarrow calcNi\pi jk(examples, \pi, i, j, k, r)$;
            $sum \leftarrow sum + \ln\Gamma(\alpha_{i\pi jk} + N_{i\pi jk}) - \ln\Gamma(\alpha_{i\pi jk})$;
        end
    end
    return -$sum$;
for $i \leftarrow 1$ to $n$ do
    for $\pi \in parentSets[i]$ do
        $s_i(\pi) \leftarrow calcSi(i, \pi, r, \alpha, examples))$;
    end
end
return $s$;
```

Algorithm 16: *calcS(n, r, parentSets, α, examples)* (taken from [130]).

where r_i denotes the number of states of the i-th variable, and q_i ($q_{i\pi}$ in the pseudocode) represents the number of states in the parent set taken into account. Essentially, all α_{ijk} associated with the same variable i and parent set π ($\alpha_{i\pi jk}$ in the pseudocode) share the same value. Additional insights into this choice are provided in Section 5.3.4. During this step, the α_{ijk} value must be determined for all "variable" - "parent set" - "parent set state" - "variable state" combinations, resulting in a complexity of $\mathcal{O}(n^3 r_{max}^3)$, with r_{max} denoting the maximum number of states of the Bayesian variables.

The subsequent step involves calculating the local scores s for all "Bayesian variable" - "parent set" combinations according to Equation (2.15). However, these computations are feasible only for very small datasets due to the factorial nature of the Γ function and the multiplications of Γ values involved. The solution consists in moving the logarithm inside through algebraic transformations until its argument reduces to the gamma function alone. In the implementation described here, the natural logarithm (ln) has been employed, resulting in the following formula (which is the one used in Algorithm 16):

$$s_i(\Pi_i(B_s)) = -\sum_{j=1}^{q_i}\Big[\ln(\Gamma(\alpha_{ij})) - \ln(\Gamma(N_{ij} + \alpha_{ij})) + \sum_{k=1}^{r_i}[\ln(\Gamma(N_{ijk} + \alpha_{ijk})) - \ln(\Gamma(\alpha_{ijk}))]\Big]. \tag{5.1}$$

This form allows leveraging the (natural) log-gamma function (denoted as $\ln\Gamma$ in the pseudocode), which exhibits a significantly slower growth compared to the gamma one. Moreover, in this way, the products in Equation (2.15) can be substituted with additions, further mitigating the risk of out-of-range values. Regarding the pseudocode, the *calcNiπjk* procedure calculates the number of times the variable i is in its k-th state while its parent set π is in its j-th state (for an empty parent set, only the state of the i-th variable is taken into account). The algorithm's complexity is $\mathcal{O}(n^3 N r_{max}^3)$.

After the calculation of the score values s, the parent set weights w can be computed according to Equation (2.14). The corresponding pseudocode is provided in Algorithm 17. In particular, since the maximum number of parents has been limited to two, the pseudocode does not consider scenarios with larger parent sets. The complexity of the algorithm that computes w is $\mathcal{O}(n^3)$.

```
Input: number of Bayesian variables $n$, list of parent sets $parentSets$, score values $s$
Result: $w = (\{w_i(\pi)\ s.t.\ \pi \in parentSets[i]\})_{i=1}^{n}$ with $w_i(\pi)$ being the weight calculated for
        the Bayesian variable $i$ given the parent set $\pi$
function calcWi(i, $\pi$, s):                                                // Eq. (2.14)
    if $\pi = \emptyset$ then
        return $s_i(\emptyset)$;
    else if $size(\pi) = 1$ then
        return $s_i(\pi) - s_i(\emptyset)$;
    else if $size(\pi) = 2$ then
        $p_1,\ p_2 \leftarrow \pi[1],\ \pi[2]$;
        return $s_i(\pi) - s_i(\{p_1\}) - s_i(\{p_2\}) + s_i(\emptyset)$;
    end
for $i \leftarrow 1$ to $n$ do
    for $\pi \in parentSets[i]$ do
        $w_i(\pi) \leftarrow calcWi(i, \pi, s)$;
    end
end
return $w$;
```

Algorithm 17: *calcW(n, parentSets, s)* (taken from [130]).

Ultimately, the penalty values must be computed. To this end, it is necessary to calculate the auxiliary quantities Δ_{ji} according to Equations (2.23) and (2.24). Although the complexity of this step would theoretically be $\mathcal{O}(n^3)$, the complexity in the implementation is $\mathcal{O}(n^4)$ due to the data structures utilized to store the parent sets. Once Δ has been calculated, all penalties can be computed. Specifically, $\delta_{max}^{(i)}$ is calculated for each Bayesian variable according to Equation (2.20), resulting in an overall complexity (for all $\delta_{max}^{(i)}$) of $\mathcal{O}(n^2)$. Conversely, the penalty bound associated with the consistency Hamiltonian (Equation 2.21) can be simplified since δ_{trans} is independent of its superscript indices. The resulting penalty bound is

$$\delta_{consist} > (n-2)\delta_{trans}. \tag{5.2}$$

In practical terms, δ_{trans} is calculated according to Equation (2.22) with a complexity of $\mathcal{O}(n^2)$ (note that δ_{trans} is a single value). After that, $\delta_{consist}$ is computed according to the simplified bound (Equation 5.2) with a complexity of $\mathcal{O}(1)$. To satisfy the lower bounds, the boundary values plus one have been used as penalty values.

At this point, the QUBO matrix Q can be filled as illustrated in Algorithm 18. This matrix, whose dimensions have been detailed in Section 2.4.1, contains all zeros at the beginning (look at line 9 of Algorithm 15, which has a complexity of $O(n^4)$). Specifically, the outermost loop, encompassing almost all the pseudocode, iterates on the Bayesian variables. The first part of the algorithm (lines 2-12) is associated with the score Hamiltonian, i.e., Equation (2.13). For each "Bayesian variable" - "parent set" combination, the parent set weight $w_i(\pi)$ ($w_i(J)$ in O'Gorman's formulation) is summed to the appropriate cell. Practically, the coefficients of the linear terms in Equation (2.13) (namely, the terms involving only one QUBO variable (d_{ji})) are added to cells of Q situated on the main diagonal. Conversely, the coefficients of the quadratic terms, involving two QUBO variables ($d_{xi}d_{yi}$), are added to cells outside the diagonal; here, the first variable defines the row, while the second one defines the column. The next segment of the algorithm corresponds to the max Hamiltonian (lines 13-25), namely, Equation (2.16). The strategy for inserting the coefficients is similar to that used for the score Hamiltonian but, in this case, there is a square in the formula. Therefore, first of all, the binary variables in Equation (2.16) and their coefficients inside the square are identified and stored in two lists (lines 14-15). After that, based on the square expansion, the linear and quadratic coefficients (including the multiplication by $\delta_{max}^{(i)}$) are added to the respective cells. The last part of the algorithm refers to the transitivity and consistency Hamiltonians (lines 26-44), namely, Equations (2.18) and (2.19). Since the number of terms in these formulas is small and there is no square, the procedure is quite

```
Input: zero matrix $Q$, number of Bayesian variables $n$, list of parent sets $parentSets$, parent set
       weights $w$, list of penalty values $\delta_{max}$, penalty value $\delta_{trans}$, penalty value $\delta_{consist}$
Result: QUBO matrix $Q$ filled according to $H_{score}$, $H_{max}$, $H_{trans}$, and $H_{consist}$
for $i \leftarrow 1$ to $n$ do
    /* $H_{score}$-related terms (Eq. (2.13)) */
    for $\pi \in parentSets[i]$ do
        if $size(\pi) = 1$ then                                  // diagonal elements
            $j \leftarrow \pi[1]$;
            $row \leftarrow col \leftarrow indexOf(d_{ji})$;
            $Q[row][col] \leftarrow Q[row][col] + w_i(\pi)$;
        else if $size(\pi) = 2$ then                             // out-of-diagonal elements
            $x,\ y \leftarrow \pi[1],\ \pi[2]$;
            $row,\ col \leftarrow indexOf(d_{xi}),\ indexOf(d_{yi})$;
            $Q[row][col] \leftarrow Q[row][col] + w_i(\pi)$;
        end
    end
    /* $H_{max}$-related terms (Eq. (2.16)) */
    $m \leftarrow 2$;                                            // max. num. of parents
    $sqBinVars \leftarrow binaryVarsInSquare()$;     // $d_i$ and $y_i$ in (2.16) are sums of binary vars
    $c \leftarrow binaryVarsCoefficientsInSquare()$;          // the coefficients are either -1 or -2
    for $j \leftarrow 1$ to $size(sqBinVars)$ do
        $row \leftarrow col \leftarrow indexOf(sqBinVars[j])$;                // diagonal elements indices
        $Q[row][col] \leftarrow Q[row][col] + \delta_{max}^{(i)} \cdot c[j]^2$;                    // squared term
        $Q[row][col] \leftarrow Q[row][col] + \delta_{max}^{(i)} \cdot (2 \cdot m \cdot c[j])$;        // double product with $m$
        for $k \leftarrow j + 1$ to $size(sqBinVars)$ do             // out-of-diagonal elements
            $row \leftarrow indexOf(sqBinVars[j])$;
            $col \leftarrow indexOf(sqBinVars[k])$;
            $Q[row][col] \leftarrow Q[row][col] + \delta_{max}^{(i)} \cdot (2 \cdot c[j] \cdot c[k])$;   // double product between vars
        end
    end
    /* $H_{cycle}$-related terms */
    for $j \leftarrow i + 1$ to $n$ do
        /* $H_{trans}$-related terms (Eq. (2.18)) */
        for $k \leftarrow j + 1$ to $n$ do
            $row \leftarrow col \leftarrow indexOf(r_{ik})$;
            $Q[row][col] \leftarrow Q[row][col] + \delta_{trans}$;          // $r_{ik}$ coefficient (diagonal element)
            $row,\ col \leftarrow indexOf(r_{ij}),\ indexOf(r_{jk})$;
            $Q[row][col] \leftarrow Q[row][col] + \delta_{trans}$;                      // $r_{ij} \cdot r_{jk}$ coefficient
            $row,\ col \leftarrow indexOf(r_{ij}),\ indexOf(r_{ik})$;
            $Q[row][col] \leftarrow Q[row][col] - \delta_{trans}$;                      // $r_{ij} \cdot r_{ik}$ coefficient
            $row,\ col \leftarrow indexOf(r_{ik}),\ indexOf(r_{jk})$;
            $Q[row][col] \leftarrow Q[row][col] - \delta_{trans}$;                      // $r_{ik} \cdot r_{jk}$ coefficient
        end
        /* $H_{consist}$-related terms (Eq. (2.19) */
        $row,\ col \leftarrow indexOf(d_{ji}),\ indexOf(r_{ij})$;
        $Q[row][col] \leftarrow Q[row][col] + \delta_{consist}$;                        // $d_{ji} \cdot r_{ij}$ coefficient
        $row \leftarrow col \leftarrow indexOf(d_{ij})$;
        $Q[row][col] \leftarrow Q[row][col] + \delta_{consist}$;            // $d_{ij}$ coefficient (diagonal element)
        $row,\ col \leftarrow indexOf(d_{ij}),\ indexOf(r_{ij})$;
        $Q[row][col] \leftarrow Q[row][col] - \delta_{consist}$;                        // $d_{ij} \cdot r_{ij}$ coefficient
    end
end
return $Q$;
```

Algorithm 18: *fillQ(Q, n, parentSets, w, δ_{max}, δ_{trans}, $\delta_{consist}$)* (taken from [130]).

simple. In particular, lines 28-35 add the coefficients from Equation (2.18) to the appropriate cells for any set of three Bayesian variables (H_{trans} penalties). Analogously, lines 37-42 sum the coefficients from Equation (2.19) to the right cells for each pair of Bayesian variables ($H_{consist}$ penalties). The overall complexity of the matrix filling procedure (Algorithm 18) is $\mathcal{O}(n^3)$.

Eventually, it is worth highlighting that the variables in the QUBO matrix Q are arranged as follows: first, the binary variables that encode the edges (d_{ij}); next, the binary slack variables (y_{il}) associated with the maximum parent constraint; and finally, the binary variables (r_{ij}) that encode the topological order. If the variables in the quadratic terms (defining the row and column indices) are appropriately sorted, the resulting matrix is upper triangular. Otherwise, the non-zero values below the main diagonal should be moved to the corresponding cells above the main diagonal by summing up the values involved.

5.1.2 Complexity Observations

The QUBO matrix construction (Algorithm 15) is characterised by an overall complexity of $\mathcal{O}(n^4 + n^3 N r_{max}^3)$, which is influenced by different factors: the number of Bayesian variables (n) in the BNSL problem, the dataset size (N), and the maximum number of states (r_{max}) of the problem variables. Specifically, if the dataset size is bigger than the number of Bayesian variables, the term dominating the algorithm complexity becomes $n^3 N r_{max}^3$. This situation is the most common one. In fact, the limitations in the number of qubits and connectivity in the current quantum annealers restrict n from being too large, while N needs to be sufficiently large in order to have an effective learning. As a consequence, the computation of the local score values s (Algorithm 16) is typically the most resource-expensive operation in the construction of the QUBO matrix. Otherwise, it would be the initialization of Q to zero, along with the computation of Δ (given how it has been implemented). Regarding the maximum number of states (r_{max}), its impact is particularly significant for Bayesian variables with continuous states. Indeed, for these variables, a discretization is necessary, and a higher representation accuracy corresponds to a higher execution time.

5.1.3 Execution Speedup

The QUBO matrix construction (including the computation of the intermediate values) is the most time-consuming step in the execution. A potential speedup could be achieved by employing a different programming language, such as C++. However, D-Wave provides only Python APIs to interact with quantum annealers, making it unfeasible. Conversely, a viable solution consists in using Numba [46], a just-in-time compiler for Python, to dynamically compile the code. Specifically, Numba requires the application of a decorator to the functions that need to be compiled. Then, the first time a decorated function is called, it is compiled into machine code, and the following calls will execute the machine code rather than the original Python code. It is worth noting that Numba performs better on code featuring loops, NumPy arrays and library functions. Additionally, the compilation is effective only if a function is called multiple times; a single call within a run may result in a slower execution.

In particular, Numba has been utilized solely in the experiments concerning the divide et impera approach (Section 5.3.5), as it has been introduced after concluding the experiments regarding O'Gorman's algorithm (Section 5.3.4). In addition, the dynamic compilation has been applied solely to the two most frequently invoked functions in the execution, i.e., *calcNiπjk* and an internal procedure.

5.2 Divide et Impera Approach

Because of the limited connectivity of the available quantum annealers, a high number of qubits is currently needed for embedding problems in the QPU topologies. In addition, the QUBO formulation introduced by O'Gorman et al. for the BNSL problem is characterised by a dense connectivity, making its usage infeasible even for instances with a moderate number of Bayesian variables. With these issues in mind, a divide et impera approach has been formulated and evaluated. The corresponding pseudocode is provided in Algorithm 19.

The first step consists in formulating the BNSL subproblems (lines 1-3). Let n denote the number of Bayesian variables of the original problem, r be an array specifying the number of states for each Bayesian variable, and *examples* denote an $N \times n$ matrix representing the dataset. The subproblems

```
Input: number of variables of the original BNSL problem $n$, number of variables for each
       subproblem $k$, list of number of states $r$, dataset $examples$
Output: adjacency matrix of the solution to the original problem $sol$
   /* Subproblems formulation */
 1 $subproblems \leftarrow combinations(n, k)$;
 2 $subproblemsR \leftarrow filter(r, subproblems)$;
 3 $subproblemsEx \leftarrow filter(examples, subproblems)$;
   /* Subproblems solution */
 4 $S \leftarrow Set()$;
 5 for $i \leftarrow 0$ to $size(subproblems)$ do
 6     $subprob$, $subprobR$, $subprobEx \leftarrow subproblems[i]$, $subproblemsR[i]$, $subproblemsEx[i]$;
 7     $subprobQ \leftarrow calcQUBOMatrix(size(subprob), subprobR, subprobEx)$;  // Algorithm 15
 8     $subprobAdjMatrix \leftarrow solveQUBO(subprobQ)$;
 9     $S.add(subprob, subprobAdjMatrix)$;
10 end
   /* Original solution reconstruction */
11 $C$, $P \leftarrow countEdgesAndPenalties(S, k)$;
12 $sol \leftarrow zeroMatrix()$;
13 for $i \leftarrow 0$ to $n$ do
14     for $j \leftarrow 0$ to $n$ do
15         if $i \neq j$ then
16             if $(C_{ij} - P_{ij}) > 0$ and $C_{ij} > C_{ji}$ then
17                 $sol[i][j] = 1$;
18             end
19         end
20     end
21 end
22 return $sol$;
```

Algorithm 19: *divideEtImpera(n, k, r, examples)* (taken from [130]).

are constructed as combinations of n variables taken k at a time, with k being the desired number of variables per subproblem. Specifically, in this algorithm, all combinations of Bayesian variables are taken into account, and the subproblems are identified by the indices of the variables involved. The computational complexity of this operation is $O(c\binom{n}{k})$, with c being the (constant) cost of generating a list of indices. In particular, given that the minimum reasonable number of variables for the QUBO encoding is 3 (H_{trans} assumes $n \geq 3$), k should be set to a value equal to or greater than 3. Subsequently, for each combination of k variables, the r vector and the *examples* matrix must be filtered in order to obtain a vector of k elements and an $N \times k$ matrix. The complexity of this last operation is $O(\binom{n}{k}(k + Nk))$.

After constructing the subproblems, O'Gorman's algorithm is applied to each of them (line 7), yielding the corresponding QUBO matrices, which can be provided to the annealer or solved using other methods (line 8). In this way, an adjacency matrix is obtained for each subproblem. In terms of computational complexity, this step is linear in the number of subproblems ($\binom{n}{k}$).

Ultimately, the solution to the original BNSL problem is reconstructed using the subproblems solutions (lines 11-21). Let S denote the set of subproblems solutions, with each solution including the list of variable indices and the adjacency matrix for the corresponding graph. Here, the significant information is the presence of edges. Hence, first of all, the occurrences of each edge in the subproblems solutions are counted (each pair of variables appears in multiple subproblems). The output of this counting operation, whose complexity is $O(\binom{n}{k}k^2)$, is represented by the set of counts C (with C_{ij} being the number of appearances of the directed edge (i, j)). Then, the reconstruction process begins. In particular, two strategies have been devised for this purpose. The simplest strategy consists in adding to the final solution every edge (i, j) that occurs at least once in a subproblem (namely, for which $C_{ij} > 0$). If both C_{ij} and C_{ji} are greater than 0, the edge with the highest number of occurrences is taken, in order to avoid creating cycles of two nodes (the graph must be a DAG). In contrast, the second strategy (detailed in the pseudocode) necessitates additional information for the reconstruction, i.e., the penalty values P_{ij}. Essentially, P_{ij} corresponds to the number of subproblems, including i and j, in which the edge (i, j) is absent. Thus, the complexity of the computation is the same as for C. In practical terms, an edge (i, j) is added to the final solution if (i) the difference between C_{ij} and P_{ij} is greater than 0 and (ii) C_{ij} is greater than C_{ji}. Otherwise, it is discarded. It is worth noticing that, if these conditions are met, so are those of the first method ($C_{ij} > 0$ and $C_{ij} > C_{ji}$). In the experiments, the second strategy has been employed, as some preliminary experiments have indicated its superiority. Regarding the overall complexity of the reconstruction step, it is $O(\binom{n}{k}k^2 + n^2)$.

5.3 Empirical Evaluation

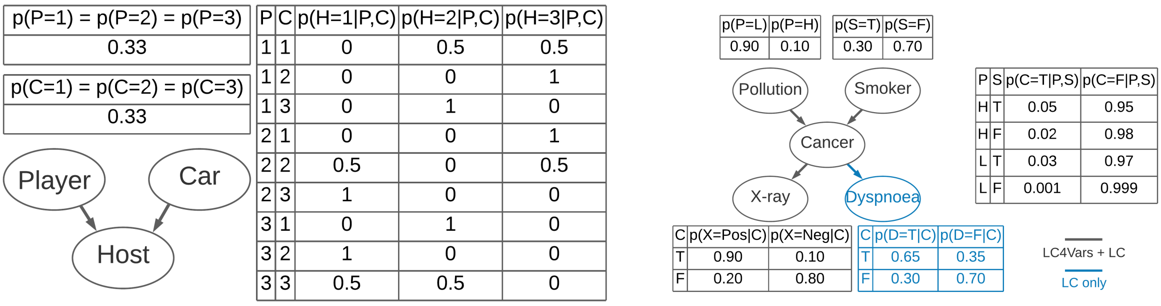
This section covers the Bayesian problems chosen, the methodology used for generating datasets, the methods evaluated, the experimental setup, and the results obtained for both O'Gorman's algorithm and the divide et impera approach.

5.3.1 Bayesian Problems

Three of the four Bayesian problems considered have been taken from the Bayes Server site [106], while the fourth one (Lung Cancer) has been picked from a different source [20]. Specifically, the implementation of O'Gorman's algorithm has been evaluated on the Monty Hall, Lung Cancer, and Waste problems. Conversely, the divide et impera approach has been evaluated on the Lung Cancer, Waste, and Alarm problems.

First of all, the Monty Hall Problem (MHP) has been selected due to its simplicity. Indeed, the Bayesian network for this problem (see Figure 5.1, on the left) comprises three variables ($n = 3$), with each variable having three possible states ($\{1, 2, 3\}$). In particular, three is also the minimum reasonable number of variables for the QUBO formulation proposed by O'Gorman et al., as the transitivity Hamiltonian requires the presence of at least three Bayesian variables.

Concerning the Lung Cancer problem, it has been chosen for its moderately higher number of variables compared to the MHP. In detail, the original network (LC) comprises $n = 5$ variables, with each Bayesian variable having two possible states. In addition, for a more detailed examination of the scaling behaviour of the algorithm with respect to the problem size, a variant of LC with $n = 4$ variables (LC4Vars), obtained by eliminating the "Dyspnoea" node, has been taken into account. Both networks are illustrated in Figure 5.1 (right).



p(P=1) = p(P=2) = p(P=3)
0.33

p(C=1) = p(C=2) = p(C=3)
0.33

P	C	p(H=1|P,C)	p(H=2|P,C)	p(H=3|P,C)
1	1	0	0.5	0.5
1	2	0	0	1
1	3	0	1	0
2	1	0	0	1
2	2	0.5	0	0.5
2	3	1	0	0
3	1	0	1	0
3	2	1	0	0
3	3	0.5	0.5	0

p(P=L)	p(P=H)
0.90	0.10

p(S=T)	p(S=F)
0.30	0.70

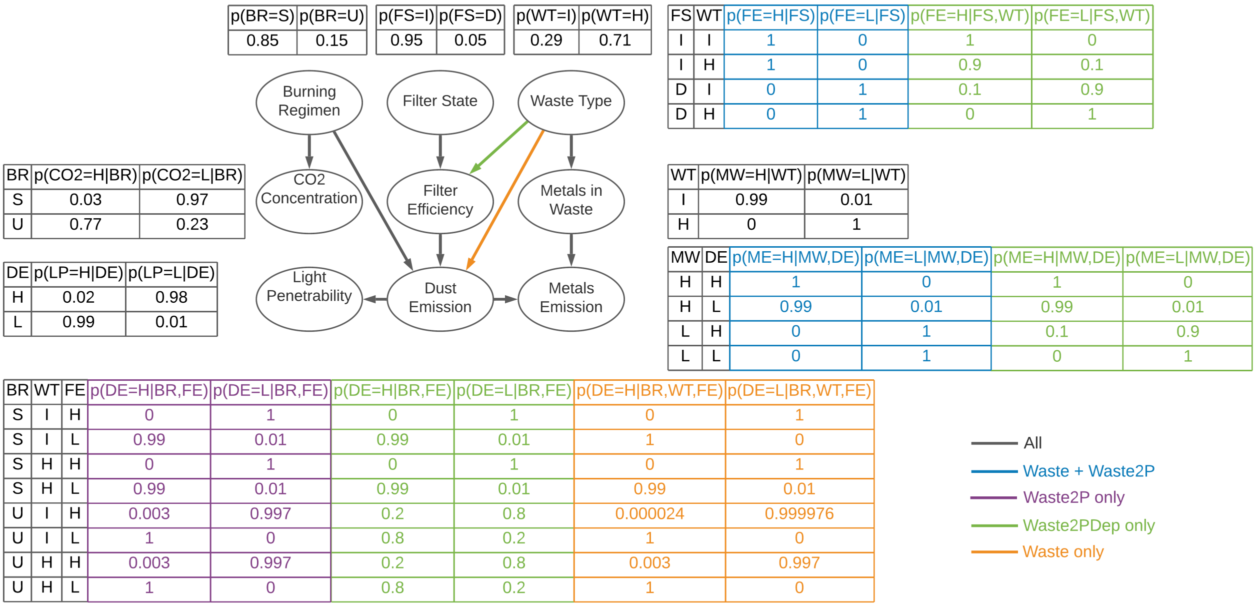
P	S	p(C=T|P,S)	p(C=F|P,S)
H	T	0.05	0.95
H	F	0.02	0.98
L	T	0.03	0.97
L	F	0.001	0.999

C	p(X=Pos|C)	p(X=Neg|C)
T	0.90	0.10
F	0.20	0.80

C	p(D=T|C)	p(D=F|C)
T	0.65	0.35
F	0.30	0.70

Figure 5.1: Monty Hall Problem (left) and Lung Cancer (right) (taken from [130]).

Instead, the Waste problem has been selected for several reasons: it has a significantly larger size with respect to the previous problems, it features continuous Bayesian variables, and it includes a variable with more than two parents. Specifically, the original Bayesian network for this problem consists of nine variables ($n = 9$), three of which are discrete with two states and six are continuous. Since O'Gorman's QUBO encoding supports only discrete variables, a discretization has been necessary for the continuous ones. Nevertheless, due to the different mean values of these continuous variables,



p(BR=S)	p(BR=U)
0.85	0.15

p(FS=I)	p(FS=D)
0.95	0.05

p(WT=I)	p(WT=H)
0.29	0.71

FS	WT	p(FE=H|FS)	p(FE=L|FS)	p(FE=H|FS,WT)	p(FE=L|FS,WT)
I	I	1	0	1	0
I	H	1	0	0.9	0.1
D	I	0	1	0.1	0.9
D	H	0	1	0	1

BR	p(CO2=H|BR)	p(CO2=L|BR)
S	0.03	0.97
U	0.77	0.23

WT	p(MW=H|WT)	p(MW=L|WT)
I	0.99	0.01
H	0	1

DE	p(LP=H|DE)	p(LP=L|DE)
H	0.02	0.98
L	0.99	0.01

MW	DE	p(ME=H|MW,DE)	p(ME=L|MW,DE)	p(ME=H|MW,DE)	p(ME=L|MW,DE)
H	H	1	0	1	0
H	L	0.99	0.01	0.99	0.01
L	H	0	1	0.1	0.9
L	L	0	1	0	1

BR	WT	FE	p(DE=H|BR,FE)	p(DE=L|BR,FE)	p(DE=H|BR,FE)	p(DE=L|BR,FE)	p(DE=H|BR,WT,FE)	p(DE=L|BR,WT,FE)
S	I	H	0	1	0	1	0	1
S	I	L	0.99	0.01	0.99	0.01	1	0
S	H	H	0	1	0	1	0	1
S	H	L	0.99	0.01	0.99	0.01	0.99	0.01
U	I	H	0.003	0.997	0.2	0.8	0.000024	0.999976
U	I	L	1	0	0.8	0.2	1	0
U	H	H	0.003	0.997	0.2	0.8	0.003	0.997
U	H	L	1	0	0.8	0.2	1	0

Figure 5.2: Waste (taken from [130]).

a single discretization threshold could not be used. In practice, each continuous variable has been converted into a discrete one with two states using the following procedure: first, the lowest and highest values have been found by evaluating all settings of the parent variables, and the average of these values has been taken as the discretization threshold; then, for each combination of the parent states, the mean and the variance of the continuous variable have been determined, and the probability of the high state H (as opposed to the low state L) has been set to the probability of a Gaussian with these parameters having a value higher than the threshold. In particular, to identify the lowest (highest) value, the corresponding variance has been subtracted from (summed to) the minimum (maximum) mean value observed. In the case of a continuous variable with a continuous parent, the parent has been discretized first, and the lowest and highest values have been employed as evidence for the parent states L and H. The Bayesian problem resulting from this process is denoted as Waste.

Actually, different variants of the Waste problem, summarized in Figure 5.2, have been considered. Specifically, the difference between Waste and the original problem consists in the absence of the edge between *Waste Type* and *Filter Efficiency*, which has been cancelled by the discretization procedure. Conversely, the edge between *Waste Type* and *Dust Emission* has been deleted in Waste2P (variant of Waste) in order to have a maximum number of parents equal to two. In practical terms, the most balanced probability values have been retained for the *Dust Emission* node. Eventually, Waste2PDep is a modified version of Waste2P characterised by the reintroduction of the edge between *Waste Type* and *Filter Efficiency*. To achieve this, some probability values have been manually altered in the *Filter Efficiency* probability table. Additionally, in Waste2PDep, some other slight adjustments have been made to the probability values in the *Dust Emission* and *Metals Emission* tables in order to obtain more balanced probability distributions.

Eventually, Alarm has been chosen due to its size and the presence of a variable with four parents, which allows evaluating the divide et impera approach's capability to reconstruct Bayesian networks with more than two parents. Specifically, the original problem comprises 38 Bayesian variables but the version employed here includes only 15 of them (with their relationships preserved). Indeed, the maximum number of Bayesian variables that can be embedded in the Pegasus topology using O'Gorman's formulation is approximately 18 (at the time of running the experiments). The Bayesian network used in the experiments is illustrated in Figure 5.3.

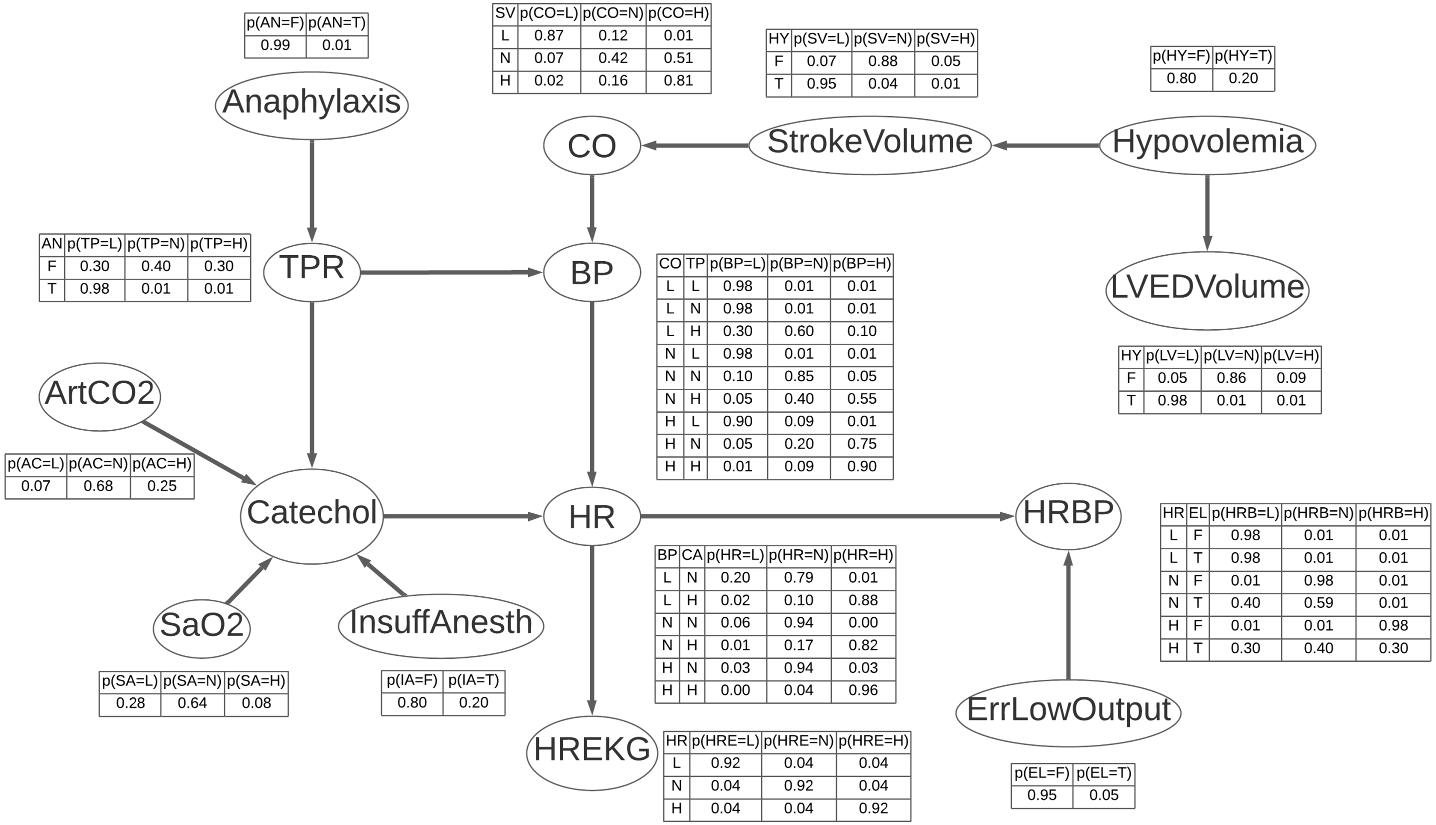
5.3.2 Datasets Generation

For each problem, datasets with different sizes have been generated using two distinct creation methods. In detail, three different dataset sizes (N) have been utilized: 10^4 (10K), 10^5 (100K), and 10^6

Catechol

IA	TP	AC	SA	p(CA=N)	p(CA=H)
F	L	L	L	0.05	0.95
F	L	L	N	0.10	0.90
F	L	L	H	0.95	0.05
F	L	N	L	0.05	0.95
F	L	N	N	0.10	0.90
F	L	N	H	0.95	0.05
F	L	H	L	0.01	0.99
F	L	H	N	0.10	0.90
F	L	H	H	0.30	0.70
F	N	L	L	0.05	0.95
F	N	L	N	0.95	0.05
F	N	L	H	0.99	0.01
F	N	N	L	0.05	0.95
F	N	N	N	0.95	0.05
F	N	N	H	0.99	0.01
F	N	H	L	0.01	0.99
F	N	H	N	0.30	0.70
F	N	H	H	0.99	0.01
F	H	L	L	0.05	0.95
F	H	L	N	0.95	0.05
F	H	L	H	0.95	0.05
F	H	N	L	0.05	0.95
F	H	N	N	0.95	0.05
F	H	N	H	0.99	0.01
F	H	H	L	0.01	0.99
F	H	H	N	0.30	0.70
F	H	H	H	0.30	0.70

IA	TP	AC	SA	p(CA=N)	p(CA=H)
T	L	L	L	0.01	0.99
T	L	L	N	0.01	0.99
T	L	L	H	0.01	0.99
T	L	N	L	0.01	0.99
T	L	N	N	0.01	0.99
T	L	N	H	0.01	0.99
T	L	H	L	0.01	0.99
T	L	H	N	0.01	0.99
T	L	H	H	0.01	0.99
T	N	L	L	0.01	0.99
T	N	L	N	0.01	0.99
T	N	L	H	0.05	0.95
T	N	N	L	0.01	0.99
T	N	N	N	0.01	0.99
T	N	N	H	0.05	0.95
T	N	H	L	0.01	0.99
T	N	H	N	0.01	0.99
T	N	H	H	0.01	0.99
T	H	L	L	0.01	0.99
T	H	L	N	0.05	0.95
T	H	L	H	0.05	0.95
T	H	N	L	0.01	0.99
T	H	N	N	0.05	0.95
T	H	N	H	0.05	0.95
T	H	H	L	0.01	0.99
T	H	H	N	0.01	0.99
T	H	H	H	0.01	0.99



p(AN=F)	p(AN=T)
0.99	0.01

AN	p(TP=L)	p(TP=N)	p(TP=H)
F	0.30	0.40	0.30
T	0.98	0.01	0.01

p(AC=L)	p(AC=N)	p(AC=H)
0.07	0.68	0.25

p(SA=L)	p(SA=N)	p(SA=H)
0.28	0.64	0.08

p(IA=F)	p(IA=T)
0.80	0.20

SV	p(CO=L)	p(CO=N)	p(CO=H)
L	0.87	0.12	0.01
N	0.07	0.42	0.51
H	0.02	0.16	0.81

HY	p(SV=L)	p(SV=N)	p(SV=H)
F	0.07	0.88	0.05
T	0.95	0.04	0.01

p(HY=F)	p(HY=T)
0.80	0.20

CO	TP	p(BP=L)	p(BP=N)	p(BP=H)
L	L	0.98	0.01	0.01
L	N	0.98	0.01	0.01
L	H	0.30	0.60	0.10
N	L	0.98	0.01	0.01
N	N	0.10	0.85	0.05
N	H	0.05	0.40	0.55
H	L	0.90	0.09	0.01
H	N	0.05	0.20	0.75
H	H	0.01	0.09	0.90

HY	p(LV=L)	p(LV=N)	p(LV=H)
F	0.05	0.86	0.09
T	0.98	0.01	0.01

BP	CA	p(HR=L)	p(HR=N)	p(HR=H)
L	N	0.20	0.79	0.01
L	H	0.02	0.10	0.88
N	N	0.06	0.94	0.00
N	H	0.01	0.17	0.82
H	N	0.03	0.94	0.03
H	H	0.00	0.04	0.96

HR	EL	p(HRB=L)	p(HRB=N)	p(HRB=H)
L	F	0.98	0.01	0.01
L	T	0.98	0.01	0.01
N	F	0.01	0.98	0.01
N	T	0.40	0.59	0.01
H	F	0.01	0.01	0.98
H	T	0.30	0.40	0.30

HR	p(HRE=L)	p(HRE=N)	p(HRE=H)
L	0.92	0.04	0.04
N	0.04	0.92	0.04
H	0.04	0.04	0.92

p(EL=F)	p(EL=T)
0.95	0.05

Figure 5.3: Alarm (taken from [130]).

(1M). Concerning the generation methods, the first one consists in sampling N examples from the probability distribution of the considered network. Conversely, the goal of the second one is to produce datasets with zero variance, namely, with state combinations appearing the expected number of times. To achieve this, the probability p of every combination of states is computed, and, for each combination, $\lfloor N * p \rfloor$ samples are included in the dataset. However, for certain probability values, it is not possible to generate an integer number of samples, and, therefore, the dataset variance is not precisely zero. Additionally, the number of samples of the resulting dataset might be lower than the desired one. For example, in the case of the Alarm problem, the dataset of size $N = 10^4$ generated using the second approach contains approximately 9000 samples, since some state combinations have too low probabilities to be represented (the other datasets are not substantially affected by this issue). The datasets generated with the second method are denoted here as *Exp*.

5.3.3 Methods and Experimental Setup

O'Gorman's algorithm provides as output a QUBO matrix encoding the BNSL problem instance given as input. To solve the QUBO formulation of the problem, three different methods have been considered in the experiments: quantum annealing, simulated annealing, and exhaustive search (ES). Quantum annealing has been presented in detail in Section 2.1.1. Instead, simulated annealing is a classical metaheuristic technique for solving optimization problems [62], and additional information can be found here[1]. Lastly, the exhaustive search corresponds to an optimised brute force approach.

In particular, applying a straightforward brute force approach on the QUBO formulation would be impractical even for moderately sized BNSL instances due to the excessive number of binary variables involved. Therefore, ES restricts the brute force to the edge binary variables d_{ij}, evaluating only the best configuration of y_{il} and r_{ij}. Specifically, for each Bayesian variable i, the y_{il} binary variables are set such that y_i (in Equation 2.16) equals the difference between m and d_i. In this way, penalties from the max Hamiltonian are avoided. Clearly, this is feasible only for nodes with no more than m parents. If the maximum number of parents constraint is violated, the minimum penalty corresponds to $y_i = 0$. Conversely, finding the best configuration for the r_{ij} binary variables is more difficult. In fact, if two Bayesian variables i and j are not connected (namely, there is no path from one variable to the other), a trivial setup for r_{ij} does not exist. For instance, setting all uncertain binary variables to the same value might lead to the creation of a cycle. The solution that has been adopted consists in adding one edge at a time to the graph encoded in d_{ij}, while checking that no cycle has been produced, in order to complete the graph and be able to compute a topological order. This process enables the proper configuration of all r_{ij} binary variables, avoiding any penalty resulting from the cycle Hamiltonian. Clearly, the penalty cannot be avoided if the graph encoded in d_{ij} includes a cycle. Setting y_{il} and r_{ij} has an overall complexity of $\mathcal{O}(n^3)$ for sparse graphs and $\mathcal{O}(n^4)$ for dense graphs. In conclusion, ES has still an exponential complexity with respect to the number of Bayesian variables n, but the number of operations required is significantly reduced compared to a simple brute force. Another enhancement introduced in ES is the parallel evaluation of the solutions.

In conducting experiments on the implementation of O'Gorman's algorithm, different combinations of annealing parameters (number of reads and annealing time) [48] have been evaluated for QA, with detailed values provided in the respective results sections. Additionally, the default annealing schedule (for the Advantage 1.1 system) has been used, and the embedding into the QPU topology has been delegated to the *EmbeddingComposite* class (see Section 2.2.1). Concerning SA, the implementation supplied by D-Wave [47] has been employed. With the exception of the number of reads, for which different values have been used, the default configuration has been maintained, and the execution has been performed on a local machine. Conversely, ES, which has also been run locally, does not necessitate to configure parameters. Specifically, a machine equipped with a quad-core CPU (Intel i5-6400) and 16 GB of RAM has been employed for the generation of the datasets, the construction of the QUBO matrices, and the execution of the classical methods.

Regarding the divide et impera approach, a single configuration of annealing parameters has been assessed for QA, with the rationale and values specified in the corresponding results section. In this case, the default annealing schedule for the Advantage 4.1 system has been used, as the ear-

[1] `https://en.wikipedia.org/wiki/Simulated_annealing`

lier model had already been retired. Conversely, the QPU embedding method and SA implementation/configuration have not been changed with respect to the previous experiments, except for the number of reads for SA. Eventually, ES has not been considered for these experiments due to its extensive time requirements. All classical operations for the divide et impera approach have been performed on a machine equipped with a quad-core CPU (Intel i7-7700HQ) and 16 GB of RAM.

Lastly, it is worth observing that the performance-related analyses conducted for both the implementation of O'Gorman's algorithm and the divide et impera approach exclusively focus on the *Exp* datasets, since some preliminary tests have established that the variance in performance between *Exp* and non-*Exp* datasets is negligible.

5.3.4 O'Gorman's Algorithm Results

Various experiments have been conducted on the implementation of O'Gorman's algorithm, and the following paragraphs present the results obtained.

5.3.4.1 QUBO Formulation Correctness and α_{ijk} Hyperparameters

The α_{ijk} hyperparameters must be properly configured to ensure that the image ($\mathbf{x}^T Q \mathbf{x}$) of the expected solution is the global minimum. To verify this, the global minimum solution has been found using ES and compared with the expected one. Specifically, the QUBO form of the expected solution ($\mathbf{x}$) has been obtained by setting the d_{ij} binary variables according to the Bayesian network used for the dataset generation. Instead, the optimal configuration of y_{il} and r_{ij} has been determined using the approach exploited by ES (see Section 5.3.3).

Given that the goal is to learn the structure of a Bayesian network from a dataset, the α_{ijk} values need to be uninformative, namely, they should not include information about the target Bayesian network. Hence, the first α_{ijk} values that have been evaluated are $N/(r_i \cdot q_i)$ and 1, following the proposal of Heckerman et al. [44]. The results are provided in Table 5.1. Specifically, the first alternative ($N/(r_i \cdot q_i)$) has shown decent performance on the smallest problem (MHP) and has performed poorly on all the others (with more Bayesian variables). In practical terms, this approach adds, in Equation (2.15), N prior counts uniformly distributed among all "variable" - "parent set" state combinations. On the other hand, the second alternative (namely, 1) has not worked at all. Consequently, other values have been taken into account, i.e., $1/(r_i \cdot q_i)$ and $1/r_i$, which have been chosen with the intent of minimizing their impact on the counts. As indicated in the same table, both alternatives have exhibited the desired behaviour, and the former has been selected as the default one.

The ratios shown in Table 5.1 have been obtained employing 8 datasets for each combination of problem and α_{ijk} value. These datasets have been generated with various sizes and using both generation methods outlined in Section 5.3.2. In particular, four datasets with $N = 10^4$ (including one *Exp*), two datasets with $N = 10^5$ (including one *Exp*), and two datasets with $N = 10^6$ (including one *Exp*) have been employed.

Table 5.1: Ratio of the number of times in which the best and the expected solutions coincide to the number of tests (8 for each cell), for different problems and α_{ijk} values (taken from [130]).

Problem	$\alpha_{ijk} = N/(r_i \cdot q_i)$	$\alpha_{ijk} = 1$	$\alpha_{ijk} = 1/(r_i \cdot q_i)$	$\alpha_{ijk} = 1/r_i$
MHP	0.75	0.00	**1.00**	1.00
LC4Vars	0.00	0.00	**1.00**	1.00
LC	0.00	0.00	**1.00**	1.00

5.3.4.2 Dataset Size and QUBO Matrix Construction Time

After determining the appropriate α_{ijk} values, an examination of the dataset size's impact on the QUBO matrix construction time has been conducted. As reported in Table 5.2, the required time exhibits a linear increase with the dataset size, aligning with the complexity $\mathcal{O}(n^4+n^3Nr_{max}^3)$ discussed in Section 5.1.2. Essentially, constructing the QUBO matrix with large datasets is unfeasible, especially when dealing with problems with a high number of variables. Since a dataset size beyond $N = 10^4$ yields no performance improvement (as shown Table 5.3), it is better to use a limited dataset size.

Concerning the data presented in the tables, the time values in Table 5.2 have been acquired

Table 5.2: Average QUBO matrix (Q) construction time in seconds, for different problems and dataset sizes. For each entry, 5 different datasets (of which one *Exp*) have been used (taken from [130]).

Problem	$N = 10^4$	$N = 10^5$	$N = 10^6$
MHP	0.55	4.03	40.03
LC4Vars	0.64	5.86	58.78
LC	1.40	13.55	136.21
Waste	9.79	98.85	1010.30
Waste2P	9.74	98.78	1001.96
Waste2PDep	9.77	100.26	1007.61

Table 5.3: Average solution value found by QA in Waste2PDep on *Exp* datasets of different sizes. 10^4 reads, 20 μs of annealing time, and 10 runs (for each dataset size) have been used (taken from [130]).

Problem	$N = 10^4$	$N = 10^5$	$N = 10^6$
Waste2PDep	0.963	0.960	0.965

employing one *Exp* and four non-*Exp* datasets for each combination of problem and dataset size, resulting in five runs for each entry. On the other hand, the values reported in Table 5.3 have been obtained using QA, with *Exp* datasets only, 10^4 reads, 20 μs of annealing time, and 10 runs for each dataset size. The metric shown in the second table is described in detail in Section 5.3.4.4. For now, it suffices to understand that larger values in the table correspond to better performance.

5.3.4.3 Number of Reads and Annealing Time (QA)

The number of reads, namely, measurements, and the annealing time per read are two crucial parameters influencing QA performance. Therefore, an extensive empirical assessment has been conducted, with 10 runs for each configuration, and the results obtained are detailed in Table 5.4. Specifically, the maximum permitted number of reads on D-Wave systems is 10^4, while the maximum annealing time per read is 2000 μs. Nonetheless, internal constraints limit the annealing time to no more than 999 μs with 10^3 reads and 99 μs with 10^4 reads. Upon examining the results, it becomes evident that a higher number of reads improves the performance, and the same is true for the annealing time. However, the impact of the number of reads is more marked. In fact, the results obtained with the maximum permitted number of reads and an annealing time well below the joint limit are way better than those obtained with a maximized annealing time. Therefore, the optimal configuration is represented by the maximum allowed number of reads (10^4) and the joint limit annealing time (99 μs).

Table 5.4: Ratio of the number of times the global minimum is found by QA to the number of experiments (10 for each configuration), for different numbers of reads and annealing times on the LC *Exp* dataset with size $N = 10^4$ (taken from [130]).

	Annealing time (μs)					
reads #	1	10	20	50	99	999
10^3	—	0.00	—	—	—	0.10
10^4	0.00	0.20	0.20	0.40	0.50	—

5.3.4.4 Performance

Eventually, all methods outlined in Section 5.3.3 have been applied to the problems detailed in Section 5.3.1 to compare their performance. The results obtained are shown in Table 5.5. Specifically, in these experiments, *Exp* datasets with a size of $N = 10^4$ have been used. Furthermore, 10^4 reads have been employed for SA, while, for QA, the optimal configuration has been used (namely, 10^4 reads and an annealing time per read of 99 μs). Ten runs have been executed for both SA and QA, and the success rate is defined as the ratio of the number of runs where the expected solution has been found to the total number of runs. Concerning the result value, it corresponds to the ratio of the QUBO image (x^TQx) of the solution found to the QUBO image of the expected solution, averaged across runs

Table 5.5: Comparison of ES, SA, and QA performances on different problems, using *Exp* datasets of size $N = 10^4$, 10^4 reads for SA, and the best setup for QA (10^4 reads, 99 μs annealing time). The number of runs, for both SA and QA, is 10 (taken from [130]).

		Exhaustive search		Simulated annealing			Quantum annealing			
n	Problem	Success rate	Avg. resol. time (s)	Success rate	Average result	Avg. sol. time (s)	Success rate	Average result	Avg. resol. time (s)	Average # exp. sol.
3	MHP	1.00	0.04	1.00	1.0000	3.40	1.00	1.0000	1.76	215.90
4	LC4Vars	1.00	0.42	1.00	1.0000	5.52	1.00	1.0000	1.77	11.40
5	LC	1.00	137.53	1.00	1.0000	8.93	0.50	0.9987	1.80	0.60
9	Waste	—	—	0.00	1.0145	12.85	0.00	0.9898	2.09	0.00
9	Waste2P	—	—	0.00	0.9999	13.15	0.00	0.9754	2.17	0.00
9	Waste2PDep	—	—	0.00	0.9998	12.39	0.00	0.9780	2.10	0.00

(greater values indicate better performance, as the image value of the expected solution is consistently negative). Instead, the time values comprise only the QUBO matrix resolution. In particular, for QA, these values represent the QPU access time (refer to [54] for additional information). Ultimately, the last column (*Average # exp. sol.*) corresponds to the average number of times that the expected solution has been retrieved in a single QA run.

Essentially, ES has demonstrated superior performance compared to SA and QA in the case of the smallest problems, namely, MHP and LC4Vars, consistently finding the global minimum in a shorter time. Nevertheless, because of its exponential complexity, ES has been outperformed on the LC problem ($n = 5$) and has proven to be excessively time-consuming for larger ones.

Regarding the other methods, QA has consistently identified the optimum solution for problems with three and four Bayesian variables, also requiring less time than SA. Additionally, it has discovered the global minimum multiple times in each run (the number of measurements per run is 10^4), suggesting that a smaller number of reads could be sufficient for these problems. Conversely, in the case of the five-variable problem, QA has found the global minimum (occasionally appearing more than once) only half of the times, while SA has consistently discovered it, although in a slightly longer execution time. Ultimately, neither QA nor SA have ever succeeded in finding the optimum solution for the largest problems ($n = 9$). Nonetheless, the solutions found were generally of good quality (in terms of QUBO image value), particularly those returned by SA, even though its execution time was again slightly higher. In addition, it is worth observing that solutions with a superior score to that of the expected solution have been discovered for the Waste problem. Specifically, this has consistently happened when using SA (whose average result value is greater than 1.0), and occasionally when using QA. In practice, the presence of a node with three parents in the expected solution penalizes it, allowing other solutions satisfying the maximum parent constraint ($m \leq 2$) to have a better score.

Moreover, the annealing time's influence on the performance of QA in the same experiments has been examined. The results obtained with annealing times of 1 μs and 99 μs are shown in Table 5.6. Basically, an increased annealing time has not enhanced the performance on the smallest problem (MHP). However, it has yielded better results on average, with only a marginal increase in execution

Table 5.6: Comparison of quantum annealing performance on several problems for different values of annealing time, using *Exp* datasets of size $N = 10^4$ and 10^4 reads. The number of runs is 10 (taken from [130]).

		Annealing time per sample 1 μs				Annealing time per sample 99 μs			
n	Problem	Success rate	Average result	Avg. resol. time (s)	Average # exp. sol.	Success rate	Average result	Avg. resol. time (s)	Average # exp. sol.
3	MHP	1.00	1.0000	0.78	304.70	1.00	1.0000	1.76	215.90
4	LC4Vars	0.90	0.9997	0.81	5.20	1.00	1.0000	1.77	11.40
5	LC	0.40	0.9980	0.87	0.50	0.50	0.9987	1.80	0.60
9	Waste	0.00	0.9619	1.15	0.00	0.00	0.9898	2.09	0.00
9	Waste2P	0.00	0.9473	1.15	0.00	0.00	0.9754	2.17	0.00
9	Waste2PDep	0.00	0.9633	1.33	0.00	0.00	0.9780	2.10	0.00

time, for all other problems. Specifically, for problems featuring four (LC4Vars) and five (LC) Bayesian variables, a higher success rate and an increased number of occurrences of the optimum solution have been also observed.

Lastly, as SA does not have restrictions on the number of reads, additional experiments have been conducted on the Waste2PDep problem using the *Exp* dataset with $N = 10^4$, and 10^5 and 10^6 reads. While the execution time has exhibited a linear increase, no significant enhancement in performance has been noted.

5.3.5 Divide et Impera Results

As discussed in Section 5.3.1, the divide et impera approach has been evaluated on two problems utilized for testing O'Gorman's algorithm, i.e., LC and Waste (in their main variant), and on an additional problem, namely, Alarm. The results obtained are described in the following sections.

5.3.5.1 Execution Speedup and Timing

First of all, an analysis of the speedup achieved through the technique outlined in Section 5.1.3 has been conducted. Specifically, for each considered problem, a single *Exp* dataset with $N = 10^4$ and one run have been employed. Concerning the number of variables for each subproblem (k), all values ranging from three (which is the minimum reasonable value, refer to Section 5.2) to n have been evaluated, with $k = n$ corresponding to the direct application of the implementation of O'Gorman's algorithm. The results obtained are shown in Table 5.7, with the reported time values encompassing the formulation of subproblems and the construction of the QUBO matrices (hence, both the resolution of the subproblems and the reconstruction of the final solution are excluded). In particular, only LC and Waste have been taken into account here, as collecting times without speedup for Alarm using the available machine would have been unfeasible. Regarding LC, the time needed has been reduced on average by approximately 9 times for the divide et impera approach and by approximately 5.6 times for O'Gorman's algorithm. In contrast, for the Waste problem, the speedup has been more pronounced because of the presence of more variables and/or subproblems, resulting in an average speedup of approximately 38.4 times for the divide et impera approach and about 17 times for O'Gorman's algorithm.

Conversely, Table 5.8 presents some statistics calculated on analogous time values, for which four distinct non-*Exp* datasets with $N = 10^4$ have been used. The exclusion of the *Exp* datasets is due to their potential lower number of samples, as elucidated in Section 5.3.2. In addition, since all the time values are referred to the optimized code in this case, the Alarm problem has also been taken into account, but the maximum k value that has been evaluated for it is 9 (the time required is still high). Lastly, it is worth observing that the limit case, which corresponds to the direct application of O'Gorman's algorithm ($k = n$), has not been considered here. Specifically, the average time is influenced by both the size and the number of subproblems. In fact, the average time per subproblem

Table 5.7: Speedup achieved for different k values on LC and Waste, using an *Exp* datasets of size $N = 10^4$ and a single run. The time values, expressed in seconds, include the subproblems formulation and the QUBO matrices construction. In particular, D.e.I. = divide et impera, O'G. = O'Gorman (taken from [130]).

Problem	k	Subproblems #	Time (no speedup)	Time (with speedup)	Speedup
LC ($n = 5$)	3 (D.e.I.)	10	27.66	4.54	6.09x
	4 (D.e.I.)	5	54.4	4.55	11.96x
	5 (O'G.)	1	21.41	3.84	5.58x
Waste ($n = 9$)	3 (D.e.I.)	84	237.5	10.27	23.13x
	4 (D.e.I.)	126	1754.35	32.62	53.78x
	5 (D.e.I.)	126	2704.54	66.02	40.97x
	6 (D.e.I.)	84	2907.48	78.8	36.90x
	7 (D.e.I.)	36	2186.56	54.91	39.82x
	8 (D.e.I.)	9	858.57	23.97	35.82x
	9 (O'G.)	1	149.58	8.8	17x

Table 5.8: Statistics on subproblems formulation and QUBO matrices construction time for different k values. Specifically, 4 non-*Exp* datasets of size $N = 10^4$ have been used for each problem (one run for each dataset). In addition, the time values, expressed in seconds, refer to the optimized code (i.e., with speedup) (taken from [130]).

Problem	k	Subproblems #	Average time	STD time	CV time	Average time per subproblem
LC ($n = 5$)	3	10	1.32	0.02	0.014	0.132
	4	5	1.28	0.03	0.022	0.256
Waste ($n = 9$)	3	84	3.72	0.15	0.040	0.044
	4	126	8.89	0.10	0.011	0.071
	5	126	16.54	0.28	0.017	0.131
	6	84	18.80	0.22	0.012	0.224
	7	36	13.55	0.16	0.012	0.376
	8	9	5.92	0.13	0.022	0.658
Alarm ($n = 15$)	3	455	26.68	0.20	0.007	0.059
	4	1365	203.15	0.62	0.003	0.149
	5	3003	1063.36	4.00	0.004	0.354
	6	5005	2952.22	7.08	0.002	0.590
	7	6435	6548.12	235.37	0.036	1.018
	8	6435	10361.01	489.65	0.047	1.610
	9	5005	11370.70	112.19	0.010	2.272

increases with the subproblems size, as shown in the last column. Furthermore, an examination of the standard deviation (STD) and the coefficient of variation (CV) reveals that the variance in the input data has a negligible impact on the time values. In particular, the CV value is consistently below 0.05.

5.3.5.2 Performance

To assess the effectiveness of the divide et impera approach, the same setup has been employed for all problems. In particular, this setup consists of a single *Exp* dataset with a size of $N = 10^4$, five runs, and a number of variables per subproblem (k) ranging from three (which is the minimum reasonable value) to n (which is the total number of Bayesian variables), with the upper limit corresponding to the direct execution of O'Gorman's algorithm. Concerning the methods for solving the QUBO formulation, only SA and QA have been used in these experiments. In fact, due to the high time requirements, the application of ES would have been unfeasible for subproblems created with a high k value. Additionally, 100 reads and an annealing time of 1 μs have been employed for QA, given the limited quantum resources available and the high number of subproblems to solve across all experiments. To ensure a fair comparison, the number of reads for SA has been also set to 100. Eventually, it is worth remarking that, in these experiments, only the second reconstruction strategy devised for the divide et impera approach has been used, as elucidated in Section 5.2.

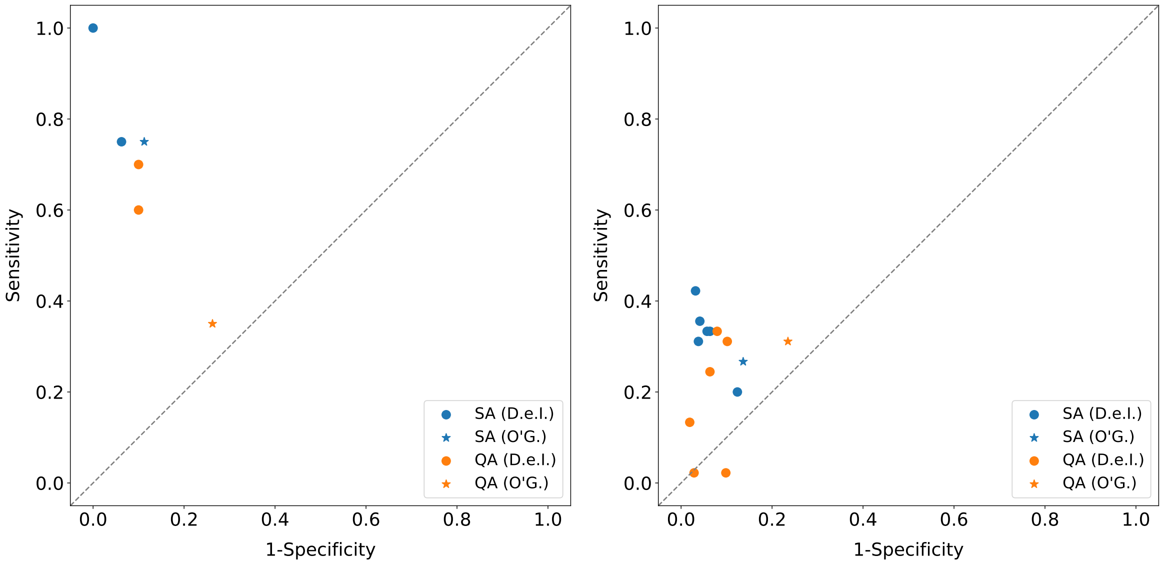
Beginning with LC, the obtained results are detailed in Table 5.9, accompanied by a "ROC curve"-like plot in Figure 5.4a. Essentially, SA has outperformed QA on LC, and the divide et impera approach has demonstrated superior performance to O'Gorman's algorithm for both resolution methods. In fact, higher sensitivity and specificity correspond to better results. It is worth observing that SA, with $k = 4$, has identified the perfect solution (four correct edges and zero wrong ones) in all five runs. Furthermore, an examination of the number of unique edges found across all runs (fifth column) has revealed that, for the divide et impera approach, SA tends to consistently find the same correct and incorrect edges, while QA exhibits more variability, as does O'Gorman's algorithm.

Regarding the Waste problem, whose results are outlined in Table 5.10 and depicted in Figure 5.4b, both SA and QA exhibit worse overall performance. In particular, SA has outperformed QA, except for $k = 3$, where QA has achieved better results on average. Conversely, the superiority of the divide et impera approach over the direct application of O'Gorman's algorithm has turned out to be less pronounced. Specifically, for SA, the divide et impera approach has outperformed O'Gorman's algorithm for nearly all k values. Concerning QA, despite a relatively high sensitivity compared to all

QA results, O'Gorman's algorithm has achieved a low specificity. In the end, for both SA and QA, there exists at least one k value for which the divide et impera approach has outperformed O'Gorman's algorithm. However, the perfect solution has never been found. In detail, SA, with $k = 6$, has yielded the best results, discovering four correct edges out of nine in nearly all runs, with an average of only two incorrect edges. Additionally, the maximum number of correct edges identified in a single run is 6 (SA with $k = 4$). However, it is worth highlighting that this problem has undergone a discretization procedure and has a node with three parents. Lastly, the insights gained from LC regarding the number of unique edges found are applicable also to the Waste problem. The only difference consists in the correct edges found by the divide et impera approach with QA, which tend to be consistent across runs.

Table 5.9: Results achieved by the divide et impera approach on the LC problem, for different numbers of variables per subproblem (k) and methods (SA/QA), using an *Exp* dataset of size $N = 10^4$, five runs, 100 reads for SA, and 100 reads and 1 μs of annealing time for QA. The last k value (5) corresponds to the direct application of the implementation of O'Gorman's algorithm. In particular, D.e.I. = divide et impera, O'G. = O'Gorman (taken from [130]).

LC ($n = 5$, edges $= 4$)											
k	Method	Metric	# for each run					# unique	Average #	Sensitivity	Specificity
3 (D.e.I.)	SA	Correct edges	2	2	4	4	3	4	3	0.75	0.94
		Wrong edges	2	2	0	0	1	2	1		
	QA	Correct edges	1	4	4	1	2	4	2.4	0.60	0.90
		Wrong edges	3	0	0	3	2	4	1.6		
4 (D.e.I.)	SA	Correct edges	4	4	4	4	4	4	4	1.00	1.00
		Wrong edges	0	0	0	0	0	0	0		
	QA	Correct edges	3	3	3	3	2	4	2.8	0.70	0.90
		Wrong edges	2	2	1	1	2	5	1.6		
5 (O'G.)	SA	Correct edges	3	4	3	3	2	4	3	0.75	0.89
		Wrong edges	2	0	2	2	3	4	1.8		
	QA	Correct edges	1	2	0	2	2	3	1.4	0.35	0.74
		Wrong edges	5	5	6	2	3	11	4.2		



(a) LC.

(b) Waste.

Figure 5.4: Sensitivity versus (1 - Specificity) for the LC problem (a) and the Waste problem (b). These plots result from the data reported in Tables 5.9 and 5.10, respectively (figures taken from [130]).

Table 5.10: Results achieved by the divide et impera approach on the Waste problem, for different numbers of variables per subproblem (k) and methods (SA/QA), using an *Exp* dataset of size $N = 10^4$, five runs, 100 reads for SA, and 100 reads and 1 μs of annealing time for QA. The last k value (9) corresponds to the direct application of the implementation of O'Gorman's algorithm. In particular, D.e.I. = divide et impera, O'G. = O'Gorman, Sens. = sensitivity, Spec. = specificity (taken from [130]).

Waste ($n = 9$, edges$\,=9$)												
k	Method	Metric	# for each run					# unique	Average #	Sens.	Spec.
3 (D.e.I.)	SA	Correct edges	2	2	2	2	1	2	1.8	0.20	0.88
		Wrong edges	8	7	7	8	9	10	7.8		
	QA	Correct edges	1	4	3	4	2	6	2.8	0.31	0.90
		Wrong edges	8	6	6	5	7	13	6.4		
4 (D.e.I.)	SA	Correct edges	2	6	2	3	2	6	3	0.33	0.94
		Wrong edges	5	1	5	4	5	5	4		
	QA	Correct edges	4	1	3	3	4	6	3	0.33	0.92
		Wrong edges	4	7	4	5	5	10	5		
5 (D.e.I.)	SA	Correct edges	2	3	4	3	3	5	3	0.33	0.94
		Wrong edges	5	4	3	2	4	7	3.6		
	QA	Correct edges	3	5	2	0	1	5	2.2	0.24	0.94
		Wrong edges	4	2	5	4	5	10	4.0		
6 (D.e.I.)	SA	Correct edges	4	4	4	4	3	5	3.8	0.42	0.97
		Wrong edges	2	1	3	1	3	4	2		
	QA	Correct edges	1	1	1	2	1	3	1.2	0.13	0.98
		Wrong edges	1	2	1	0	2	5	1.2		
7 (D.e.I.)	SA	Correct edges	5	4	1	3	3	5	3.2	0.36	0.96
		Wrong edges	1	1	5	3	3	6	2.6		
	QA	Correct edges	0	0	0	1	0	1	0.2	0.02	0.97
		Wrong edges	2	3	1	2	1	8	1.8		
8 (D.e.I.)	SA	Correct edges	2	1	4	5	2	5	2.8	0.31	0.96
		Wrong edges	3	4	2	0	3	6	2.4		
	QA	Correct edges	1	0	0	0	0	1	0.2	0.02	0.90
		Wrong edges	4	8	6	6	7	25	6.2		
9 (O'G.)	SA	Correct edges	3	3	3	1	2	5	2.4	0.27	0.86
		Wrong edges	9	8	8	9	9	25	8.6		
	QA	Correct edges	2	2	3	3	4	7	2.8	0.31	0.77
		Wrong edges	15	16	15	12	16	49	14.8		

Eventually, the results pertaining to the Alarm problem are detailed in Table 5.11 and illustrated in Figure 5.5. Specifically, the divide et impera approach with QA has not been assessed on this problem due to the high number of subproblems involved and some connectivity issues resulting from the sequential submission of a huge number of QUBO problems to D-Wave's annealer. Similarly to the other problems, the divide et impera approach has achieved better results than the direct application of O'Gorman's algorithm, except for $k = 12$ (not in terms of specificity). Additionally, SA has significantly outperformed O'Gorman's algorithm with QA. Regarding the solution quality, SA with $k = 4$ has achieved the best results, identifying an average of 12.6 correct edges out of 15 (note that a variable with four parents is present). Nevertheless, the number of wrong edges, 16.4 on average, is high. This configuration has also identified the highest number of correct edges (13). Lastly, two observations must be made: the divide et impera approach tends to find a lower number of edges (both correct and incorrect) for higher k values; the same approach tends to consistently identify the same correct and incorrect edges across runs (refer to the fifth column), while the behaviour of

Table 5.11: Results achieved by the divide et impera approach on the Alarm problem, for different numbers of variables per subproblem (k), using an *Exp* dataset of size $N = 10^4$, five runs, 100 reads for SA, and 100 reads and 1 μs of annealing time for QA. The last k value (15) corresponds to the direct application of the implementation of O'Gorman's algorithm. In particular, D.e.I. = divide et impera, O'G. = O'Gorman, Sens. = sensitivity, Spec. = specificity (taken from [130]).

Alarm ($n = 15$, edges = 15)											
k	Method	Metric	# for each run					# unique	Average #	Sens.	Spec.
3	SA	Correct edges	10	10	11	10	10	12	10.2	0.68	0.85
(D.e.I.)		Wrong edges	31	31	30	29	28	38	29.8		
4	SA	Correct edges	12	13	13	12	13	13	12.6	0.84	0.92
(D.e.I.)		Wrong edges	17	16	17	16	16	20	16.4		
5	SA	Correct edges	12	12	12	12	13	13	12.2	0.81	0.94
(D.e.I.)		Wrong edges	13	13	13	12	12	15	12.6		
6	SA	Correct edges	8	7	7	7	8	8	7.4	0.49	0.93
(D.e.I.)		Wrong edges	14	14	14	14	13	14	13.8		
7	SA	Correct edges	6	6	6	6	6	6	6	0.40	0.93
(D.e.I.)		Wrong edges	13	13	13	13	13	13	13		
8	SA	Correct edges	7	6	6	7	6	7	6.4	0.43	0.94
(D.e.I.)		Wrong edges	11	11	11	11	12	12	11.2		
9	SA	Correct edges	7	6	7	7	6	7	6.6	0.44	0.96
(D.e.I.)		Wrong edges	8	9	7	7	9	10	8.0		
10	SA	Correct edges	6	6	7	7	7	7	6.6	0.44	0.96
(D.e.I.)		Wrong edges	9	8	8	7	7	9	7.8		
11	SA	Correct edges	6	7	7	7	6	7	6.6	0.44	0.96
(D.e.I.)		Wrong edges	8	8	8	8	9	9	8.2		
12	SA	Correct edges	5	6	6	5	6	6	5.6	0.37	0.95
(D.e.I.)		Wrong edges	9	9	8	9	9	10	8.8		
13	SA	Correct edges	5	7	6	7	6	8	6.2	0.41	0.96
(D.e.I.)		Wrong edges	9	6	8	7	7	9	7.4		
14	SA	Correct edges	5	7	6	7	5	9	6	0.40	0.96
(D.e.I.)		Wrong edges	10	8	6	6	7	11	7.4		
15 (O'G.)	SA	Correct edges	4	5	6	6	9	9	6	0.40	0.92
		Wrong edges	18	16	16	16	15	55	16.2		
	QA	Correct edges	2	4	5	7	1	12	3.8	0.25	0.80
		Wrong edges	39	41	36	40	38	128	38.8		

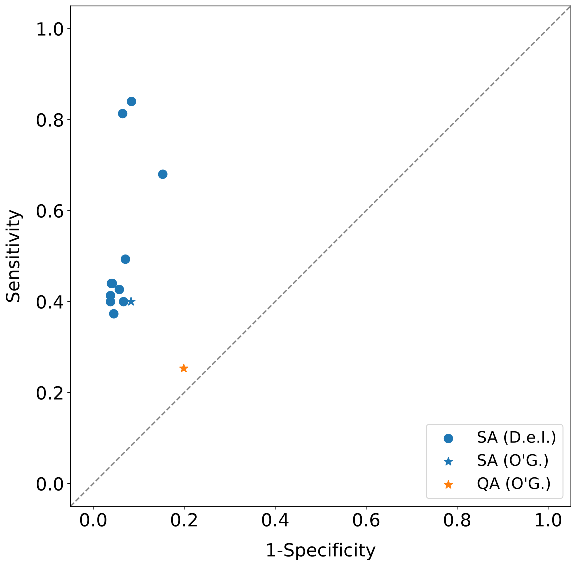


Figure 5.5: Sensitivity versus (1 - Specificity) for the Alarm problem. This plot results from the data reported in Table 5.11 (taken from [130]).

O'Gorman's algorithm is more variable. The exceptions are the wrong edges for $k = 3$ and the correct edges for O'Gorman's algorithm with SA.

5.4 Discussion

This chapter has introduced a Python implementation of O'Gorman's algorithm for tackling BNSL problems on quantum annealers, a divide et impera approach for handling BNSL instances with a larger number of variables, a complexity analysis of these algorithms, and their empirical evaluation. Specifically, to enhance the usability of O'Gorman's algorithm, some algebraic manipulations have been applied to the original formulation, and a simplified penalty lower bound has been presented. Additionally, a good configuration for the model hyperparameters has been empirically found. The results have shown the effectiveness of O'Gorman's algorithm for Bayesian networks with small sizes ($n \leq 5$) and no more than two parents per node. For larger Bayesian networks ($n = 9$), good-quality solutions in terms of QUBO image value have been found, but not the correct ones. Moreover, quantum annealing, with optimal annealing parameters, has obtained comparable or slightly inferior results with respect to simulated annealing, demonstrating the competitiveness of the current annealing architectures for this task. Regarding the divide et impera approach, the results have revealed its superiority over the direct application of O'Gorman's algorithm. For all considered problems and resolution methods, at least one k value for which the divide et impera approach has obtained better results has been found (in truth, these k values are often the majority). However, the quality of the solutions found in terms of resulting Bayesian network was not optimal. This could be attributed to the use of non-ideal annealing parameters for QA, due to the limited resources available, and of a reduced number of reads for SA, for a fair comparison. The non-optimal annealing setup could also be the reason for QA being significantly outperformed by SA in this second set of experiments. Lastly, the experiments have also proven the effectiveness of the proposed execution speedup technique.

In future work, the divide et impera approach could be evaluated on Bayesian problems exceeding the maximum size embeddable in the Pegasus architecture using O'Gorman's algorithm. Other possibilities involve using more-performing annealing parameters, and considering only a subset of the subproblems of size k. To conclude, both the code for the implementation of O'Gorman's algorithm [92] and the code for the divide et impera approach [27] are publicly available under the GPLv2 licence.

6 Local SVMs Training

This chapter is a reworked version of the article "Local Binary and Multiclass SVMs Trained on a Quantum Annealer" [129]. In practice, in this chapter, the local application of quantum-trained SVM models is introduced and empirically evaluated. In more detail, FaLK-SVM, a method for efficient local SVMs (see Section 2.6.1), has been combined with two quantum-trained SVM models, the ones detailed in Sections 2.5.1 and 2.5.2. Thus, the addressed tasks are binary and multiclass classification. In particular, local binary SVMs have already proven to be successful compared to the corresponding global models, and quantum-trained SVMs have shown comparable performance with respect to their classical counterparts. However, the quantum-trained SVMs suffer from the reduced connectivity of the available quantum annealers, which limits the size of the trainable models. The usage of local, instead of global, quantum-trained models represents a valid solution in this sense. For the empirical evaluation, D-Wave's quantum annealers and real-world datasets taken from the remote sensing domain have been employed. The results obtained demonstrate the effectiveness of the approach, also when a multiclass SVM model is considered.

6.1 Local Quantum-Trained SVMs

This section deals with the proposed approach and the implementation details. The code is available at `https://github.com/ZarHenry96/local-qtrained-svms`.

6.1.1 Approach

Quantum-trained support vector machines have demonstrated similar performance to their classical counterparts [25, 26, 118]. Nevertheless, the limited connectivity of the current quantum annealers imposes limitations on the size of the trainable SVM models. To take advantage of larger training sets, different solutions have been proposed, such as building an ensemble of SVMs [118], or weighting the models found by the annealer based on the performance achieved on a (large) validation set [26] (more details are available in Sections 2.5.1 and 2.5.2). The approach proposed here consists in applying the quantum-trained SVM models locally. Indeed, in the entirely classical realm, local SVMs have proven to be superior compared to the corresponding global models (for more information, refer to Section 2.6). Moreover, since the local models are trained only on the k-neighborhood of the model centre, large training sets represent no more an issue.

In practice, in this work, FaLK-SVM, the method for efficient local SVMs described in Section 2.6.1, is combined with two quantum-trained SVM models: the quantum-trained SVM for binary classification illustrated in Section 2.5.1 (QBSVM), and the quantum-trained SVM for multiclass classification detailed in Section 2.5.2 (QMSVM). The resulting flow is quite straightforward. Indeed, the only difference with respect to the standard FaLK-SVM lies in the local models used, which are trained on a quantum annealer and executed classically. Actually, another novelty of this work is the evaluation of FaLK-SVM with local single-step multiclass classification models like QMSVM and CS SVM, which has been considered mainly for comparative purposes (QMSVM is based on CS SVM, as reported in Section 2.5.2). In fact, FaLK-SVM has already been evaluated on a multiclass classification task, but using a one-against-one (OAO) approach with local binary SVMs [105].

6.1.2 Implementation

The approach presented in the previous section has been implemented starting from the FaLK-SVM implementation provided by Segata [103]. In particular, the programming language used for that implementation of FaLK-SVM is C++, while the code for QBSVM [117] and QMSVM [24] is written in Python, as it needs to interact with D-Wave's quantum annealers via the Ocean-SDK, whose APIs are only available in Python. Hence, in order to interface FaLK-SVM with the quantum-trained models, a Python class named *PythonSVM* has been developed and embedded in the FaLK-SVM

C++ framework, enabling the execution of Python code within the C++ application. Specifically, the utilities provided by the *Python.h* header file have been used for this purpose.

From a high-level approach-related perspective, two aspects are worth to be mentioned. First of all, in order to reduce the training time, the reuse of the QUBO matrix embedding has been introduced. Essentially, the first time a QUBO matrix of a certain size is submitted to the quantum annealer, the embedding for a complete matrix of the same size is computed, applied, and stored in memory. In all subsequent calls with QUBO matrices of the same size, the previously computed embedding is loaded and applied. This is particularly advantageous since the size of the QUBO matrix is the same for all local models. Secondly, two interesting features have been implemented, but not used in the experiments presented here. The former is the local application of the strategies detailed in Sections 2.5.1 and 2.5.2 for taking advantage of larger datasets, which allows increasing the size of the neighborhoods used for training of the local models (that is limited by the annealer connectivity). The latter is related to multiclass classification tasks and consists in the dynamic selection of the local model to be used in a k-neighborhood depending on the number of classes present. Indeed, for two classes, QBSVM requires half of the binary variables compared to QMSVM.

From a low-level model-related perspective, some changes have been made to the original codes. On the FaLK-SVM side, the performance metrics calculation and the criterion used to check the class balance of the m k-neighborhoods employed in the FaLK-SVMl's local model selection procedure have been extended to multiclass classification scenarios. Concerning the grid-search local model selection, support for the parameters of the quantum-trained models has been added; in addition, the number of folds κ for the internal custom κ-fold cross-validation (10 by default) and the number of samples on which the performance of the m models are evaluated ($\frac{k'}{2}$ by default) have been parametrized. Lastly, a data standardization procedure has been introduced in the external canonical κ-fold cross-validation used for evaluating the performance of FaLK-SVM. On the local models side, a post-selection procedure has been implemented for the bias (b) of QBSVM. In practice, all values in the interval $[-10, +10]$, with a step of 0.1, are evaluated on the training set in order to find the best one. Some preliminary experiments have shown that this works extremely better than applying Equation (2.27) for computing b. Regarding CS SVM, a C implementation of the model [58] has been employed. In the local version, the CS SVM executables are directly called from the Python code (after locally mapping the class labels to $\{1, \ldots, C\}$, if required). Obviously, more efficient solutions exist. For instance, in the large-scale experiments presented in Section 6.2, a custom version of the CS SVM has been utilized in the local setup. This faster version, trained using a slightly modified C code and executed using new Python code, cannot be distributed due to the license of the original CS SVM implementation [58].

6.2 Empirical Evaluation

This section deals with the methods evaluated, the datasets used, the experimental setup employed, and the results obtained. In particular, the classical side of the experiments has been executed on a shared machine equipped with an Intel Xeon Gold 6238R processor running at 2.20 GHz and 125 GB of RAM, while the quantum side has been executed on the Advantage system 5.3/5.4 provided by D-Wave, a quantum annealer located at Forschungszentrum Jülich.

6.2.1 Methods

Table 6.1 reports all the methods considered in this work. In detail, four local methods (Table 6.1a) and four global methods (Table 6.1b) have been taken into account here. The local ones correspond to the combination of FaLK-SVMl (the version of FaLK-SVM including the local model selection procedure, see Section 2.6.1) with different local models: a binary and a multiclass classically-trained SVMs, i.e., SVM and CS SVM, and their quantum-trained counterparts, namely, QBSVM and QMSVM. Note that *FaLK-SVMl (C)* corresponds to the original FaLK-SVMl implementation; additional details on the other local methods can be found in Section 6.1. Concerning the global methods, they consist in the global application of the just mentioned classically- and quantum-trained SVM models. Specifically, for the standard binary SVM, the implementation provided by LibSVM [17] version 2.88 (the one integrated in the original FaLK-SVM framework) has been used. Instead, for QBSVM and

Table 6.1: Local (a) and global (b) methods considered (taken from [129]).

(a) Local methods.

Name	Classification type	Local model	Local model training
FaLK-SVMl (C)	Binary	SVM	Classical
FaLK-SVMl (QB)	Binary	QBSVM	Quantum
FaLK-SVMl (CS)	Multiclass	CS SVM	Classical
FaLK-SVMl (QM)	Multiclass	QMSVM	Quantum

(b) Global methods.

Name	Classification type	Model training
SVM	Binary	Classical
QBSVM	Binary	Quantum
CS SVM	Multiclass	Classical
QMSVM	Multiclass	Quantum

QMSVM, the strategies for managing large datasets detailed in Sections 2.5.1 and 2.5.2 have been applied. Otherwise, it would have not been possible to train them on the considered datasets, given the properties of the available quantum annealers and the dataset sizes used.

6.2.2 Datasets

The methods listed in Table 6.1 have been evaluated on datasets belonging to the remote sensing domain, a domain to which both FaLK-SVM and the quantum-trained SVMs have already been applied with good results [16, 25, 26, 105]. In particular, the datasets used here have been generated starting from the SemCity Toulouse [93] and ISPRS Potsdam [57] datasets, two datasets consisting of multispectral images with multiple classes (used also in the QMSVM article [26]). Essentially, the task consists in predicting the class to which each pixel belongs. The number of features and the classes selected for binary and multiclass classification for both datasets are reported in Table 6.2. In the case of multiclass classification, the number of classes has been limited to three in order to maximize the number of samples that could be embedded in the annealer. Instead, the dataset sizes used are specified in Section 6.2.3, as they are experiment-dependent. Concerning the datasets generation, an equal (or almost equal, depending on the desired size and the number of classes) number of samples for each class has been randomly sampled from tile 4 in the case of Toulouse, and tile 6.9 in the case of Potsdam. The exception is represented by the last experiment, i.e., the large-scale one (more details on the experiments are provided in the following section). In that case, the training set has been generated by sampling an equal number of data points for each class from each of the 24 Potsdam tiles labelled as training. Additionally, two different test sets have been prepared for that experiment: the former consists of data points randomly sampled in the usual way from Potsdam tile 5.13 (which is not a training tile); the latter, meant for visualization, includes all the data points that belong to the classes of interest and are located inside a 1000 × 1000 pixels square within the same tile.

Table 6.2: Number of features and selected classes for both basis datasets (taken from [129]).

Dataset name	Features number	Selected classes (binary)	Selected classes (multiclass)
Toulouse	8	building, pervious surface	building, pervious surface, water
Potsdam	5	low vegetation, tree	building, low vegetation, tree

6.2.3 Experimental Setup

In this work, four experiments with different goals have been conducted. Specifically, in the first experiment, the performance of all binary classification methods considered are evaluated and compared. In the second experiment, the same thing is done with the multiclass classification methods. Instead, in the third experiment, the performance scaling of only the local and global entirely classical methods (both binary and multiclass) is analysed; the methods involving quantum-trained models have not

Table 6.3: Datasets sizes used in each of the four experiments. The | symbol in the last row separates the training set size from the (two) test sets sizes (taken from [129]).

Experiment	Basis datasets	Datasets sizes
I - binary classification	Toulouse, Potsdam	500
II - multiclass classification	Toulouse, Potsdam	150, 500
III - performance scaling	Toulouse, Potsdam	1000, 1500, 2000, 10000, 15000, 40000
IV - large scale (multiclass)	Potsdam	11016 | (300000, 871188[a])

[a] The occurrences of the three classes in this test set are 389900, 343438, and 137850, respectively.

been taken into account in this case as the quantum annealing resource consumption would have been too high. In the last experiment, the performance of all multiclass classification methods, including the ones involving quantum-trained models, are evaluated (and visualized) on a large-scale dataset.

In the first three experiments, a κ-fold cross-validation procedure with ten folds ($\kappa = 10$) has been used to evaluate the performance of the methods. In practice, the input dataset is split into κ subsets. Then, $\kappa - 1$ folds form the training set, while the last one becomes the test set. This last step is iterated until all folds have been used once as the test set. Specifically, the "stratified" κ-fold cross-validation, which tries to preserve the original class ratio in the folds, has been utilized here. Additionally, for a fair comparison, the same datasets splits have been used for the different methods. Instead, in the last experiment, no cross-validation procedure has been utilized, as the input data was already split into training set and test sets. The datasets sizes used in each of the four experiments are reported in Table 6.3; as explained in Section 6.2.2, the large-scale experiment differs from the other ones in the dataset generation method employed. Furthermore, in all experiments, a data standardization procedure (subtraction of the mean and division by the standard deviation) has been applied to the training and test data features before training and executing the (local/global) methods.

Concerning the parameters values used for the different methods, they are reported in Table 6.4. Let us focus first on the binary classification methods (Table 6.4a). The size of the training neighborhood (k) for the local methods has been set to a value (80) close to the maximum number of samples embeddable in the current quantum annealers using the QBSVM QUBO formulation (given the values of B and K, and computing the embedding for a complete matrix). Additionally, a relatively-high level of local models overlapping (controlled by k') has been used. Concerning FaLK-SVMl's local model selection, 8 local models (m) and 5 folds (κ *int.*) have been utilized; additionally, all k' samples have been used in the evaluation of the m local models. In particular, the grid search has been applied only to the Gaussian kernel width γ (see Equation 2.25), to identify the best value between

Table 6.4: Parameters values used for binary (a) and multiclass (b) classification methods. In particular, "int." stands for internal, and the m value between parentheses in (b) is the one used in the large-scale experiment (taken from [129]).

(a) Binary classification methods.

Method	k	k'	m	κ (int.)	Kernel	γ	A	B	K	ξ	S
FaLK-SVMl (C)	80	60	8	5	Gaussian	$-0.5, 1$	3	-	-	-	-
FaLK-SVMl (QB)	80	60	8	5	Gaussian	$-0.5, 1$	-	2	2	1	100
SVM	-	-	-	-	Gaussian	1	3	-	-	-	-
QBSVM	-	-	-	-	Gaussian	1	-	2	2	1	100

(b) Multiclass classification methods.

Method	k	k'	m	κ (int.)	Kernel	γ	A	K	μ	β	S
FaLK-SVMl (CS)	24	18	8 (10)	3	Gaussian	$-0.5, 1$	1	-	-	-	-
FaLK-SVMl (QM)	24	18	8 (10)	3	Gaussian	$-0.5, 1$	-	2	1	1	100
CS SVM	-	-	-	-	Gaussian	1	1	-	-	-	-
QMSVM	-	-	-	-	Gaussian	1	-	2	1	1	100

-0.5 and 1, where -0.5 corresponds to the usage of the median of the distances distribution in the neighborhood as the kernel width. Hence, for $\gamma = -0.5$, each local SVM model might have a different kernel width (additional details on this can be found in the FaLK-SVM article [104]). Instead, for the global methods, γ has been set to 1 (their implementations do not support the usage of the median as the kernel width). For a fair comparison between classically- and quantum-trained models, the SVM cost parameter (A) has been set to 3 (in the QBSVM formulation, A is determined by B and K). Regarding the QBSVM-related parameters, the encoding basis (B) and the number of binary variables per coefficient (K) have been kept small, to allow the embedding of a sufficiently large number of training samples. Moreover, the penalty coefficient (ξ) has been set to 1 (the same value used for QMSVM, where it is denoted as μ), and the best 100 solutions returned by the annealer have been considered for averaging (S). Eventually, it is worth mentioning that, for the global application of QBSVM, a stratified training data split has been performed, with a number of samples per slice equal to k (except for the last slice, potentially having less samples).

Similar considerations hold for the multiclass classification methods (Table 6.4b). Indeed, the size of the training neighborhood (k) for the local methods has been set to a value (24) close to the maximum number of samples embeddable in the current quantum annealers using the QMSVM QUBO formulation (given the values of $C = 3$ and K, and computing the embedding for a complete matrix). Regarding the local model selection, 3 instead of 5 folds (κ *int.*) have been used, as the number of samples involved is smaller. Additionally, only in the large-scale experiment, 10 instead of 8 local models (m) have been employed. Instead, the CS SVM cost parameter (A) has been set to 1 in order to have a fair comparison with the methods based on the QMSVM formulation. Indeed, the following relationship holds: $A = 1/\beta$. Concerning the QMSVM-related parameters (K, μ, β, S), the same configuration used for the QMSVM article [26] (where K is denoted as B) has been utilized. Specifically, the accuracy threshold definition ($thr = 0.2*\min(acc)+0.8*\max(acc)$) and the *multiplier* value (10) for the combination of the S best solutions found by the annealer have also been taken from that work. Instead, the *max_min_ratio* value used for pruning the small matrix coefficients has been set here to a high value (1000), making the pruning procedure ineffective (in this work, the embedding is computed for a complete matrix). Lastly, it is worth observing that, for the global application of QMSVM, a stratified random selection of the training samples has been performed, with a number of selected samples equal to k.

The quantum annealing parameters values used in the experiments are reported in Table 6.5. In particular, this is the same configuration that has been used in the QMSVM article [26]. With the setup utilized in this work, the training of a single QBSVM model takes approximately 0.360 s of quantum annealing time (the number of binary variables to embed is 160). A slightly lower time interval is required for a QMSVM model (for which the number of binary variables to embed is 144).

Table 6.5: Quantum annealing parameters values used in the experiments (taken from [129]).

Number of reads	Annealing time	Chain strength
1000	$200\mu s$	1

6.2.4 Results

In this work, the performance metric considered for the methods evaluation is the classification accuracy, which is defined as

$$accuracy = \frac{number\ of\ correctly\ classified\ instances}{total\ number\ of\ instances} . \tag{6.1}$$

Specifically, in the first three experiments, where the κ-fold cross-validation has been used, the accuracy computed on the entire dataset (considering the predictions of the κ models) is reported. Actually, since a stratified κ-fold cross-validation has been employed, the folds might not have exactly the same number of elements. Consequently, the reported accuracy might not coincide exactly with the average accuracy over folds. However, the difference is negligible. Instead, in the last experiment, the accuracy obtained on the two test sets is shown. Additionally, for the second test set, given its imbalance in terms of class occurrences, two other metrics are reported. The metrics in question are the balanced accuracy [101], which corresponds to the average recall over classes, and the F1 score [102] (i.e., the harmonic mean of precision and recall) averaged over classes.

6.2.4.1 Binary Classification

In the first experiment, the performance of the considered binary classification methods have been evaluated. The results obtained are reported in Table 6.6. Essentially, in the case of Toulouse, all methods have achieved good results, but the entirely classical methods have shown better performance overall and the local methods have outperformed their global counterparts. Instead, in the case of Potsdam, the methods have achieved worse results overall, with the classical SVM obtaining the best performance and FaLK-SVMl (QB) outperforming its classical counterpart (although not by much). Regarding QBSVM, it has performed the worst among the methods tested also in this case. Hence, the local application of QBSVM has turned out to be effective. Indeed, it has obtained results not too far, if not better, than those of its classical counterpart.

Table 6.6: Accuracy achieved by binary classification methods (columns) on different datasets (rows) of size 500. For FaLK-SVMl, the average number of local models (over folds) and the size of the local models are reported between square brackets; instead, for QBSVM, the number of models (all with size 80, except the last one with size 50) is shown between square brackets (taken from [129]).

	FaLK-SVMl (C)	FaLK-SVMl (QB)	SVM	QBSVM
Toulouse (500)	92.4% [15.9, 80]	89.8% [15.9, 80]	91.8%	88.2% [6,]
Potsdam (500)	69.8% [16.5, 80]	70.4% [16.5, 80]	73.0%	68.6% [6,]

6.2.4.2 Multiclass Classification

In the second experiment, the performance of the multiclass classification methods taken into account have been evaluated. The results obtained are reported in Table 6.7. Let us consider first the smaller datasets (size 150), for which the average number of local models is comparable to that of the binary classification methods in the previous experiment. In detail, in the case of Toulouse, the local methods have achieved the best results and the entirely classical ones (both local and global) have outperformed their quantum-trained counterparts. Overall, the performance obtained are good. In the case of Potsdam, the accuracy values are lower, but the trend is similar. The only exception is represented by FaLK-SVMl (QM), which has been outperformed not only by FaLK-SVMl (CS) but also by CS SVM. However, when considering larger datasets (size 500), FaLK-SVMl (QM) has turned out to be the best-performing method, achieving results better than both FaLK-SVMl (CS) and CS SVM. Concerning the relative performance-based ordering of the other methods, it is the same. Overall, the larger dataset size has been beneficial especially in the case of Toulouse. Lastly, also in this second experiment, the global quantum-trained model (QMSVM) is the one that has shown the worst performance among the methods tested. In practice, this second experiment has demonstrated the effectiveness of applying locally both classically- and quantum-trained single-step multiclass SVMs, with the quantum-trained ones turning out to be slightly better in the case of larger datasets.

Table 6.7: Accuracy achieved by multiclass classification methods (columns) on datasets (rows) of two different sizes. For FaLK-SVMl, the average number of local models (over folds) and the size of the local models are reported between square brackets; instead, for QMSVM, the size of the model is shown between square brackets (taken from [129]).

	FaLK-SVMl (CS)	FaLK-SVMl (QM)	CS SVM	QMSVM
Toulouse (150)	79.3% [14.2, 24]	77.3% [14.2, 24]	76.0%	72.0% [, 24]
Potsdam (150)	72.0% [14.3, 24]	66.0% [14.3, 24]	70.0%	60.7% [, 24]
Toulouse (500)	85.2% [50.2, 24]	85.4% [50.2, 24]	80.8%	73.0% [, 24]
Potsdam (500)	72.6% [47.3, 24]	72.8% [47.3, 24]	71.4%	58.8% [, 24]

6.2.4.3 Performance Scaling (Classical Methods)

In the third experiment, a performance scaling analysis has been conducted on the classical (binary and multiclass) local methods, considering their global counterparts for comparison. Indeed, in the first two experiments, FaLK-SVMl (QB) and FaLK-SVMl (QM) have achieved results not too far (except in one case) from those of their classical counterparts, which can then be exploited as indicators of performance. Furthermore, the quantum annealing time consumption would have been too high for the available resources.

The results obtained are reported in Tables 6.8 and 6.9. Let us focus first on binary classification (Table 6.8). In the case of Toulouse, the performance of both FaLK-SVMl (C) and SVM have turned out to be quite stable when increasing the dataset size, with a little worsening and a little improvement, respectively, compared to the (baseline) size 500. In particular, the accuracy values for the dataset of size 500 were already very high. Instead, in the case of Potsdam, where the original performance were worse, an improvement can be observed for both FaLK-SVMl (C) and SVM when increasing the dataset size. Moreover, the improvement is more pronounced but less constant for FaLK-SVMl (C), and less pronounced but more constant (after an initial drop) for SVM. Concerning multiclass classification (Table 6.9), the situation is the following: in both cases (Toulouse and Potsdam), the performance of FaLK-SVMl (CS) almost always improves by moving from one dataset size to the next one, while the performance of CS SVM either significantly improves at the beginning and then remains quite stable (Toulouse) or tends to fluctuate around the initial value (Potsdam). Additionally, the improvement for FaLK-SVMl (CS) is slightly more pronounced for Toulouse, despite the little drop for sizes 2000 and 10000. In practice, this experiment has demonstrated that local methods can take advantage of larger datasets, especially FaLK-SVMl (CS), which has also outperformed CS SVM in

Table 6.8: Accuracy achieved by classical binary classification methods (columns) on datasets with different sizes (rows). For comparison, the pertinent results presented in Table 6.6 are reported here. In particular, for FaLK-SVMl (C), the average number of local models (over folds) and the size of the local models are reported between square brackets (taken from [129]).

(a) Toulouse (binary).

	FaLK-SVMl (C)	SVM
500	92.4% [15.9, 80]	91.8%
1000	92.1% [33.7, 80]	91.9%
1500	91.9% [54.4, 80]	92.4%
2000	91.5% [73.8, 80]	91.7%
10000	91.6% [423.1, 80]	92.2%
15000	91.5% [645.3, 80]	92.0%
40000	91.7% [1788.9, 80]	92.1%

(b) Potsdam (binary).

	FaLK-SVMl (C)	SVM
500	69.8% [16.5, 80]	73.0%
1000	73.2% [34.6, 80]	71.8%
1500	72.3% [54.6, 80]	72.3%
2000	71.5% [74.6, 80]	72.6%
10000	74.5% [400.2, 80]	74.7%
15000	74.1% [603.4, 80]	75.0%
40000	74.8% [1641.9, 80]	75.5%

Table 6.9: Accuracy achieved by classical multiclass classification methods (columns) on datasets with different sizes (rows). For comparison, the pertinent results presented in Table 6.7 are reported here. In particular, for FaLK-SVMl (CS), the average number of local models (over folds) and the size of the local models are reported between square brackets (taken from [129]).

(a) Toulouse (multiclass).

	FaLK-SVMl (CS)	CS SVM
150	79.3% [14.2, 24]	76%
500	85.2% [50.2, 24]	80.8%
1000	87.3% [104.7, 24]	81.7%
1500	88.1% [165.9, 24]	82.1%
2000	86.4% [221.5, 24]	81.4%
10000	87.8% [1167.8, 24]	81.2%
15000	88.2% [1761.7, 24]	81.2%
40000	89.1% [4779.5, 24]	81.3%

(b) Potsdam (multiclass).

	FaLK-SVMl (CS)	CS SVM
150	72.0% [14.3, 24]	70%
500	72.6% [47.3, 24]	71.4%
1000	72.8% [100.8, 24]	69.2%
1500	73.1% [155.5, 24]	70.1%
2000	74.0% [211.5, 24]	70.4%
10000	77.7% [1079.9, 24]	69.8%
15000	77.9% [1646.0, 24]	69.6%
40000	79.0% [4465.1, 24]	69.4%

all tests. Instead, FaLK-SVMl (C) has lost almost all comparisons with SVM, although not by much, but has demonstrated good stability when the performance have not improved (Toulouse).

6.2.4.4 Large Scale (Multiclass)

In the last experiment, the performance of all multiclass classification methods have been evaluated (without κ-fold cross-validation) on a large-scale dataset based on Potsdam, characterised by one training set and two test sets (prepared as explained in Section 6.2.2). The aim consists in showcasing the performance that can be achieved by locally applying quantum-trained SVM models in a large-scale real-world scenario. For this purpose, given the limited quantum annealing resources available, only multiclass classification and Potsdam have been considered.

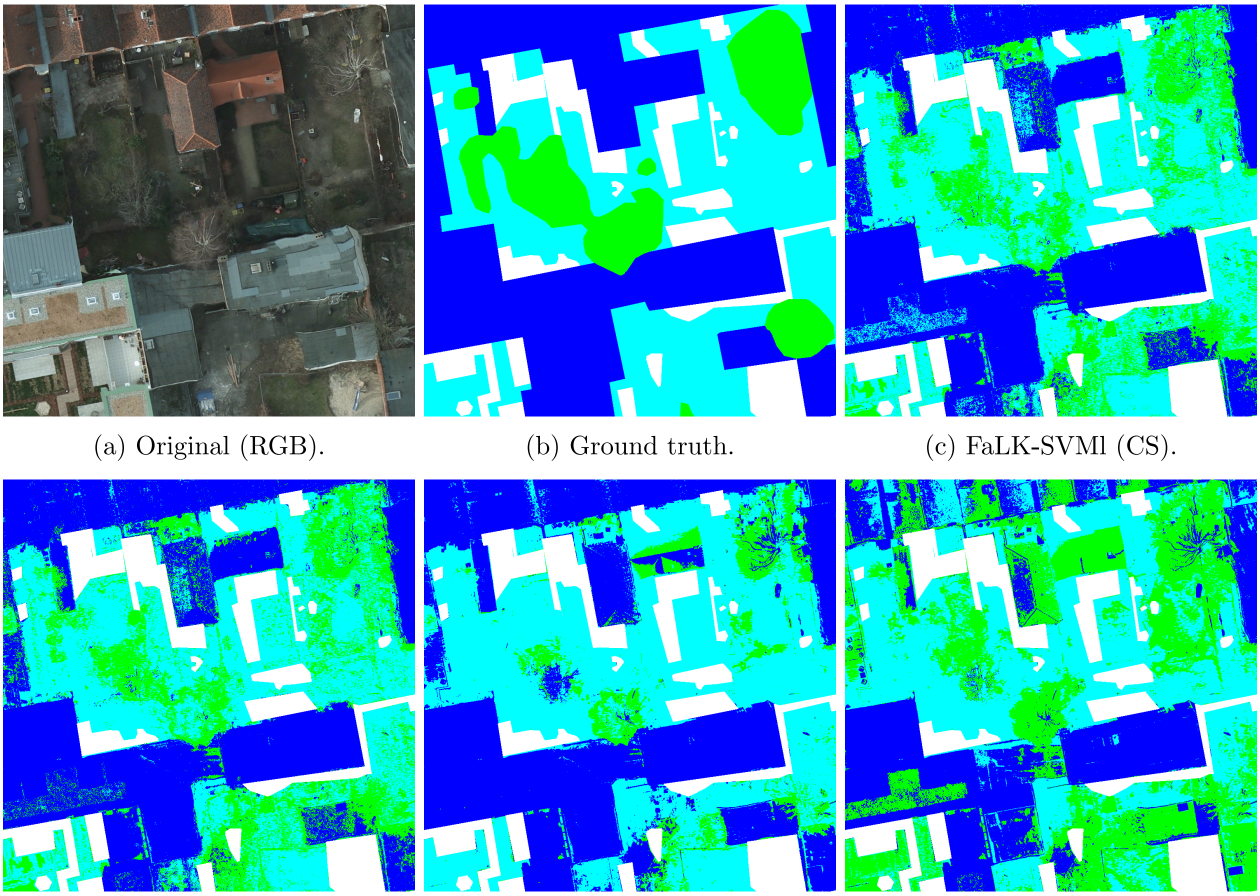
The results obtained on the first test set are reported in Table 6.10. In particular, the local methods have achieved better results than the global ones and the entirely classical methods have outperformed their quantum-trained counterparts. Considering the local methods, this last point seems to contradict what has been observed in Section 6.2.4.2, with FaLK-SVMl (QM) outperforming FaLK-SVMl (CS) for larger dataset sizes. However, the performance differences are not so big. Moreover, in this case, no κ-fold cross-validation has been used. Indeed, the training set has been built sampling data points from various tiles, and the test data points have been sampled from a different tile. Hence, the task is somehow different. Apart from this aspect, these first results are in line with the expectations based on the previous experiments outcomes. Actually, in this setup, a larger k value (100, with $k' = 75$) has also been tested for FaLK-SVM (CS). The performance have slightly worsened (accuracy = 73.6%), but the CS SVM model has already shown that it does not really benefit from a larger training set, especially in the case of Potsdam (see Section 6.2.4.3). Concerning the second test set, the results obtained are reported in Table 6.11. Unexpectedly, CS SVM has achieved the highest accuracy on this second test set, outperforming both local methods. Nevertheless, since the test set in question is unbalanced, different performance metrics should be considered. Here, the balanced accuracy and the average F1 score (over classes) have been taken into account (their values are provided in the same table). Specifically, according to both these metrics, both FaLK-SVMl (CS) and FaLK-SVMl (QM) have performed better than CS SVM, matching the trend observed for the first test set and in the previous sections. The reason behind this phenomenon is the tendency of CS SVM to predict more frequently the two most represented classes in the test set (building and low vegetation), misclassifying the less represented one (tree). This can be observed in Figure 6.1, where the predictions of the different methods are visualized. In conclusion, this last experiment has demonstrated the practical applicability of local quantum-trained SVMs (specifically, the multiclass one) in a large-scale scenario.

Table 6.10: Accuracy achieved by multiclass classification methods (columns) on a large-scale dataset based on Potsdam, characterised by 11016 training samples and 300000 test samples. For FaLK-SVMl, the number of local models and the size of the local models are reported between square brackets; instead, for QMSVM, the size of the model is shown between square brackets (taken from [129]).

	FaLK-SVMl (CS)	FaLK-SVMl (QM)	CS SVM	QMSVM
Accuracy	74.2% [1326, 24]	73.8% [1326, 24]	69.9%	55.6% [, 24]

Table 6.11: Accuracy, balanced accuracy, and average F1 score (over classes) achieved by multiclass classification methods (columns) on a second large-scale test set based on Potsdam and consisting of 871188 samples (389900, 343438, 137850). These results have been obtained using the trained models employed for Table 6.10. For FaLK-SVMl, the number of local models and the size of the local models are reported between square brackets; instead, for QMSVM, the size of the model is shown between square brackets (taken from [129]).

	FaLK-SVMl (CS)	FaLK-SVMl (QM)	CS SVM	QMSVM
Accuracy	77.7% [1326, 24]	76.6% [1326, 24]	78.6%	62.0% [, 24]
Balanced accuracy	72.4% [1326, 24]	71.7% [1326, 24]	68.5%	61.6% [, 24]
Average F1 score	71.9% [1326, 24]	70.9% [1326, 24]	69.0%	59.2% [, 24]



(a) Original (RGB).

(b) Ground truth.

(c) FaLK-SVMl (CS).

(d) FaLK-SVMl (QM).

(e) CS SVM.

(f) QMSVM.

Figure 6.1: Visualization of the results obtained by the multiclass classification methods on the second test set for the large-scale experiment. The corresponding performance metrics are provided in Table 6.11. Color legend: blue = building, light blue = low vegetation, green = tree (taken from [129]).

Indeed, the results achieved by FaLK-SVMl (QM) are quite good and not too far from those obtained by its classical counterpart.

6.3 Discussion

In this chapter, the local application of quantum-trained SVM models, with the purpose of enabling their usage on large datasets and improving their performance, has been proposed and empirically evaluated. In particular, here, a method for efficient local SVMs (FaLK-SVM) has been combined with two quantum-trained SVM models: an SVM model for binary classification (QBSVM) and an SVM model for multiclass classification (QMSVM). Additionally, for comparison, FaLK-SVM has been combined for the first time with a classical single-step multiclass classification model (CS SVM). Details about the implementation (like the post-selection procedure for the bias parameter of QBSVM) and the experimental setup have been provided. The results have shown the effectiveness of the approach, with the local applications of QBSVM and QMSVM achieving results not too far, if not better, than those of their classical counterparts. The local application of CS SVM has also achieved good results, always outperforming its global counterpart. Moreover, the performance scaling analysis conducted on the classical local methods, which can be used as indicators of performance for the quantum-trained ones, has shown that they can take advantage of larger datasets. Eventually, the last experiment has demonstrated the practical applicability of the local quantum-trained SVMs (in particular, the multiclass one) in a real-world large-scale scenario.

Future work includes the evaluation of these local quantum-trained methods on different datasets, with different parameter configurations and higher numbers of reads. Another interesting possibility is the development of a local version of the quantum-trained support vector regression model [78].

Part II

Universal Quantum Computing

7 A Local Classification Pipeline

This chapter is a reworked version of the article “Implementation and empirical evaluation of a quantum machine learning pipeline for local classification” [128]. Essentially, in this chapter, the application of a quantum locality technique as a preliminary step of a quantum machine learning model, with the goal of reducing its size and enhancing its performance, is introduced and empirically evaluated. Indeed, limiting the circuit size is of great significance in the current NISQ era [87], due to the restricted number of qubits available and the presence of noise, and the classical counterpart has already proven to be successful (see Section 2.6). Specifically, the locality considered here is the locality of data samples within the feature space, and not the spatial locality of data features utilized by quantum convolutional neural networks to process images [10, 121]. In this sense, a well-known locality technique is represented by the k-NN. In practice, the quantum pipeline taken into account comprises a quantum k-nearest neighbors algorithm based on the squared cosine similarity (see Section 3.3.1) and a quantum binary classifier based on the cosine similarity (see Section 3.4). Although the code allows executing the considered quantum models on IBM’s devices, the quantum pipeline has not been evaluated on real quantum devices, since no free large-enough device was available at the time of running the experiments. The results obtained on real-word datasets have demonstrated the effectiveness of the approach.

7.1 Quantum Pipeline

This section introduces the implemented and tested quantum pipeline, providing details about its components, implementation, and complexity. The source code can be accessed at `https://github.com/ZarHenry96/quantum-ml-pipeline`.

7.1.1 Components

The quantum pipeline assessed here comprises a quantum k-NN algorithm followed by a quantum binary classifier, with the quantum k-NN supplying the nearest neighbors as input to the subsequent model. The workflow is illustrated in Figure 7.1.

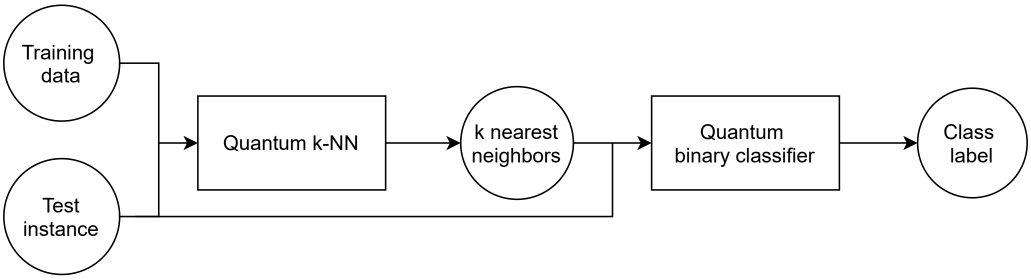


Figure 7.1: Quantum pipeline workflow overview (taken from [128]).

Regarding the quantum k-NN algorithm, numerous variants are available in the literature, as outlined in Section 3.3. Nevertheless, certain aspects must be taken into account. Firstly, variants based on the Hamming distance are not directly applicable to problems with real-valued features, as the metric in question corresponds to a distance on binary strings. Secondly, lots of variants incorporate an oracle-based algorithm derived from Grover’s, like Dürr’s or the amplitude estimation. Consequently, they necessitate a problem-specific black-box function, which impacts their ease of use and implementation. Furthermore, when using the amplitude estimation algorithm for transferring distance information to qubits states, representation issues arise in the case of real-valued features. Indeed, the estimated distances are necessarily approximated. These factors collectively make the variant introduced by Afham et al., and detailed in Section 3.3.1, the most suitable candidate for experiments involving real-valued datasets. In addition, the parallel processing of the test instances, as suggested by Ma et al. [68], has not been considered here.

The chosen quantum binary classification model is the one elucidated in Section 3.4. Its structure is relatively simple and resembles that of the selected quantum k-NN. In particular, the possibility of constructing a pipeline with these quantum models was anticipated by Pastorello and Blanzieri in the pre-print version of their article about the quantum binary classifier [80].

7.1.2 Implementation

The implementation of the quantum pipeline has been carried out using Qiskit [3], the open-source software development kit (SDK) offered by IBM. Qiskit allows constructing quantum circuits in Python and executing them on either simulators or real quantum devices. Specifically, IBM provides various simulation backends, which allow getting measurement counts as in real quantum devices but also accessing the state vector of the circuit (namely, the amplitude of each state) at any execution stage.

Basically, using Qiskit, code has been developed to dynamically initialize and construct circuits for the considered quantum algorithms based on the dataset given as input. In addition, various execution modes have been implemented for both models:

- *classical*, which runs the corresponding classical algorithm without constructing a quantum circuit;
- *statevector*, which corresponds to an ideal execution with an infinite number of runs and, practically, processes the final state vector of the circuit;
- *simulation* (referred to as *local simulation* in the code), which samples from the final probability distribution of the circuit in order to provide counts;
- *online simulation*, which mirrors *simulation* but utilizes IBM's hardware;
- *quantum*, which employs real quantum devices.

In the classical mode, the cosine distance metric is used for the k-NN, as the selected quantum k-NN returns the k nearest neighbors based on the fidelity, i.e., the squared cosine similarity in the case of pure quantum states (see Equation 3.1). For the binary classifier, Equation (3.3) is employed. In addition, it is worth mentioning that no noise has been considered in any simulated mode (including *statevector*) and that the execution mode for each component of the pipeline does not have to be the same. Lastly, the implementation allows the retrieval of results from online executions at a later time.

The pseudocode for the quantum pipeline is outlined in Algorithm 20 and is valid for all execution modes except *classical*, as it does not exploit circuits. In practice, if the *classical* mode is chosen for a component, the corresponding block is substituted with the execution of the classical counterpart. In all cases, the first step consists in the unit-norm normalization of training and test data (Line 1), which is crucial for the amplitude encoding. Essentially, the features of each data instance are divided by the instance norm (if the norm is equal to zero, all attribute values are replaced with 0.000001 before the normalization). This normalization procedure is executed even for a pipeline consisting of only classical components. Furthermore, prior to the unit-norm normalization, another normalization procedure, whose details are provided in Section 7.2.3, has been executed in the experiments as part of the general experimental setup. In this way, the quantum k-NN requirement concerning the sign of the data features has been satisfied.

Concerning the non-classical modalities, the subsequent step consists in building the quantum k-NN circuit using the normalized data (Line 2). A representative circuit is provided in Figure 7.2a. Note that the vertical barriers have been added for clarity, and, in the actual implementation, a single barrier is placed between the measurement of the ancillary qubit and the measurement of the index register in order to sample from the right probability distributions. In particular, a quantum k-NN circuit can be split into three slices: registers initialization, SWAP test without measurement, and final measurements. In the first slice, the training data index is initialized concurrently with the encoding of the training and test features. In detail, the index (q_1-q_2) and the training features (q_3-q_4) registers are jointly initialized for simplicity, as they are entangled. Then, the SWAP test without

```
  Input: training data $D$, test instance $\mathbf{x}$, number of nearest neighbors $k$, execution modalities
         (not classical) for the two components exec_mods
  Result: class label $label \in \{-1, 1\}$
1 $D$, $\mathbf{x} \leftarrow normalization(D,\ \mathbf{x})$;
  /* Quantum k-NN */
2 $circ_{qknn} \leftarrow buildQKNNCircuit(D,\ \mathbf{x},\ \mathbf{exec\_mods}[0])$;       // See Figure 7.2a
3 $res_{qknn} \leftarrow execute(circ_{qknn},\ \mathbf{exec\_mods}[0])$;
4 $k\_nn \leftarrow getKNearestNeighbors(D,\ k,\ res_{qknn},\ \mathbf{exec\_mods}[0])$;
  /* Quantum binary classifier */
5 $circ_{qbc} \leftarrow buildQBCCircuit(k\_nn,\ \mathbf{x},\ \mathbf{exec\_mods}[1])$;       // See Figure 7.2b
6 $res_{qbc} \leftarrow execute(circ_{qbc},\ \mathbf{exec\_mods}[1])$;
7 $label \leftarrow getLabel(res_{qbc},\ \mathbf{exec\_mods}[1])$;
8 return $label$;
```

Algorithm 20: Quantum pipeline (taken from [128]).

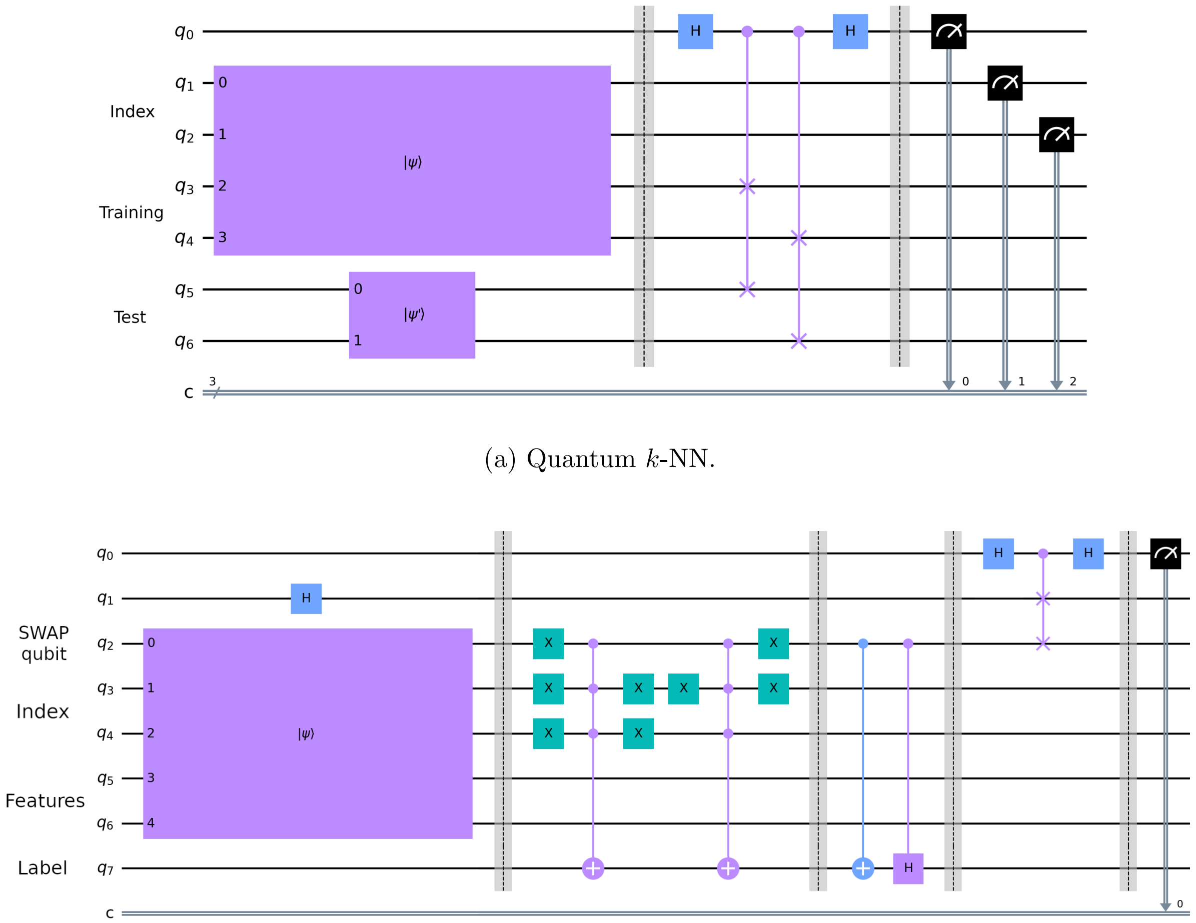
measurement is carried out by using two Hadamard gates and a number of controlled-SWAP gates that increases linearly with the number of feature qubits. Specifically, the SWAP gates operate on the training (q_3-q_4) and test features (q_5-q_6) registers. Lastly, the state of the first ancillary qubit (q_0) and the state of the index register are measured (the measurements are not present in the case of *statevector*). Concerning the required number of qubits ($qubits_{qknn}$), which depends on the dataset, it is determined by

$$qubits_{qknn} = 1 + qubits_{qknn\_index} + 2 * qubits_{features} \ , \tag{7.1}$$

with the value 1 corresponding to the SWAP test measurement qubit (q_0), $qubits_{qknn\_index}$ denoting the number of index qubits (q_3-q_4), and $qubits_{features}$ representing the number of feature qubits (either q_3-q_4 or q_5-q_6).

After constructing the circuit, the quantum k-NN is run based on the specified mode (Line 3). If the circuit involves measurements, it is executed multiple times to achieve the desired number of counts (controlled by the parameter *simulation_shots*, as outlined in Section 7.2.3). Then, the output of the execution (res_{qknn}), corresponding to either a state vector or state counts indicating the frequency of each observed state, is processed to identify the k nearest neighbors (Line 4). Specifically, the amplitudes/counts are utilized to estimate the $P(i|0) - P(i|1)$ quantity, where i denotes a training data index and 0/1 corresponds to the state of q_0 (refer to Equation 3.2). Subsequently, the training data is rearranged based on this quantity, which is proportional to the similarity to the test instance, allowing the retrieval of the nearest neighbors.

The subsequent step consists in building the quantum binary classifier circuit according to the selected k nearest neighbors and the normalized test instance (Line 5). A representative circuit is provided in Figure 7.2b, where the vertical barriers have been added for clarity. The circuit can be split into five main slices: registers initialization, configuration of training data labels, set up of the test label, SWAP test without measurement, and final measurement. Regarding the initialization, the first qubit undergoing the swap (q_1) is prepared in the uniform superposition of 0 and 1 ($|+\rangle$). Conversely, the second one (q_2) is entangled with the register (q_3-q_4) corresponding to the training data index and the register (q_5-q_6) encoding both the training and test features in superposition. Therefore, for simplicity, it is jointly initialized with all of them (from q_3 to q_6). Concerning the qubit encoding the label (q_7), it assumes only specific values (0, 1, and the uniform superposition of them). Hence, it is configured separately (second and third slices) to simplify the joint initialization. In more detail, the training data labels (second slice) are encoded in the last qubit of the circuit by means of NOT gates (represented by Xs in the image), which allow selecting the correct index register states, and multi-controlled NOT gates, which encode the given values. Actually, this procedure is necessary only for one label value, i.e., -1, as it is mapped to the state 1. Additionally, the NOT gates applied to the second SWAP qubit (q_2) before and after this step are required in order to operate on the states linked to the training data. Conversely, for the test instance (third slice), the label qubit is initialized



(a) Quantum k-NN.

(b) Quantum binary classifier.

Figure 7.2: Quantum pipeline circuits example. The first circuit (a) corresponds to the quantum k-NN, the second one (b) to the quantum binary classifier. In the case of the *statevector* modality, the final measurements are not present (taken from [128]).

in the $|-\rangle$ state using a controlled NOT gate and a controlled Hadamard gate. Subsequently, a SWAP test without measurement is executed (fourth slice), followed by the measurement of the first qubit (q_0) (last slice). In this case, as well, no final measurement is present for the *statevector* modality. The number of required qubits ($qubits_{qbc}$) is given by

$$qubits_{qbc} = 3 + qubits_{qbc\_index} + qubits_{features} + 1\,. \tag{7.2}$$

Specifically, the value 3 represents the qubits required by the SWAP test (q_0-q_1-q_2), $qubits_{qbc\_index}$ denotes the number of index qubits (q_3-q_4), $qubits_{features}$ represents the number of feature qubits (q_5-q_6), and 1 corresponds to the label qubit (q_7).

After building the circuit, the quantum binary classifier is executed (Line 6), with the observations made for the quantum k-NN holding also for the classifier. Ultimately, the output of the execution is processed (Line 7) to predict the label of the test instance. Specifically, the amplitudes/counts allow estimating the probability $P(1)$ of getting 1 by measuring the qubit q_0. This probability value enables the prediction of the class label (-1 if $P(1) > 0.25$, 1 otherwise, as per Equation 3.7), which is returned as the output of the pipeline (Line 8).

7.1.3 Complexity Observations

According to Afham et al. [2], the gate complexity of their quantum k-NN model is $O(\log_2 d)$, where d denotes the number of data features. Nevertheless, that complexity is not defined in terms of elementary gates, and the initialization of the registers is not taken into account, as they assume the existence of an initialization quantum oracle. Conversely, Pastorello and Blanzieri [79] define the time complexity of their quantum binary classifier as $O(\epsilon^{-2}\,\log_2(Nd))$, where ϵ is the wanted upper bound on the prediction error, and N is the size of the training set. However, they assume the existence of a QRAM from which the input state to the SWAP test (fourth and fifth slices of the circuit in Figure 7.2b) can be retrieved.

Regarding the implementation of the pipeline presented here, it is possible to make several observations for the execution modalities other than *classical*. Firstly, the unit-norm normalization step has a complexity of $O(Nd)$, as it requires scanning all data features. Building the quantum k-NN circuit has a complexity of $O(2^{\lceil\log_2 N\rceil+\lceil\log_2 d\rceil} + \lceil\log_2 d\rceil)$, where the first term accounts for the joint index-training initialization and the second term for the SWAP test. Conversely, executing the circuit has a complexity that depends on both the execution mode and its implementation inside Qiskit. For non-*statevector* executions, it is also determined by the number of measurements. Concerning the number of gates, it is essential to notice that the initialization of registers is a costly operation, as generating an arbitrary target state requires finding the right sequence of elementary gates. Additionally, the number of controlled-SWAP gates, which are not elementary gates, increases with the number of data features. The final step corresponds to the nearest neighbors extraction, which requires the processing of the execution output. In particular, processing the final state vector of the circuit has a complexity of $O(2^{1+\lceil\log_2 N\rceil+2\lceil\log_2 d\rceil})$, while processing the state counts has a complexity of $O(2^{1+\lceil\log_2 N\rceil})$. Regardless of the execution modality, after processing the output, the k nearest neighbors are identified by sorting the index values, which has a complexity of $O(N\log_2 N)$.

Focusing on the second part of the pipeline, the complexity of the quantum binary classifier depends on the number of nearest neighbors k. Specifically, building the classifier circuit has a complexity of $O(2^{1+\lceil\log_2 k\rceil+\lceil\log_2 d\rceil} + k\,2\lceil\log_2 k\rceil)$, where the second term is a worst-case estimate for setting up the training labels. Concerning the execution of the circuit and the number of gates, the considerations made for the quantum k-NN apply also to the classifier. Indeed, although the classifier circuit includes only one controlled-SWAP gate, there could be several (up to a maximum of k) multi-controlled CNOT gates, which significantly affect the performance as well. Lastly, the processing of the execution output has a complexity of $O(2^{4+\lceil\log_2 k\rceil+\lceil\log_2 d\rceil})$ for *statevector* executions, $O(1)$ in the other cases. The prediction of the final label also has a constant complexity.

To provide an idea of the runtimes, for a scenario where $N = 168$, $d = 12$, and $k = 9$, the execution time on the machine employed in the experiments (whose specifications are outlined at the beginning of Section 7.2) is in the order of 2-3 seconds. This applies to both *statevector* and *simulation*, with both components executed with the same modality (actually, *simulation* is slightly more time-consuming). In the considered scenario, the size of the quantum circuit is 17 qubits for the quantum k-NN and 12 qubits for the quantum binary classifier.

7.2 Empirical Evaluation

This section presents the quantum and classical algorithms considered, the selection and preparation of datasets, the experiments setup, and the obtained results. Specifically, the experiments have been conducted on a shared machine equipped with an Intel Xeon Gold 6238R processor operating at 2.20 GHz and 125 GB of RAM.

7.2.1 Methods

The quantum pipeline described in Section 7.1 has been evaluated with various combinations of execution modalities, as outlined in Table 7.1a. Specifically, the objectives of these experiments were to assess the classical pipeline's performance, confirm the equivalence between *classical* and *statevector* modalities (the latter representing an ideal execution), and examine the impact of simulation on performance. Actually, the original plan included also quantum executions. However, the maximum number of qubits available for a free account was five at the time of running the experiments (due to the retirement of the largest free machine). Consequently, the reliability of these results strictly depends on that of Qiskit's simulator.

The quantum binary classifier alone has been tested under the modalities outlined in Table 7.1b. Specifically, the *classical* mode has not been considered due to its effective equivalence with *statevector*.

Table 7.1: Quantum pipeline modalities (a), quantum binary classifier modalities (b), and baseline methods (c) considered (taken from [128]).

(a)

Quantum pipeline
classical - classical
statevector - classical
classical - statevector
statevector - statevector
simulation - classical
classical - simulation
simulation - simulation

(b)

Quantum classifier
statevector
simulation

(c)

Baseline methods
random forest
SVM
k-NN
k-NN + classifier
k-NN + SVM

These experiments aimed to gather data to validate any potential enhancement in the performance of the model with the introduction of a quantum locality technique like the quantum k-NN.

Lastly, the performance of the pipeline has been compared with that of the classical baseline methods detailed in Table 7.1c. The implementation provided by scikit-learn [85] has been used for many of them, while others have been implemented using scikit-learn procedures. Specifically, the default scikit-learn parameters have been utilized for the random forest, with the number of trees being equal to 100. Instead, for the SVM, two kernels have been evaluated, namely, Gaussian and linear kernels. Two distance metrics have also been tested for the k-NN, i.e., cosine and Euclidean distances. Eventually, two pipelines have been taken into account. The *k-NN + classifier* is the classical analogue of the quantum pipeline, with the classifier being the binary classifier based on the cosine similarity described by Equation (3.3). Conversely, the *k-NN + SVM* is a pipeline that has been considered in order to assess the benefits of using a more complex model (such as the SVM) instead of the binary classifier. These pipelines have been evaluated with both k-NN distance metrics (cosine and Euclidean) and both SVM kernels (Gaussian and linear). In conclusion, it is worth highlighting that the difference between the *k-NN + classifier* with cosine distance metric and the *classical - classical* execution of the quantum pipeline lies in the absence of the unit-norm normalization of the input data (for the former).

7.2.2 Datasets

All datasets employed in these experiments come from the UCI Machine Learning Repository [29]. Specifically, the dataset selection has been performed according to the following criteria: the associated task is classification (all the models taken into account are classifiers); the features are numerical and, preferably, real-valued (the *02_transfusion* dataset consists solely of integer attributes but is labeled as real-valued on the UCI website); the number of attributes is less than or equal to 16; the number of instances is not too big, i.e., less than one thousand.

The rationale behind the filter on the attribute type is the amplitude encoding utilized by the quantum models taken into account. Specifically, the data must be numerical, and integer values can be accepted thanks to the unit-norm normalization. Concerning the limitation on the features number, it derives from the initial plan of running experiments on a real quantum device. In particular, the free machine that would have later been retired had 15 qubits. Therefore, if the number of qubits needed for the feature encoding ($qubits_{features}$) had surpassed four (i.e., more than 16 attributes), the training instances embeddable in the quantum k-NN circuit would have been fallen below 17 ($qubits_{qknn\_index} \leq 4$), an insufficient quantity (refer to Equation 7.1). Ultimately, the constraint on the number of instances was designed to permit the encoding of the dataset in the quantum circuit without having to perform a drastic subsampling.

Ten datasets meeting the specified criteria were found (some have been discarded because of missing values or unclear structure). Nevertheless, a majority of these datasets featured more than two classes, while Pastorello and Blanzieri's classifier (but also the SVM) operates on binary labels. Consequently, only the two most represented classes have been kept and mapped to $\{-1, 1\}$ (in case of a tie, the first two classes have been selected); all the other instances have been discarded. Additionally, if the size

Table 7.2: Datasets properties (the dataset names are links that lead to the corresponding UCI pages). Note: "qb." stands for qubits (taken from [128]).

Name	**Original size**	**Original classes #**	**Features #**	**Size (15 qb.)**	**Size (32 qb.)**
01_iris_setosa_versicolor	150	3	4	100	-
01_iris_setosa_virginica				100	-
01_iris_versicolor_virginica				100	-
02_transfusion[122]	748	2	4	748	-
03_vertebral_column_2C	310	2	6	310	-
04_seeds_1_2	210	3	7	140	-
05_ecoli_cp_im	336	8	7	220	-
06_glasses_1_2	214	6	9	80	146
07_breast_tissue_adi_fadmasgla	106	6	9	71	-
08_breast_cancer[83]	116	2	9	80	116
09_accent_recognition_uk_us	329	6	12	80	210
10_leaf_11_9[108]	340	30	14	30	-

of the resulting dataset was still surpassing the number of instances that can be encoded in the circuit of the quantum k-NN with 15 qubits, a random subsampling preserving the ratio between classes has been performed. Regarding this last operation, the reduction of the dataset size due to the split into training and test sets (accurately detailed in the subsequent section) has also been considered. The resulting datasets and their properties are presented in Table 7.2.

Several aspects of the data presented in the table are worth to be mentioned: the dataset names include a suffix denoting the names of the chosen classes if the original number of classes exceeded two; all possible combinations of classes have been considered for the Iris dataset (01), resulting in a total of 12 datasets; the suggested merging of three classes has been executed for *07_breast_tissue_adi_fadmasgla*. Concerning the datasets sizes used in the experiments, they are detailed in the last two columns: the first one (*Size, 15 qb.*) reports the sizes resulting from the entire process presented in the preceding paragraph, while the second one (*Size, 32 qb.*) displays the sizes without the final subsampling operation (only for datasets for which it was necessary). In detail, the three datasets requiring subsampling (namely, *06_glasses_1_2*, *08_breast_cancer*, and *09_accent_recognition_uk_us*) have been exploited to study the impact of a larger training set, and the value 32 corresponds to the maximum number of qubits for an online simulation (way less qubits are needed in these experiments).

It is also worth highlighting that the two erroneous instances documented on the Iris UCI webpage have been corrected before executing any other operation. The datasets employed in these experiments are publicly available [125].

7.2.3 Experimental Setup

All methods listed in Section 7.2.1 have been executed on all datasets present in Table 7.2 (for both 15 and 32 qubits limits), with the exception of the quantum binary classifier, which has not been applied to the *02_transfusion* dataset due to the need for an additional qubit (refer to Equation 7.2). The κ-fold cross-validation has been used as model evaluation technique. In practice, the dataset is divided into κ subsets, also known as folds. Then, $\kappa - 1$ folds constitute the training set, and the remaining one becomes the test set. This last step is repeated κ times, ensuring that each fold is employed as the test set exactly once. Specifically, the stratified κ-fold cross-validation, which maintains the class ratio inside the folds as close as possible to the original one, has been used here. Lastly, it is important to mention that the same seed has been used for the folds generation in all experiments, ensuring that all methods have processed identical folds.

The experiments parameters are detailed in Table 7.3. Specifically, *k_folds* denotes the number of folds, which has been set to 5, a commonly used value in machine learning. Conversely, k represents the selected number of nearest neighbors, a crucial parameter for the quantum pipelines, as well as for the classical k-NN and the baseline pipelines (*k-NN + classifier* and *k-NN + SVM*). Hence, a

Table 7.3: Parameters of the experiments (taken from [128]).

Parameter	Value(s)
k_folds	5
k	3, 5, 7, 9
simulation_shots	1024
simulation_runs	5

range of odd (small) values in arithmetic progression has been tested. Regarding *simulation_shots*, it denotes the number of measurements (and thus circuit executions) carried out in the *simulation* mode, and the same value (1024, the default provided by Qiskit) has been used for both the quantum k-NN and the quantum binary classifier. Lastly, *simulation_runs* denotes the number of runs performed for experiments involving a quantum model (k-NN/classifier) in the *simulation* mode or the random forest. Specifically, the chosen value (5) constitutes a trade-off between the possibility of assessing the statistical significance of results and the computational resources needed. Moreover, no seed has been set for these stochastic methods.

In every cross-validation iteration, a min-max data normalization method has been executed before running the model or pipeline. Specifically, each training set attribute has been rescaled to the [0, 1] interval by subtracting the minimum value and dividing by the range. The attributes characterised by a zero range (constant attributes) have been set to zero. As customary, the minimum and range values of the training set have been used for the normalization of the test instances. In the case of a test instance attribute exceeding the interval boundaries after the normalization, a clipping operation has been executed. Moreover, after the feature normalization, a unit-norm normalization procedure has been applied to the input data for the quantum models (including the standalone quantum classifier), as outlined in Section 7.1.2. It is worth observing that the min-max normalization has also ensured consistency in attribute signs, avoiding sign-related issues for the quantum k-NN.

7.2.4 Results

The results are presented via scatter plots based on the accuracy metric, with the accuracy for a fold being defined as

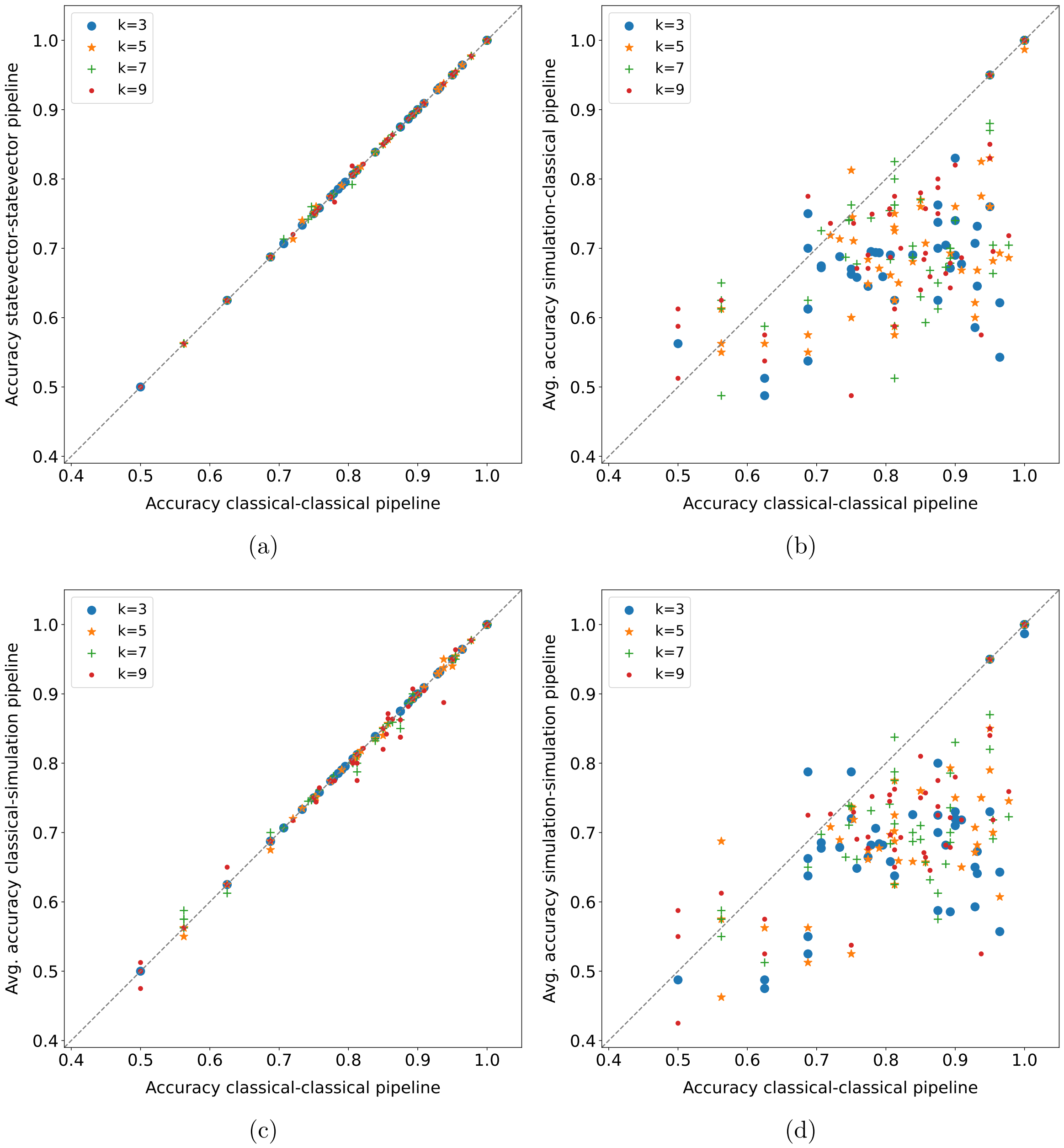
$$accuracy = \frac{number\ of\ correctly\ classified\ instances\ in\ the\ fold}{total\ number\ of\ instances\ in\ the\ fold} \tag{7.3}$$

(this definition is a more specific version of the one provided in Equation 6.1). For multiple runs, the fold average accuracy is shown. Additionally, the statistical significance of the results has been assessed using the Wilcoxon signed-rank test [116], as the data are paired.

Specifically, the first analysis focuses on the various execution modalities of both the quantum pipeline and the quantum binary classifier. Subsequently, a comparison of the two quantum models is presented, and the impact of the dataset size on their performance is analysed. Lastly, an evaluation of the distance metrics considered for the baseline methods is provided, followed by a comparison between the quantum pipeline and the baseline methods.

7.2.4.1 Execution Modalities Comparison (Quantum Pipeline)

Figure 7.3 shows several comparisons between execution modalities for the quantum pipeline on the *15 qubits* datasets. Each point in these plots corresponds to the accuracy achieved in a fold (or its average across runs). In detail, as expected, the *statevector - statevector* modality, which represents an ideal execution, is found to be equivalent to the *classical - classical* modality (Figure 7.3a). The few deviations from the main diagonal stem from two factors: the two modalities utilize different policies for selecting the nearest neighbors in the case of a distance tie; the *02_transfusion* dataset includes data points with the same features and different class labels. Despite this, the difference between the two modalities is not statistically significant according to the Wilcoxon signed-rank test (Table 7.4). It is important to remember that the advantage of the considered quantum models/pipelines over their classical counterparts does not lie in the accuracy but in the execution time. Instead, the plots showing the comparison between the *classical - classical* mode and the *statevector - classical / classical*



(a)

(b)

(c)

(d)

Figure 7.3: Execution modalities comparison on *15 qubits* datasets for the quantum pipeline. Each point represents the accuracy obtained in a fold (or its average across runs) (taken from [128]).

Table 7.4: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the fold accuracy distributions shown in Figure 7.3. The values reported in the table are the p-values obtained (taken from [128]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 7.3a	1.000	0.276	1.000	0.655
Figure 7.3b	8.80E-08	1.18E-07	2.46E-06	4.01E-06
Figure 7.3c	0.414	0.052	0.486	0.092
Figure 7.3d	1.04E-07	2.03E-07	2.27E-07	4.60E-07

Table 7.5: Average *usage on dataset* of the second model for the pipelines including the classical (or statevector) k-NN with cosine distance (a) and Euclidean distance (b). The *usage on dataset* is 1 when the second model is always employed (taken from [128]).

(a) Cosine.

k	15 qubits	32 qubits
3	0.245 ± 0.204	0.375 ± 0.128
5	0.357 ± 0.290	0.590 ± 0.160
7	0.406 ± 0.323	0.690 ± 0.158
9	0.461 ± 0.356	0.758 ± 0.137

(b) Euclidean.

k	15 qubits	32 qubits
3	0.218 ± 0.229	0.402 ± 0.133
5	0.298 ± 0.305	0.617 ± 0.187
7	0.346 ± 0.346	0.686 ± 0.188
9	0.381 ± 0.371	0.755 ± 0.166

- *statevector* have not been reported here as they are identical or nearly identical (without deviant points) to Figure 7.3a.

While the quantum pipeline is equivalent in accuracy to the classical one in an ideal scenario, this equivalence does not hold when the pipeline is simulated (or executed on a real quantum device). Figures 7.3b and 7.3d reveal that simulating the quantum k-NN with 1024 shots (namely, measurements) has a negative impact on the pipeline's performance, regardless of the chosen k value. In fact, the majority of points are positioned below the main diagonal and the difference is statistically significant for all k values, as indicated in Table 7.4. In practical terms, the considered quantum k-NN is highly sensitive to small fluctuations in the estimated probability values, and a higher number of repetitions should be used to improve the results. Conversely, simulating only the quantum binary classifier with 1024 shots (Figure 7.3c) seems to not significantly affect the pipeline performance, and the difference is not statistically significant. However, for the datasets employed here, the model following the classical (or *statevector*) k-NN is rarely used in practice. In most cases, the data samples selected by the classical k-NN all belong to the same class, making it challenging to draw definitive conclusions about the subsequent classifier. Table 7.5a reports the average *usage on dataset* of the second model in the pipeline for the cosine distance metric and each k value.

7.2.4.2 Execution Modalities Comparison (Quantum Binary Classifier)

Figure 7.4 shows the comparison between *statevector* and *simulation* modes for the quantum binary classifier on the *15 qubits* datasets. The *classical* modality has not been considered because of its

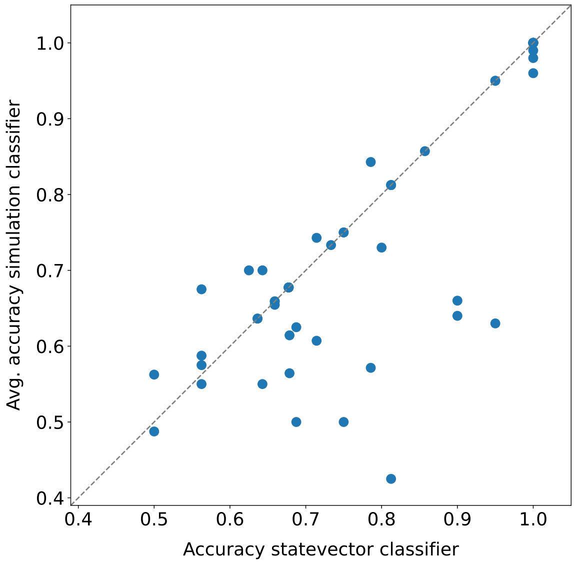
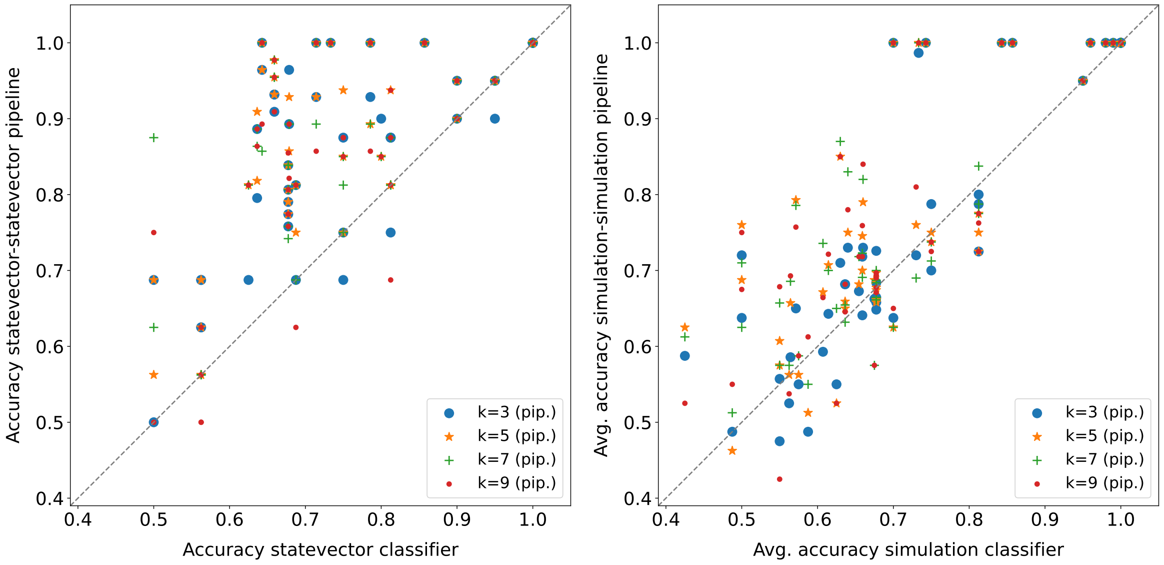


Figure 7.4: Execution modalities comparison on *15 qubits* datasets for the quantum binary classifier. The *02_transfusion* dataset is not present, and each point represents the accuracy obtained in a fold (or its average across runs). The p-value obtained by applying the Wilcoxon signed-rank test ($\alpha = 0.05$) to the fold accuracy distributions is 0.016 (taken from [128]).

equivalence with *statevector*. Moreover, the *02_transfusion* dataset has been excluded as an extra qubit would have been necessary (as explained in Section 7.2.3). As in the previous plots, each dot corresponds to the accuracy achieved in a fold (or its average across runs). Basically, the quantum binary classifier as well is influenced by probability fluctuations and by the shots number. In fact, the accuracy achieved by *simulation* is generally inferior to that of *statevector*, and this difference is statistically significant according to the Wilcoxon signed-rank test (p-value= 0.016).

7.2.4.3 Quantum Pipeline - Quantum Binary Classifier Comparison

Figure 7.5 shows the comparison between the quantum pipeline and the quantum binary classifier on the *15 qubits* datasets. The scatter plots are structured as those in Figure 7.3. However, the *02_transfusion* dataset is absent (to enable the comparison), and the k values reported in the legend refer exclusively to the pipeline. As illustrated in Figure 7.5a, the *statevector - statevector* pipeline outperforms the *statevector* classifier independently of the k value, and the difference is statistically significant (Table 7.6). Although there are folds where the classifier alone achieves a higher accuracy, they represent a minority. This demonstrates the effectiveness of employing a quantum locality technique like the quantum k-NN as a preliminary step of a quantum classifier, and more generally, the value of locality. Conversely, the pipeline's superiority is less pronounced when considering the simulated versions of the models (Figure 7.5b). The quantum pipeline, because of the considered quantum k-NN, is significantly more affected by probability fluctuations with respect to the classifier alone. Nonetheless, it demonstrates better overall performance, regardless of the k value (here, the difference is statistically significant as well). Lastly, examining the performance of the different k values in these two plots, no dominant one emerges. In fact, the optimal k value tends to be dataset-dependent. Determining the optimal value for a given dataset would require evaluating the performance of multiple k values employing part of the dataset as a validation set, a task beyond the scope of this paper.



(a) (b)

Figure 7.5: Quantum pipeline - quantum binary classifier comparison on common *15 qubits* datasets. Each point represents the accuracy obtained in a fold (or its average across runs); the k values refer only to the pipeline (taken from [128]).

Table 7.6: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the fold accuracy distributions shown in Figure 7.5. The values reported in the table are the p-values obtained (taken from [128]).

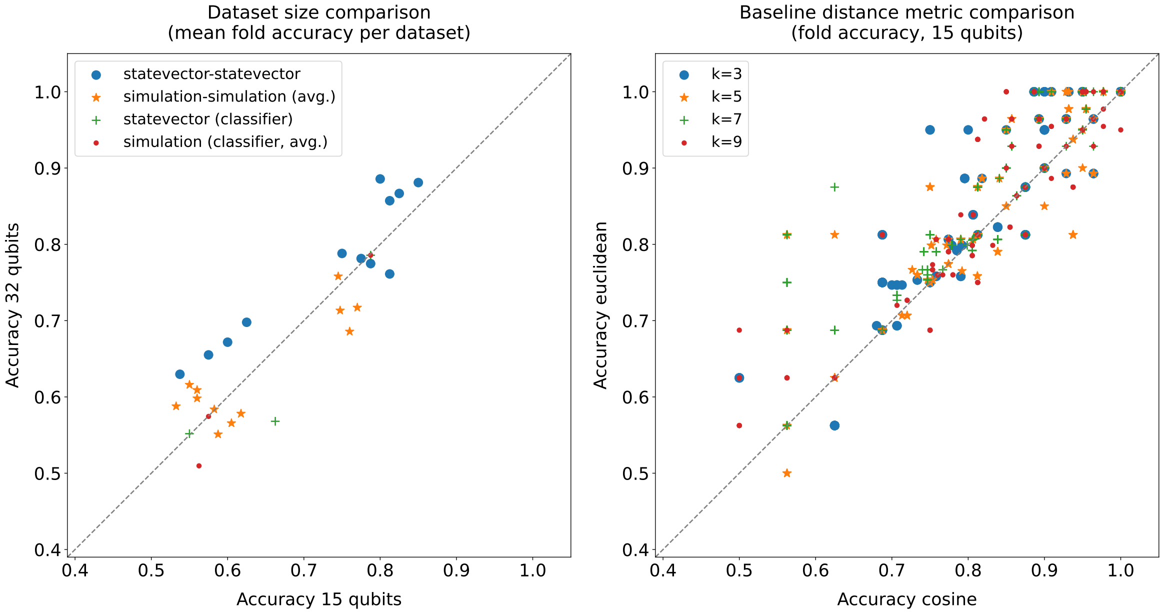
	k=3	**k=5**	**k=7**	**k=9**
Figure 7.5a	5.39E-07	7.88E-07	2.55E-06	2.98E-06
Figure 7.5b	0.042	0.001	3.35E-04	5.84E-04

7.2.4.4 Dataset Sizes Comparison

Figure 7.6a shows the impact of the dataset size on the performance of the quantum pipeline and quantum binary classifier. Specifically, for this chart, only the three datasets in Table 7.2 with both *15 qubits* and *32 qubits* sizes have been taken into account. Furthermore, as a fold-by-fold comparison would not be meaningful in this case, each point in the plot corresponds to the mean fold accuracy on a dataset (or its average across runs). Lastly, the number of points for the pipelines is four times higher compared to the classifier alone, as all k values are taken into account. In practical terms, a larger dataset tends to positively influence the performance of the pipeline in the ideal case (*statevector - statevector*) and has an overall neutral effect in the simulated case (the occurrences of improvement and worsening are balanced). Conversely, the quantum classifier performance tends to degrade in both cases. This demonstrates the ability of the pipeline to leverage a larger number of samples. Although the difference is statistically significant only for the *statevector - statevector* pipeline, as reported in Table 7.7a, the low number of points should also be considered.

7.2.4.5 Distance Metrics Comparison

Figure 7.6b illustrates the comparison between the cosine and Euclidean distances on the *15 qubits* datasets, considering all the baseline methods relying on the k-NN algorithm, namely, the k-NN, the



(a) (b)

Figure 7.6: Dataset sizes (a) and distance metrics (b) comparisons. In the dataset sizes comparison (a), each point represents the mean fold accuracy obtained on a dataset (or its average across runs); the pipeline comparisons include all k values. In the distance metrics comparison (b), the results obtained by the k-NN-based baseline methods (*k-NN*, *k-NN + classifier*, *k-NN + SVM Gaussian*, *k-NN + SVM linear*) on the *15 qubits* datasets are taken into account; each point represents the accuracy obtained in a fold (taken from [128]).

Table 7.7: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the mean fold accuracy distributions shown in Figure 7.6a (a). Same test applied to the fold accuracy distributions shown in Figure 7.6b (b) (taken from [128]).

(a) Dataset size.

	p-value
statevector - statevector	0.016
simulation - simulation	0.910
statevector	0.750
simulation	0.250

(b) Distance metric.

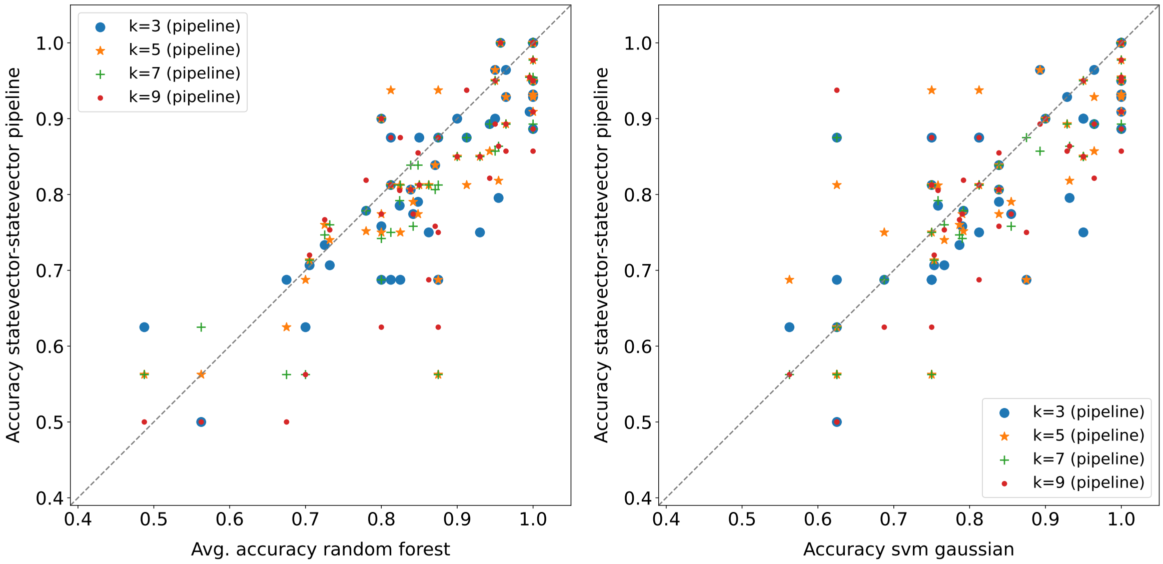
	p-value
k=3	9.85E-10
k=5	8.33E-08
k=7	7.65E-15
k=9	4.86E-08

k-NN + classifier, and the *k-NN + SVM* with both Gaussian and linear kernels. In this plot, each point corresponds to the accuracy achieved in a fold by one of these four methods. Essentially, for all k values, the Euclidean distance statistically outperforms the cosine distance, as reported in Table 7.7b. Hence, having a quantum k-NN version based on the Euclidean distance metric could be advantageous.

7.2.4.6 Quantum Pipeline - Baseline Methods Comparison

Figures 7.7 and 7.8 show some comparisons between the *statevector - statevector* pipeline and baseline methods on the *15 qubits* datasets. In particular, each point corresponds to the accuracy achieved in a fold (or its average across runs), and the k values reported in the legends of Figures 7.7a and 7.7b refer exclusively to the pipeline. Basically, the random forest consistently outperforms the quantum pipeline for all k values (Figure 7.7a), and the same trend is observed for the best-performing SVM, namely, the SVM with the Gaussian kernel (Figure 7.7b). The differences are statistically significant in both cases, as demonstrated by Table 7.8, with the only exception being the SVM - pipeline comparison with $k = 5$. Conversely, the SVM with the linear kernel (not displayed here) achieves only marginally better performance compared to the pipeline, and the difference is not statistically significant.

Concerning the k-NN, the version employing the cosine distance metric appears to be equivalent in accuracy to the *statevector - statevector* pipeline (Figure 7.8a), with few deviations analogous to those in Figure 7.3a. As reported in Table 7.9, the difference is not statistically significant. Hence, it is also on par with the *classical - classical* pipeline (refer to Section 7.2.4.1). This implies that, for the datasets taken into account, employing either a majority voting or the binary classifier based on the cosine similarity after extracting the k nearest neighbors with the cosine distance metric has exactly the same effect. In fact, the absence of unit-norm normalization in the baseline methods does not influence the cosine distance (or the cosine similarity), as it intrinsically normalizes the data given as input. However, the low effective usage of the second model in the pipeline must also be considered (Table 7.5a). Conversely, the k-NN employing the Euclidean distance statistically outperforms the



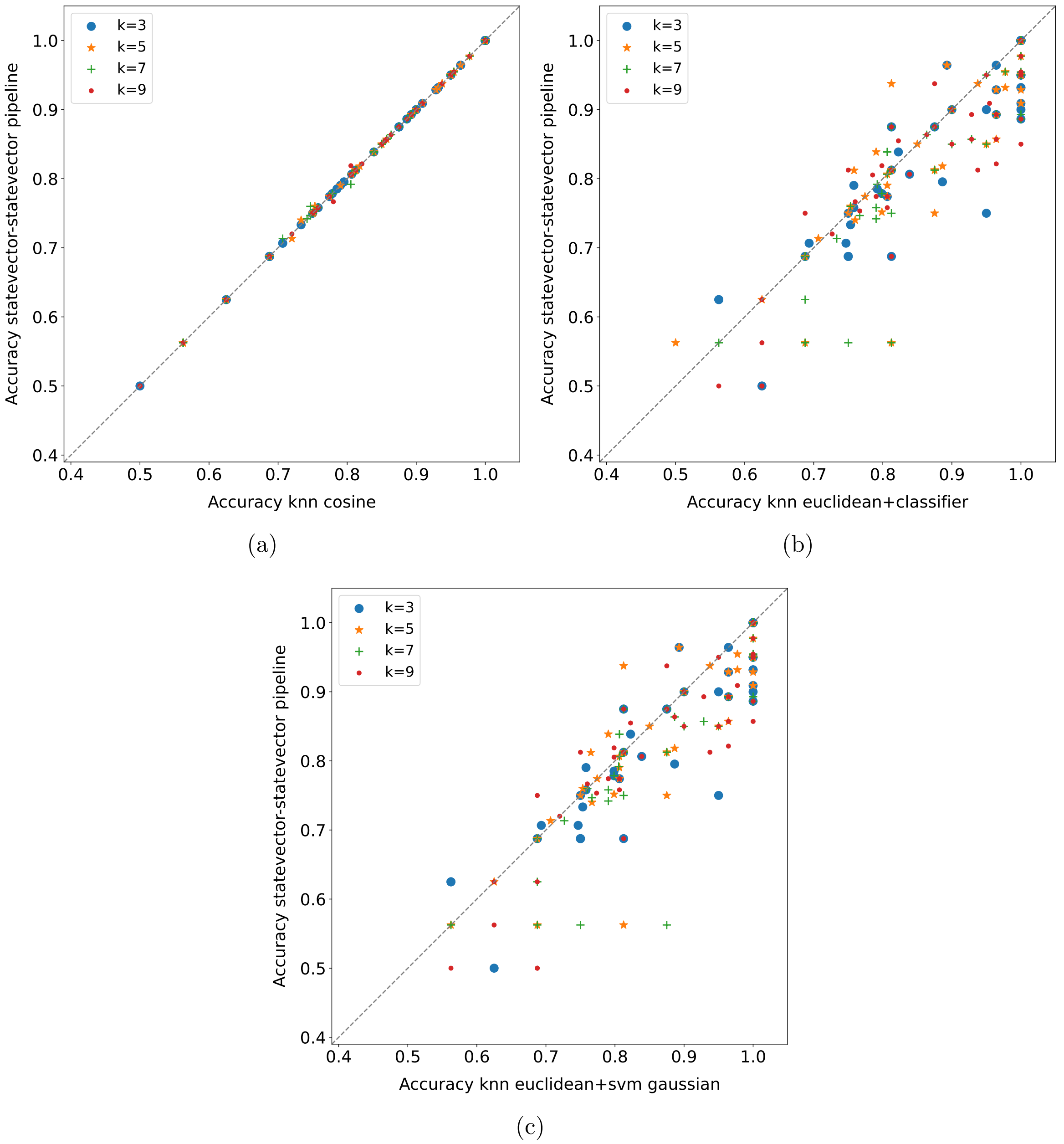
(a)

(b)

Figure 7.7: Quantum pipeline - baseline methods comparison on *15 qubits* datasets. The pipeline modality is *statevector - statevector*, each point represents the accuracy obtained in a fold (or its average across runs), and the k-values refer only to the pipeline (taken from [128]).

Table 7.8: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the fold accuracy distributions shown in Figure 7.7. The values reported in the table are the p-values obtained (taken from [128]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 7.7a	1.54E-04	0.001	1.24E-04	1.88E-04
Figure 7.7b	0.020	0.106	0.003	0.004



(a)

(b)

(c)

Figure 7.8: Quantum pipeline - (k-NN-based) baseline methods comparison on *15 qubits* datasets. Each point in these plots represents the accuracy obtained in a fold (taken from [128]).

Table 7.9: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the fold accuracy distributions shown in Figure 7.8. The values reported in the table are the p-values obtained (taken from [128]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 7.8a	1.000	0.276	1.000	0.655
Figure 7.8b	0.003	0.006	5.22E-05	0.001
Figure 7.8c	0.003	0.002	2.15E-05	4.36E-04

quantum pipeline across all k values, and the scatter plot mirrors that of Figure 7.8b (refer to Table 7.9 for the results of the statistical test). Essentially, when utilizing the same distance metric, the k-NN and the *k-NN + classifier* exhibit identical performance on the considered datasets. Consequently, the *statevector - statevector* pipeline obtains the same results as the *k-NN + classifier* with cosine distance metric (the scatter plot mirrors that of Figure 7.8a) and is overcome by the same model employing the Euclidean distance independently of the k value (Figure 7.8b). Actually, due to the previous observation on the unit-norm normalization, the *k-NN + classifier* with cosine distance and the *classical - classical* pipeline turn out to be exactly the same model. Lastly, turning attention to the baseline pipelines incorporating the SVM model, the best-performing among them is the *k-NN + SVM* with Euclidean distance and Gaussian kernel, which achieves statistically better results compared to the *statevector - statevector* pipeline regardless of the k value, as depicted in Figure 7.8c (the results of the statistical test are provided in Table 7.9). Concerning the other versions of this baseline pipeline, the *k-NN + SVM* with cosine distance and linear kernel is nearly equivalent in accuracy to the quantum pipeline (but still winning the comparison), while the others exhibit a clear superiority. Furthermore, in the case of the Euclidean distance, the *k-NN + SVM* with linear kernel and the *k-NN + classifier* turn out to be on par, whereas, in the case of the cosine distance, the former achieves slightly better results than the latter.

Since the effective usage of the model following the extraction of the nearest neighbors is quite low (the values for the Euclidean distance can be found in Table 7.5b), definitive conclusions cannot be drawn. Nonetheless, in the baseline pipelines, the SVM appears to achieve better results compared to the cosine similarity classifier, particularly with a Gaussian kernel. Therefore, trying to combine a quantum k-NN version with a quantum SVM might be beneficial. Lastly, it is worth observing that, with the same kernel, the classical SVM generally performs better than the corresponding pipeline with cosine distance and is either outperformed by or equivalent to (in accuracy) the corresponding pipeline with Euclidean distance (still, the low effective usage of the SVM in the pipeline must be considered).

7.3 Discussion

In this chapter, the introduction of a quantum locality technique as a preliminary step of a quantum machine learning model, to improve its performance and reduce its circuit size, has been proposed and tested. In particular, the pipeline that has been considered consists of a quantum k-NN and a quantum binary classifier. Information concerning the pipeline's implementation (in Python, using Qiskit) and its complexity have been provided. The results have shown that, in the ideal case, the quantum pipeline is equivalent in accuracy to its classical counterpart and is also able to leverage larger datasets. Furthermore, the quantum pipeline has consistently outperformed the quantum binary classifier in both ideal and simulated scenarios, demonstrating the power of locality in the quantum realm. Nevertheless, the considered quantum k-NN has turned out to be highly sensitive to probability fluctuations (the quantum binary classifier to a lesser degree), and baseline methods such as the random forest and the SVMs have outperformed the quantum pipeline even in the ideal case. However, it must be considered that the effective usage of the second model in the pipeline was notably low for the datasets taken into account.

Concerning the required number of qubits, the following relationship (derived from Equations 7.1 and 7.2) holds for the considered quantum models:

$$qubits_{qknn} \leq qubits_{qbc} \iff qubits_{features} \leq 3\,.$$

In practice, using the considered quantum k-NN as a preliminary step of the quantum binary classifier proves to be advantageous (not disadvantageous) in terms of number of qubits only if the number of attributes is less than 5 (less than 9). Nevertheless, it is important to remark that this relationship holds for the specific quantum models taken into account here, and that the introduction of the quantum k-NN has demonstrated to be advantageous in terms of performance.

In addition, it is worth observing that the unit-norm normalization required by the amplitude encoding of data, which is exploited also by the considered quantum models, is characterised by a significant information loss. Indeed, for example, two data samples with the same ratio between features but different norms are mapped to the same unit vector. This issue could be addressed by introducing additional ad hoc features, similarly to the approach presented in Chapter 8.

Possible future work includes the evaluation on more complex datasets of quantum pipelines involving more complex quantum machine learning models, such as the quantum SVM [89], as its classical counterpart has shown better performance in the pipeline compared to the binary classifier.

8 A Euclidean k-NN Algorithm

This chapter is a reworked version of the article “A quantum k-nearest neighbors algorithm based on the Euclidean distance estimation” [127], which was motivated by the absence in the literature of a quantum version of the k-NN algorithm utilizing the Euclidean distance as the distance metric. In practice, in this chapter, a quantum adaptation of the k-NN algorithm, wherein Euclidean distances are computed using a novel quantum encoding with low qubit demands and a simple quantum circuit, is introduced. Similarly to other quantum algorithms, an exponential speedup compared to classical calculations is achieved only if a QRAM is available. Specifically, the performance of the proposed quantum k-NN, implemented using Qiskit, have been assessed in terms of both classification accuracy and correctness of identified nearest neighbors (measured using the Jaccard index). The empirical evaluation on a real quantum device was impeded by the qubit requirements of the considered experiments (no sufficiently large machine was available at the time of running the experiments). Among other things, the results have shown the correctness of the formulation.

8.1 Method

In this section, the novel quantum k-NN algorithm employing the Euclidean distance metric is introduced. Furthermore, a concise discussion of the algorithm's complexity in comparison to its classical counterpart is provided.

8.1.1 Algorithm

In the proposed quantum k-NN algorithm, a quantity related to the squared Euclidean distance is calculated simultaneously for all training instances. This is achieved through a novel encoding and a straightforward quantum circuit that implements a SWAP-test-like procedure without using controlled-SWAP gates. Essentially, the algorithm leverages the quantum interference phenomenon and encodes these values related to the distances, subsequently estimated via measurements, into the quantum states amplitudes. It is worth underlining that the input vectors are not subjected to a unit-norm normalization procedure, which would lead to a significant information loss (as observed in Section 7.3). Moreover, the algorithm requires a low number of qubits and does not involve any oracle, which makes its implementation particularly advantageous. A more in-depth and formal description of this novel algorithm is provided in the following.

8.1.1.1 Data Preprocessing

Let us take into account a training set, denoted as $\mathcal{U} = \{\mathbf{u}_0, ..., \mathbf{u}_{N-1}\}$, consisting of real-valued data instances $\mathbf{u}_j \in \mathbb{R}^d$, and let $\mathcal{L} = \{l_0, ..., l_{N-1}\}$ represent the set of labels associated with these instances. Additionally, let $\mathbf{u}' \in \mathbb{R}^d$ be a test instance with unknown label.

In the preprocessing step of the algorithm, the features are centered and normalized into the range $\left[-\frac{1}{2\sqrt{d}}, \frac{1}{2\sqrt{d}}\right]$. This normalization ensures that the resulting vectors have a maximum norm of $\frac{1}{2}$ and a maximum (squared) Euclidean distance of 1.

8.1.1.2 Initial State and Encoding(s)

Let $\mathcal{V} = \{\mathbf{v}_0, ..., \mathbf{v}_{N-1}\}$ and $\mathbf{v}'$ denote the training set and the test instance after the preprocessing step. The quantum circuit is initialized in state

$$|\psi\rangle = |0\rangle \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle\,|\alpha\rangle + |1\rangle\,|\beta\rangle)\right), \tag{8.1}$$

where

$$|\alpha\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle \sum_{i=0}^{F-1} x_{ji} |i\rangle \,,$$

$$|\beta\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle \sum_{i=0}^{F-1} x'_{ji} |i\rangle \,.$$

Here, F denotes a positive integer value dependent on the selected encoding, and $\mathbf{x}_j = \{x_{ji}\}_{i=0,\dots,F-1}$ and $\mathbf{x}'_j = \{x'_{ji}\}_{i=0,\dots,F-1}$ correspond to the quantum encoded forms of the preprocessed training and test data, respectively. Consequently, the required number of qubits is $2 + \lceil \log_2 N \rceil + \lceil \log_2 F \rceil$. Specifically, this work introduces and evaluates two encodings, *extension* and *translation*, whose advantages are presented in the next sections. Let us examine their definitions. Concerning the *extension* encoding, $F = 2d + 3$ and

$$x_{ji} = \begin{cases} \frac{2}{\sqrt{3}} v_{ji} \\ \frac{2}{\sqrt{3}} v_{j(i-d)} \\ \frac{2}{\sqrt{3}} \|\mathbf{v}_j\| \\ 0 \\ \sqrt{1 - 4\|\mathbf{v}_j\|^2} \end{cases} \qquad x'_{ji} = \begin{cases} -\frac{2}{\sqrt{3}} v'_i & 0 \le i < d \\ -\frac{2}{\sqrt{3}} v'_{(i-d)} & d \le i < 2d \\ \frac{2}{\sqrt{3}} \|\mathbf{v}_j\| & i = 2d \\ \sqrt{1 - \frac{4}{3}(2\|\mathbf{v}'\|^2 + \|\mathbf{v}_j\|^2)} & i = 2d+1 \\ 0 & i = 2d+2, \end{cases}$$

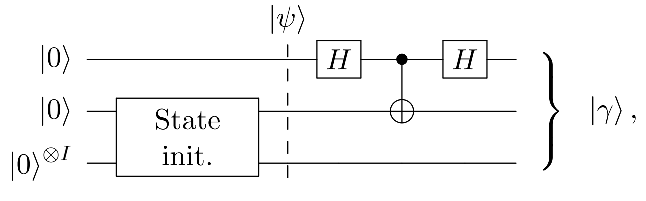
where v_{ji} is the i-th feature of the j-th preprocessed training instance, and v'_i is the i-th feature of the preprocessed test instance. Conversely, for the *translation* encoding, $F = 2d + 4$ and

$$x_{ji} = \begin{cases} v_{ji} \\ v_{j(i-d)} \\ \|\mathbf{v}_j\| \\ \frac{1}{2} \\ 0 \\ \sqrt{\frac{3}{4} - 3\|\mathbf{v}_j\|^2} \end{cases} \qquad x'_{ji} = \begin{cases} -v'_i & 0 \le i < d \\ -v'_{(i-d)} & d \le i < 2d \\ \|\mathbf{v}_j\| & i = 2d \\ -\frac{1}{2} & i = 2d+1 \\ \sqrt{\frac{3}{4} - (2\|\mathbf{v}'\|^2 + \|\mathbf{v}_j\|^2)} & i = 2d+2 \\ 0 & i = 2d+3. \end{cases}$$

Therefore, both encodings require the same number of qubits. Lastly, it is worth observing that, in both cases, $\mathbf{x}_j$ (and, consequently, $|\alpha\rangle$) is independent of the preprocessed test instance $\mathbf{v}'$, while $\mathbf{x}'_j$ (and, consequently, $|\beta\rangle$) is contingent on the preprocessed training set $\mathcal{V}$.

8.1.1.3 Bell-H Operation and Final State

Once the initial state has been prepared, an operation denoted here as *Bell-H* is executed. Specifically, the *Bell-H* is a SWAP-test-like procedure where the initially superimposed states of interest ($|\alpha\rangle$ and $|\beta\rangle$) interfere via a CNOT gate. The corresponding quantum circuit, encompassing also the preparation of the initial state (ending at the dashed vertical line), is



with $I = \lceil \log_2 N \rceil + \lceil \log_2 F \rceil$. Essentially, the *Bell-H* circuit includes a first Hadamard gate applied to the first qubit, a CNOT gate having the first qubit as the control and the second qubit as the target, and a second Hadamard gate applied to the first qubit. Therefore, the distinction with respect to a standard Bell circuit, commonly employed for generating Bell states, consists in the presence of the second Hadamard gate. Compared to the standard SWAP test, an advantageous aspect is the constant number of elementary gates used (three), regardless of the size of the involved states, while a drawback is the increased complexity of the input state preparation, particularly without a QRAM.

The resulting state, after the *Bell-H* operation, is

$$|\gamma\rangle = \frac{1}{2}\Bigg(|0\rangle \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle|\alpha\rangle + |0\rangle|\beta\rangle + |1\rangle|\beta\rangle + |1\rangle|\alpha\rangle)\right) + \\ |1\rangle \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle|\alpha\rangle - |0\rangle|\beta\rangle + |1\rangle|\beta\rangle - |1\rangle|\alpha\rangle)\right)\Bigg), \quad (8.2)$$

and the probability of obtaining 1 by measuring the first qubit is equal to

$$\begin{aligned} P(1) &= \| |1\rangle\langle 1|\gamma\rangle \|^2 = \\ &= \left\| \frac{1}{2}|1\rangle \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle|\alpha\rangle - |0\rangle|\beta\rangle + |1\rangle|\beta\rangle - |1\rangle|\alpha\rangle)\right)\right\|^2 = \\ &= \frac{1}{8}((\langle 0|\langle\alpha| - \langle 0|\langle\beta| + \langle 1|\langle\beta| - \langle 1|\langle\alpha|) \times (|0\rangle|\alpha\rangle - |0\rangle|\beta\rangle + |1\rangle|\beta\rangle - |1\rangle|\alpha\rangle) = \\ &= \frac{1}{8}(1 - \langle\alpha|\beta\rangle - \langle\beta|\alpha\rangle + 1 + 1 - \langle\beta|\alpha\rangle - \langle\alpha|\beta\rangle + 1) = \\ &= \frac{1}{8}(4 - 2\langle\alpha|\beta\rangle - 2\langle\beta|\alpha\rangle) = && (|\alpha\rangle \text{ and } |\beta\rangle \text{ have real coefficients}) \\ &= \frac{1}{8}(4 - 4\langle\alpha|\beta\rangle) = \\ &= \frac{1}{2}(1 - \langle\alpha|\beta\rangle). \end{aligned}$$

Now, let us apply some algebraic manipulations. Given the state $|\gamma\rangle$ (Equation 8.2), let us first extract the summation over the index register $|j\rangle$ within $|\alpha\rangle$ and $|\beta\rangle$. This results in a state of the form

$$\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}[...]\,|j\rangle,$$

where [...] encompasses all circuit qubits but those being part of the index register. Subsequently, let us trace out, i.e., discard, the second circuit qubit and the features register $|i\rangle$ within $|\alpha\rangle$ and $|\beta\rangle$. Mathematically, this corresponds to calculating the partial trace over these qubits of the density operator that describes the system. In this way, a reduced version of the final state that includes only the first circuit qubit and the index register is obtained; this reduced version can be expressed as

$$\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}\left[\sqrt{P(0\mid j)}\,|0\rangle + \sqrt{P(1\mid j)}\,|1\rangle\right]|j\rangle.$$

Ultimately, let us leverage the derivation illustrated above. In practical terms, $P(1 \mid j)$ is equal to $\frac{1}{2}(1 - \langle \mathbf{x}_j, \mathbf{x}'_j\rangle)$, as the summation over the index register (along with its coefficient) has been pulled out. Moreover, $P(0 \mid j)$ must be equal to $1 - P(1 \mid j)$ according to the law of total probability. These considerations lead to the following definition of the reduced final state:

$$|\delta\rangle = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}\left[\sqrt{1 - s(\mathbf{v}_j, \mathbf{v}')}\,|0\rangle + \sqrt{s(\mathbf{v}_j, \mathbf{v}')}\,|1\rangle\right]|j\rangle, \quad (8.3)$$

where

$$s(\mathbf{v}_j, \mathbf{v}') = P(qubit_1 = 1 \mid j) = \frac{1}{2}(1 - \langle \mathbf{x}_j, \mathbf{x}'_j\rangle), \quad (8.4)$$

with $qubit_1$ being the first qubit in the circuit. Essentially, $s(\mathbf{v}_j, \mathbf{v}')$ is a similarity measure related to the squared Euclidean distance between $\mathbf{v}_j$ and $\mathbf{v}'$. Indeed, a lower distance corresponds to a higher

Table 8.1: Properties of the two encodings. Notice that the range of values of $s(\mathbf{v}_j, \mathbf{v}')$ is determined by the preprocessed test instance $\mathbf{v}'$ (taken from [127]).

	Extension	Translation
$\langle \mathbf{x}_j, \mathbf{x}'_j \rangle$ value	$\frac{4}{3}(\|\mathbf{v}_j\|^2 - 2\langle \mathbf{v}_j, \mathbf{v}' \rangle)$	$\|\mathbf{v}_j\|^2 - 2\langle \mathbf{v}_j, \mathbf{v}' \rangle - \frac{1}{4}$
Minimum $s(\mathbf{v}_j, \mathbf{v}')$ range	[0.333, 0.5]	[0.5, 0.625]
Maximum $s(\mathbf{v}_j, \mathbf{v}')$ range	[0, 0.666]	[0.25, 0.75]

$s(\mathbf{v}_j, \mathbf{v}')$ value. Concerning $\langle \mathbf{x}_j, \mathbf{x}'_j \rangle$, its value depends on the selected encoding (see Table 8.1).

Examining the first row of Table 8.1, two aspects become clear: $\langle \mathbf{x}_j, \mathbf{x}'_j \rangle$ is directly related to the squared Euclidean distance between $\mathbf{v}_j$ and $\mathbf{v}'$ (in both cases), and the term $\|\mathbf{v}'\|^2$ is absent. The latter is not a problem, as $\|\mathbf{v}'\|^2$ is the same across all training instances. Instead, the second and third rows of the same table allow elucidating the strengths of each encoding. Specifically, the *extension* encoding maximises the range of values of $s(\mathbf{v}_j, \mathbf{v}')$, enabling a more robust representation of the similarity values, less susceptible to the presence of noise. Conversely, the *translation* encoding maximises the probability of obtaining 1 by measuring the first qubit, a favorable scenario for reasons that will be clarified in the subsequent section. Lastly, it is important to notice that the range of $s(\mathbf{v}_j, \mathbf{v}')$ is determined by $\mathbf{v}'$. In detail, the minimum range is given by a test instance with norm 0, while the maximum range is given by a test instance with norm $\frac{1}{2}$ (the maximum allowed value).

8.1.1.4 Measurements and Distance Estimate(s)

After executing the *Bell-H* operation, the qubits present in Equation (8.3), namely, the first circuit qubit and the index register $|j\rangle$, are measured. More precisely, the first qubit in the circuit is the first one to be measured. In this way, upon obtaining 1 (0), the nearest neighbors indices will have the highest (lowest) probability values. If the index register qubits were measured first, the probability distribution would be uniform. By repeating the execution of the circuit and the measurement process, the joint probabilities $P(0, j)$ and $P(1, j)$ are estimated as relative frequencies, enabling the estimation of the Euclidean distances. In fact, the following relationships apply:

$$P(0, j) = \frac{1 + \langle \mathbf{x}_j, \mathbf{x}'_j \rangle}{2N} \implies \langle \mathbf{x}_j, \mathbf{x}'_j \rangle = 2N \times P(0, j) - 1, \tag{8.5}$$

$$P(1, j) = \frac{1 - \langle \mathbf{x}_j, \mathbf{x}'_j \rangle}{2N} \implies \langle \mathbf{x}_j, \mathbf{x}'_j \rangle = 1 - 2N \times P(1, j). \tag{8.6}$$

Furthermore, for the *extension* encoding (refer to Table 8.1),

$$d(\mathbf{v}_j, \mathbf{v}') = \sqrt{\frac{3}{4}\langle \mathbf{x}_j, \mathbf{x}'_j \rangle + \|\mathbf{v}'\|^2}, \tag{8.7}$$

with $d(\mathbf{v}_j, \mathbf{v}')$ being the Euclidean distance between $\mathbf{v}_j$ and $\mathbf{v}'$, whereas, for the *translation* encoding,

$$d(\mathbf{v}_j, \mathbf{v}') = \sqrt{\langle \mathbf{x}_j, \mathbf{x}'_j \rangle + \frac{1}{4} + \|\mathbf{v}'\|^2}. \tag{8.8}$$

Independently of the chosen encoding, it is possible to estimate the Euclidean distances $d(\mathbf{v}_j, \mathbf{v}')$ starting from either $P(0, j)$ or $P(1, j)$. In this study, two methods for combining the information carried by the two joint probabilities have been developed and evaluated, namely, *avg* and *diff*. Specifically, the *avg* distance estimate corresponds to the mean of the Euclidean distance estimated from $P(0, j)$ and the Euclidean distance estimated from $P(1, j)$. Conversely, for the *diff* distance estimate, the scalar product value is determined as

$$\langle \mathbf{x}_j, \mathbf{x}'_j \rangle = N \times (P(0, j) - P(1, j)),$$

and the Euclidean distance is obtained through Equation (8.7) or (8.8), depending on the encoding.

Lastly, two aspects are worth to be mentioned. To this end, let us take into account two preprocessed training instances $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$, with $j_1, j_2 \in \{0, ..., N-1\}$, and the preprocessed test instance $\mathbf{v}'$. Let $d_0(\mathbf{v}_{j_1}, \mathbf{v}')$ and $d_1(\mathbf{v}_{j_1}, \mathbf{v}')$ denote the Euclidean distances from $\mathbf{v}'$ estimated from the joint probabilities $P(0, j_1)$ and $P(1, j_1)$ for $\mathbf{v}_{j_1}$ (the same for $\mathbf{v}_{j_2}$). The following relationships apply:

$$avg = \frac{d_0(\mathbf{v}_{j_1}, \mathbf{v}') + d_1(\mathbf{v}_{j_1}, \mathbf{v}')}{2},$$

$$diff = \sqrt{\frac{d_0(\mathbf{v}_{j_1}, \mathbf{v}')^2 + d_1(\mathbf{v}_{j_1}, \mathbf{v}')^2}{2}}.$$

The first one is the definition of the *avg* distance estimate, while the latter can be verified using Equation (8.7) (or 8.8, depending on the chosen encoding) in conjunction with Equations (8.5) and (8.6). Then, the preprocessed training instances $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$ might be arranged differently based on the *avg* and *diff* distance estimates. Indeed, let us consider the following situation:

$$\begin{aligned} d_0(\mathbf{v}_{j_1}, \mathbf{v}') &= 0.5 & \qquad d_0(\mathbf{v}_{j_2}, \mathbf{v}') &= 0.4 \\ d_1(\mathbf{v}_{j_1}, \mathbf{v}') &= 0.29 & \qquad d_1(\mathbf{v}_{j_2}, \mathbf{v}') &= 0.4\,. \end{aligned}$$

In the considered scenario, the *avg* distance estimates for $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$ are 0.395 and 0.4, respectively, whereas the *diff* distance estimates are 0.409 and 0.4, respectively. Therefore, $\mathbf{v}_{j_1}$ is closer than $\mathbf{v}_{j_2}$ (to $\mathbf{v}'$) according to *avg* but further according to *diff.* Secondly, assuming that $d_0(\mathbf{v}_{j_1}, \mathbf{v}')$ and $d_1(\mathbf{v}_{j_1}, \mathbf{v}')$ can be mathematically calculated, i.e., the arguments of the square root in Equation (8.7) (or 8.8, depending on the chosen encoding) are non-negative, the *avg* distance estimate is consistently lower than or equal to the corresponding *diff* estimate. Indeed, the contrary would be true only if $(d_0(\mathbf{v}_{j_1}, \mathbf{v}') - d_1(\mathbf{v}_{j_1}, \mathbf{v}'))^2 < 0$, which is not feasible. In cases where the assumption about the square root arguments does not hold because of the state counts obtained, it is possible for the *avg* distance estimate to be higher than the corresponding *diff* estimate due to the policy adopted in the current implementation (which is presented in Section 8.2).

8.1.1.5 k Nearest Neighbors and Classification

After estimating all the Euclidean distances $d(\mathbf{v}_j, \mathbf{v}')$, the training samples are classically sorted based on them. Subsequently, the k nearest neighbors are determined, and the test instance is classified through a majority voting based on the nearest neighbors labels.

8.1.2 Complexity Observations

In terms of complexity, the proposed quantum k-NN algorithm differs from its classical counterpart in the estimation/computation of the Euclidean distances. In fact, the preprocessing of data, the identification of the k nearest neighbors (by means of a distance sorting operation), and the label prediction are executed classically in both instances.

The classical k-NN algorithm computes the exact Euclidean distances with a complexity of $O(Nd)$. In contrast, the quantum k-NN algorithm only estimates the Euclidean distances. To do this, multiple *shots*, i.e., measurements, iterations, are required. Specifically, each iteration involves preparing the initial state, executing the *Bell-H* quantum circuit, and measuring the qubits state. The complexity of the initial state preparation depends on the availability of a QRAM. Indeed, with a QRAM, assuming that the real values x_{ji} and x'_{ji} are stored as classical floating point numbers, the preparation of the initial state has a complexity of $O(\log(NF))$. This results from the possibility of retrieving the states $|\alpha\rangle$ and $|\beta\rangle$ from the QRAM with a complexity of $O(\log(NF))$. In the absence of a QRAM, the desired state must be prepared starting from $|0\rangle^{\otimes(1+I)}$, and the required number of gates varies depending on the quantum processor's architecture. On the other hand, the *Bell-H* quantum circuit involves a constant number of elementary gates, resulting in a complexity of $O(1)$. The complexity of the measurement step is also constant, as the measurements are simultaneous. Eventually, provided the state counts, computing the distance estimates has a complexity of $O(N)$. Therefore, if a QRAM is available, the overall complexity becomes $O(shots \times \log(NF) + N)$, which corresponds to $O(shots \times (\log N + \log d) + N)$. It is important to observe that as N increases, the number of shots required for properly estimating the Euclidean distances also increases. In conclusion, assuming a fixed value of *shots*, estimating the Euclidean distances in the quantum k-NN algorithm has a complexity of $O(\log d + N)$, which is less than $O(Nd)$. Conversely, assuming that *shots* depends logarithmically on N, the complexity is $O(\log N \log d + N)$, thus still lower than $O(Nd)$.

8.2 Implementation

This section provides details about the implementation of the Euclidean distance quantum k-NN algorithm introduced in Section 8.1. Specifically, the algorithm has been implemented in Python using Qiskit, i.e., the open-source software development kit supplied by IBM [3]. The code, available at `https://github.com/ZarHenry96/euclidean-quantum-k-nn`, supports various execution modes, including:

- *classical*, which, after performing the preprocessing step described in Section 8.1.1.1, executes a classical k-NN algorithm with the Euclidean distance metric. Thus, it does not involve any quantum circuit;
- *statevector*, representing an ideal execution with infinite iterations. In practice, the final state vector of the circuit is processed in order to provide the output. Hence, no measurement is performed in this case;
- *simulation*, called *local simulation* in the code, which provides state counts by sampling from the final probability distribution of the circuit.

No noise is considered in these modalities, and a representative circuit for the *simulation* mode is provided in Figure 8.1.

The implementation of the algorithm follows the description provided in Section 8.1.1, but some technical details are worth to be mentioned. Regarding the preprocessing step, the normalization of each data feature is performed by subtracting the mean of the maximum and minimum feature values in the training set, and dividing by the feature range (calculated on the training data, and set to 1 in the case of a constant feature) multiplied by $\sqrt{d}$. Additionally, a clipping operation is applied to test instance features exceeding the target range. Now, let us consider the execution modes involving quantum circuits, as the *classical* one is straightforward. In detail, the initial state preparation is accomplished by computing the amplitudes of all qubits except the first one (being in state $|0\rangle$ by default) and providing them as input to Qiskit's initialization function. Concerning the indices not corresponding to training instances (present when N is not a power of 2), they are retained in the joint probabilities estimation and excluded in the distance values calculation. Furthermore, if the square

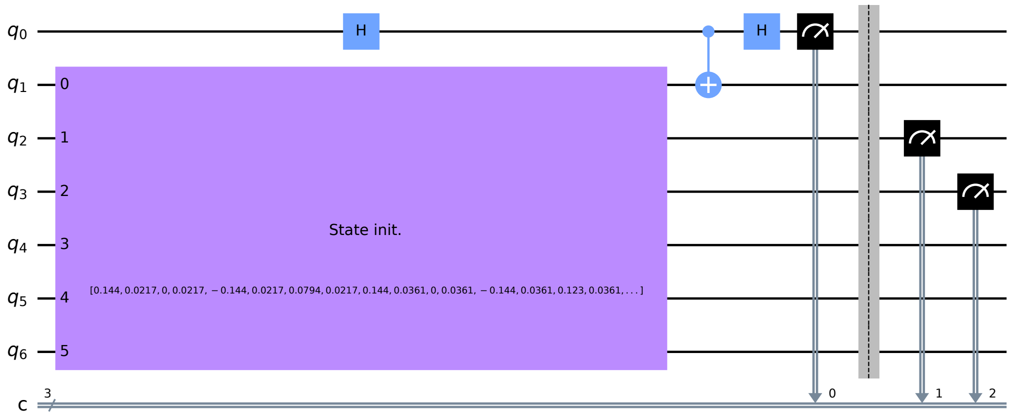


Figure 8.1: Example of quantum circuit for the quantum k-NN based on the Euclidean distance. In this case, $N = 4$, $d = 2$, and the execution modality is *simulation* (*statevector* does not include the final measurements) (taken from [127]).

root argument in Equation (8.7) or (8.8) is negative or exceeds 1 because of the state counts obtained, the distance value is set to 0 and 1, respectively. Lastly, training instances with the same Euclidean distance are arranged by increasing training set index (this also applies to the *classical* execution modality).

In conclusion, it is worth mentioning that, in the *simulation* mode, Laplace smoothing [119] has been used in the joint probabilities estimation. Basically, for a given number of counts c for the state $|a\rangle\,|j\rangle$, with $a \in \{0, 1\}$, the probability value $P(a, j)$ is estimated as

$$P(a, j) = \frac{c + p}{shots + 2Np},$$

with p being the number of pseudocounts introduced for each state, and *shots* being the total number of measurements (iterations). Specifically, pseudocounts are added only to the counts of the significant indices, namely, the indices corresponding to existing training instances.

8.3 Empirical Evaluation

In this section, the evaluated methods, the chosen datasets, the experimental setup used, and the obtained results are described. Specifically, the experiments have been executed on a shared machine equipped with an Intel Xeon Gold 6238R processor running at 2.20 GHz and 125 GB of RAM.

8.3.1 Methods

The Euclidean distance quantum k-NN presented in Section 8.1 has been evaluated under different execution modalities and (*encoding*, *distance estimate*) configurations, as detailed in Table 8.2. Experiments on real quantum machines have not been executed since large-enough free-access devices were not available at the time of running the experiments. Additionally, for comparison purposes, a few classical baseline methods (enumerated in the same table) have been taken into account. Specifically, the results data for these last methods have been taken from the work by Zardini et al. [128] (more precisely, from the associated *figshare* repository [126]), thus they are the same results data presented in Section 7.2.4.6.

Table 8.2: Methods tested (taken from [127]).

Quantum k-NN with Euclidean distance

Execution modality	**Encoding**	**Distance estimate**
classical	-	-
statevector	extension, translation	avg, diff
simulation	extension, translation	avg, diff

Baseline methods
k-NN with cosine distance
random forest (100 trees)
SVM with {Gaussian, linear} kernel

8.3.2 Datasets

The datasets employed in these experiments have been fetched from the work by Zardini et al. [128] (specifically, from the associated *figshare* repository [125]), primarily to ensure the comparability with the baseline methods results. Hence, details about the selection criteria and the preprocessing steps can be found in Section 7.2.2, as it is based on that article. In particular, the datasets employed here, which have also been uploaded to a dedicated *figshare* repository [124], correspond to the versions without subsampling listed in Table 7.2. For the sake of convenience, their properties are reported in Table 8.3.

Table 8.3: Properties of the datasets used. Note that the dataset names are links leading to the UCI pages of the original versions of the datasets (taken from [127]).

Name	Classes	Size	Features
01_iris_setosa_versicolor	2	100	4
01_iris_setosa_virginica	2	100	4
01_iris_versicolor_virginica	2	100	4
02_transfusion [122]	2	748	4
03_vertebral_column_2C	2	310	6
04_seeds_1_2	2	140	7
05_ecoli_cp_im	2	220	7
06_glasses_1_2	2	146	9
07_breast_tissue_adi_fadmasgla	2	71	9
08_breast_cancer [83]	2	116	9
09_accent_recognition_uk_us	2	210	12
10_leaf_11_9 [108]	2	30	14

8.3.3 Experimental Setup

In these experiments, the stratified κ-fold cross-validation has been chosen as the validation technique. Essentially, each dataset is partitioned into κ folds (subsets). Subsequently, $\kappa - 1$ folds are designated as the training set, while the remaining one represents the test set. This last step is iterated κ times, ensuring that each subset is employed once as the test set. The term “stratified” indicates that the class ratio in the folds is maintained as close as possible to that of the original dataset. Moreover, the same seed has been used for the folds generation across all experiments, guaranteeing that all methods have been tested on the same folds.

The parameter setup used for the quantum k-NN experiments is detailed in Table 8.4. Specifically, for all execution modes, the number of folds (*folds*) has been set to 5, a typical choice in ML. Additionally, four distinct values for the number of nearest neighbors selected (k) have been assessed. Regarding the *simulation* mode, all (*encoding*, *distance estimate*) configurations have been evaluated with 1024 measurements (*shots*), the default value in Qiskit, and only the best-performing one has been tested with other values. The number of pseudocounts for Laplace smoothing has been arbitrarily established at 10, and 5 runs (with distinct seeds) have been executed for obtaining statistical evidence. In particular, the simulation seed for each test sample has been randomly generated from a “root” run seed. Lastly, it is important to mention that the evaluation of the different k values has been performed using distinct seeds, while the *avg* and *diff* distance estimates have been assessed on the same seeds (i.e., for each test sample, the two distance estimates have been calculated with identical state counts).

Concerning the baseline methods taken into account for comparison, as explained in Section 7.2.3, a standard min-max normalization procedure had been applied to the input data features, resulting in an output range of $[0, 1]$. In addition, the classical k-NN with cosine distance metric had been evaluated using the same number of folds, folds generation seed, and k values used for the quantum k-NN. Given its stochastic nature, five runs had also been executed for the random forest.

Table 8.4: Parameter setup for the quantum k-NN experiments (taken from [127]).

Common parameters	
folds	5
k	3, 5, 7, 9

Simulation parameters	
shots	512[a], 1024, 2048[a], 4096[a], 8192[a]
pseudocounts	10
runs	5

[a] Only the best (*encoding, distance estimate*) configuration of the quantum k-NN has been tested with this number of shots.

8.3.4 Results

The results are presented using scatterplots and boxplots. Specifically, the quantum k-NN's performance have been assessed in terms of both classification accuracy and correctness of the identified nearest neighbors, while, for the baseline methods, only the classification accuracy has been taken into account. In more detail, for given a fold, the accuracy is defined as

$$accuracy = \frac{number\ of\ correctly\ classified\ instances\ in\ the\ fold}{total\ number\ of\ instances\ in\ the\ fold}$$

(this definition coincides with the one provided in Equation 7.3). In cases involving multiple runs, the average value across runs is reported. Concerning the correctness of the identified nearest neighbors, the Jaccard index and the Average Jaccard score [40] have been considered. In detail, for a given test instance, the Jaccard index (JI) is defined as

$$Jaccard\ index\ (JI) = \frac{|\mathcal{S}_c \cap \mathcal{S}_f|}{|\mathcal{S}_c \cup \mathcal{S}_f|}\,,$$

with $\mathcal{S}_c$ denoting the (classically computed) set of correct nearest neighbors, and $\mathcal{S}_f$ representing the set of identified nearest neighbors. Since each test sample has a corresponding Jaccard index value, the mean value has been taken into account for each fold, and the average of these average values across runs is reported. The same approach has been applied for the Average Jaccard (AJ) score, which, for a given test sample, is defined as

$$Average\ Jaccard\ (AJ) = \frac{1}{k}\sum_{m=1}^{k} h(\mathcal{S}_{cm}, \mathcal{S}_{fm})\,,$$

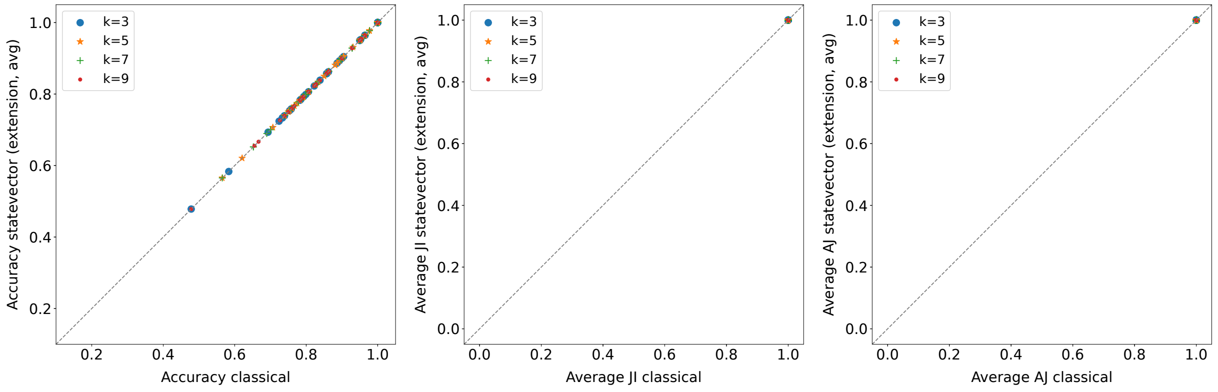
with k being the number of nearest neighbors selected, h being the function that calculates the Jaccard index, and $\mathcal{S}_{cm}$ being the set that comprises the correct nearest neighbors up to the m-th most similar element (the same holds for $\mathcal{S}_{fm}$). Lastly, the statistical significance of the results has been evaluated via the Wilcoxon signed-rank test [116], given the paired nature of the data. In certain cases (difference boxplots), the one-sample T-test [39] has also been considered.

8.3.4.1 Execution Modalities Comparison

Let us focus first on the *classical* and *statevector* execution modalities. According to Figure 8.2, these modalities are equivalent in terms of accuracy (Figure 8.2a), Jaccard index (Figure 8.2b), and Average Jaccard score (Figure 8.2c). This is also confirmed by the Wilcoxon signed-rank test, whose results are reported in Table 8.5. In particular, only one *statevector* configuration, namely, (*extension*, *avg*), is showcased here. However, the results are the same for all of them[1]. This demonstrates the correctness of the algorithm introduced in Section 8.1, as it achieves identical results to its classical counterpart in the ideal case. It is also crucial to recall that the quantum algorithm's advantage over its classical counterpart consists in the execution time.

Next, let us consider the *statevector* and *simulation* execution modalities. Figure 8.3 displays the comparison in terms of accuracy (Figure 8.3a), Jaccard index (Figure 8.3b), and Average Jaccard score (Figure 8.3c) for the (*extension*, *avg*) configuration. In practice, *statevector* statistically outperforms *simulation* with 1024 shots in both classification accuracy and correctness of the identified nearest neighbors, as confirmed by Table 8.6, demonstrating a significant drop in performance when simulating the algorithm with a limited number of shots. Specifically, the degradation is more pronounced for the Jaccard index and the Average Jaccard score. These findings are valid also for the other (*encoding*, *distance estimate*) configurations, as shown in Figures 8.4 to 8.6 and Tables 8.7 to 8.9.

[1] Actually, in the case of the *translation* encoding, the Average Jaccard score for a single fold of a single dataset has turned out to be lower than that of *classical*. The reason is the swap, for some test samples, of two nearest neighbors, caused by the numerical approximation of the distance values. Nevertheless, the difference is not statistically significant (p-value=0.317 for all k values).

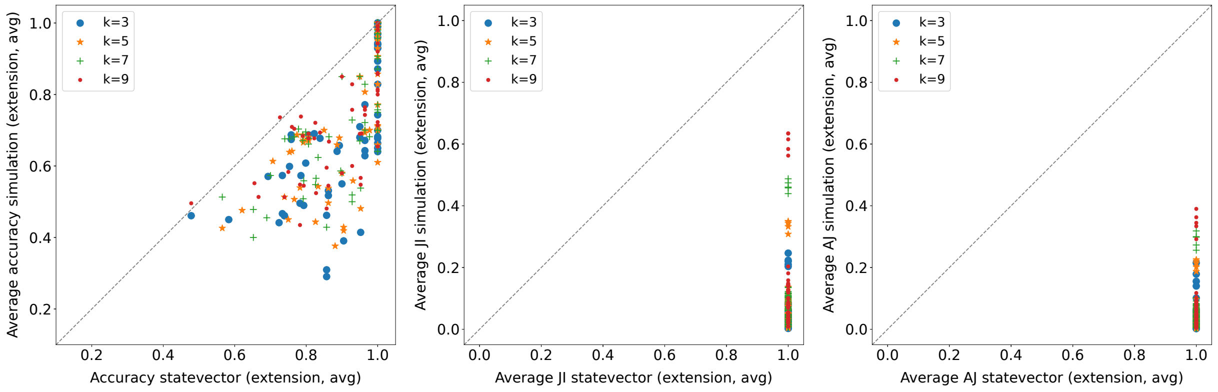


(a) (b) (c)

Figure 8.2: Comparison between *classical* and *statevector* execution modalities in terms of accuracy (a), Jaccard index (b), and Average Jaccard score (c). The configuration used for *statevector* is (*extension*, *avg*), but the results are the same for all configurations. Each point is related to a dataset fold (taken from [127]).

Table 8.5: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the distributions shown in Figure 8.2. The values reported in the table are the p-values obtained (taken from [127]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 8.2a	1.000	1.000	1.000	1.000
Figure 8.2b	1.000	1.000	1.000	1.000
Figure 8.2c	1.000	1.000	1.000	1.000

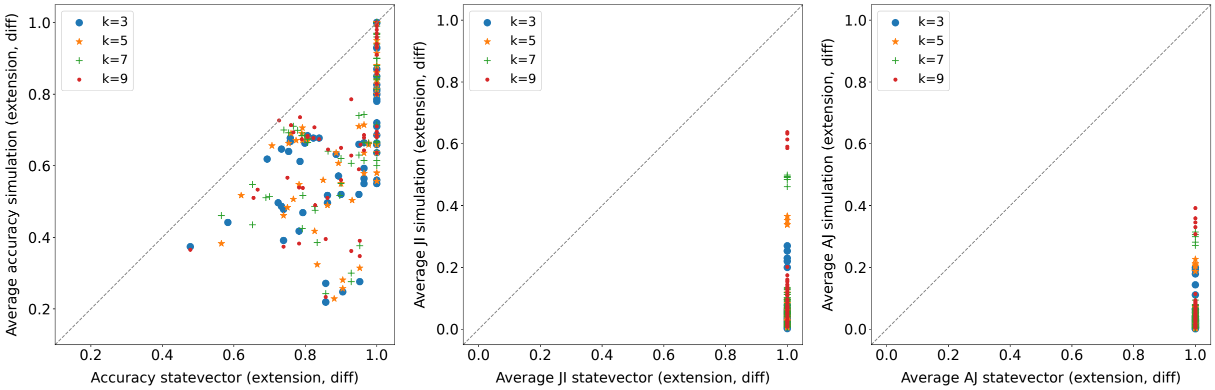


(a) (b) (c)

Figure 8.3: Comparison between *statevector* (*extension*, *avg*) and *simulation* (*extension*, *avg*) in terms of accuracy (a), Jaccard index (b), and Average Jaccard score (c). The number of shots for *simulation* is 1024, and each point is related to a dataset fold (taken from [127]).

Table 8.6: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the distributions shown in Figure 8.3. The values reported in the table are the p-values obtained (taken from [127]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 8.3a	1.624E-10	1.622E-10	5.140E-10	1.608E-09
Figure 8.3b	1.626E-11	1.630E-11	1.629E-11	1.630E-11
Figure 8.3c	1.629E-11	1.630E-11	1.630E-11	1.630E-11

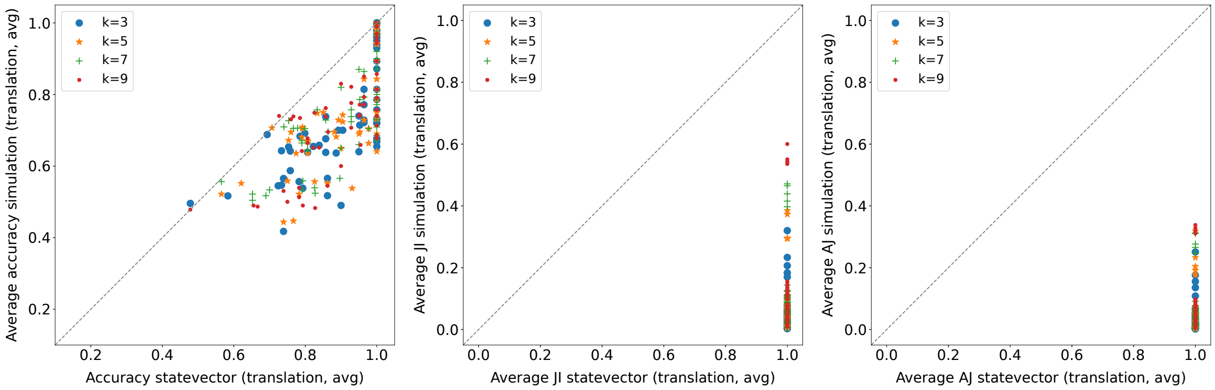


(a) (b) (c)

Figure 8.4: Comparison between *statevector* (*extension*, *diff*) and *simulation* (*extension*, *diff*) in terms of accuracy (a), Jaccard index (b), and Average Jaccard score (c). The number of shots for *simulation* is 1024, and each point is related to a dataset fold (taken from [127]).

Table 8.7: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the distributions shown in Figure 8.4. The values reported in the table are the p-values obtained (taken from [127]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 8.4a	1.106E-10	1.106E-10	7.530E-11	1.719E-10
Figure 8.4b	1.628E-11	1.629E-11	1.630E-11	1.630E-11
Figure 8.4c	1.630E-11	1.630E-11	1.630E-11	1.630E-11

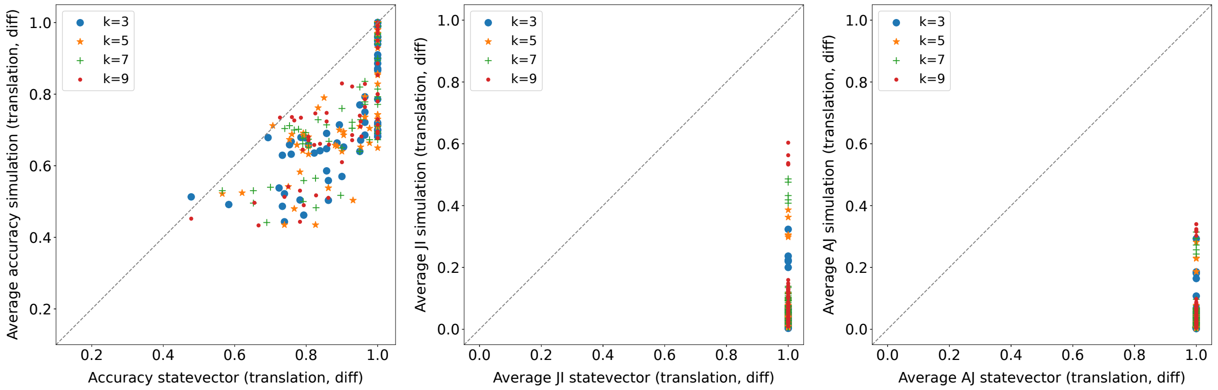


(a) (b) (c)

Figure 8.5: Comparison between *statevector* (*translation*, *avg*) and *simulation* (*translation*, *avg*) in terms of accuracy (a), Jaccard index (b), and Average Jaccard score (c). The number of shots for *simulation* is 1024, and each point is related to a dataset fold (taken from [127]).

Table 8.8: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the distributions shown in Figure 8.5. The values reported in the table are the p-values obtained (taken from [127]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 8.5a	1.234E-10	3.492E-10	3.492E-10	6.149E-10
Figure 8.5b	1.628E-11	1.630E-11	1.630E-11	1.630E-11
Figure 8.5c	1.630E-11	1.630E-11	1.630E-11	1.630E-11



(a)

(b)

(c)

Figure 8.6: Comparison between *statevector* (*translation*, *diff*) and *simulation* (*translation*, *diff*) in terms of accuracy (a), Jaccard index (b), and Average Jaccard score (c). The number of shots for *simulation* is 1024, and each point is related to a dataset fold (taken from [127]).

Table 8.9: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the distributions shown in Figure 8.6. The values reported in the table are the p-values obtained (taken from [127]).

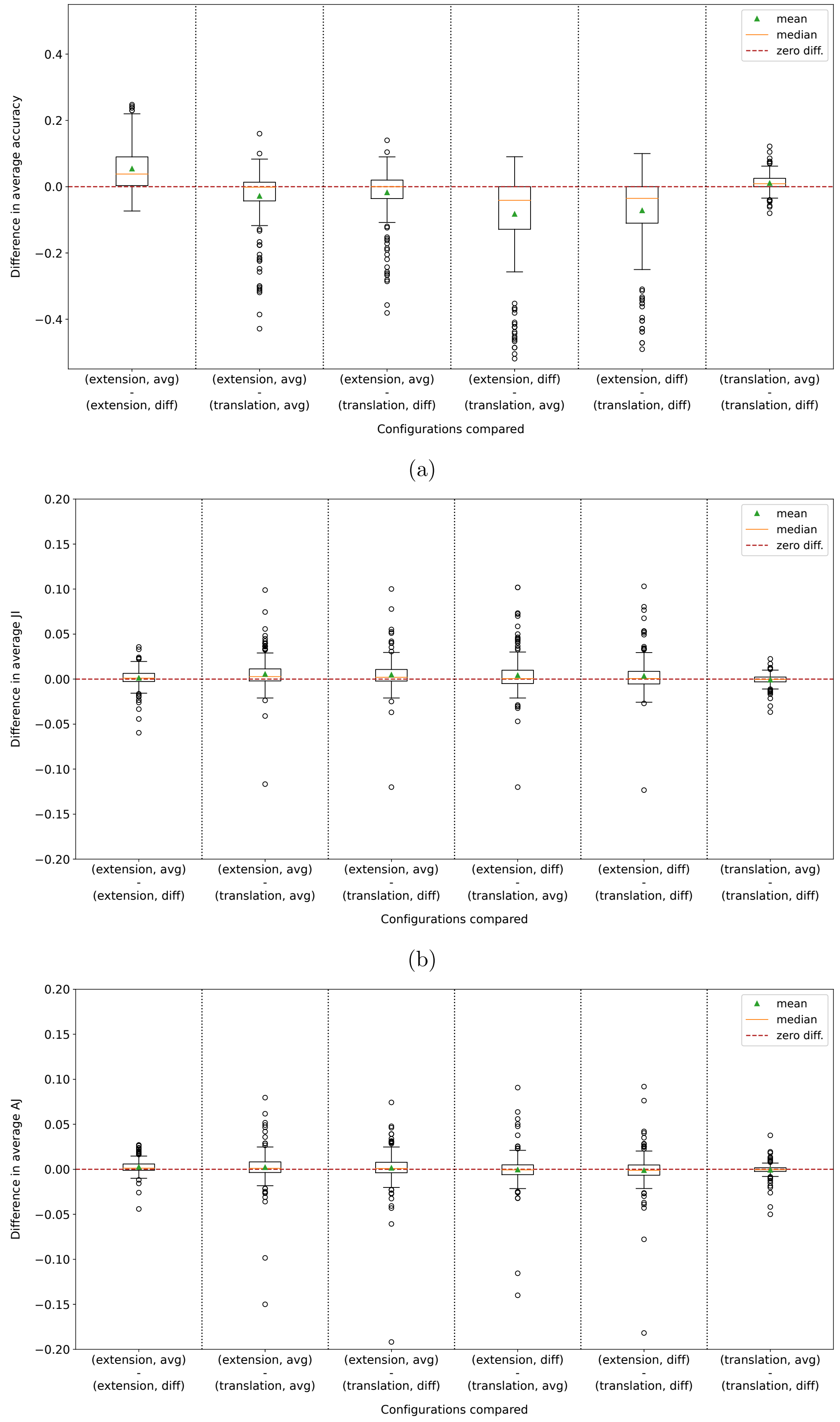
	k=3	**k=5**	**k=7**	**k=9**
Figure 8.6a	1.379E-10	1.712E-10	7.516E-11	2.523E-10
Figure 8.6b	1.627E-11	1.629E-11	1.630E-11	1.630E-11
Figure 8.6c	1.629E-11	1.630E-11	1.630E-11	1.630E-11

8.3.4.2 Encodings and Distance Estimates Comparison

This analysis focuses solely on the *simulation* modality (with 1024 shots) as all (*encoding*, *distance estimate*) configurations have obtained identical results in the case of the *statevector* modality (the distance estimates are exact in the ideal case). Specifically, the comparisons are illustrated via difference boxplots, in which each point depicts the difference for a (*dataset fold*, *k value*) pair. The comparisons in terms of accuracy, Jaccard index, and Average Jaccard score are provided in Figure 8.7.

Regarding the classification accuracy (Figure 8.7a), the best results have been obtained by the (*translation*, *avg*) configuration, which has statistically outperformed all the others, as reported in Table 8.10a. Overall, the *translation* encoding has demonstrated superior performance compared to the *extension* encoding in terms of accuracy. In addition, with the same encoding, the *avg* distance estimate has exhibited better performance than the *diff* distance estimate. Eventually, it is worth observing that all the differences are statistically significant with respect to both median (Wilcoxon signed-rank test) and mean (one-sample T-test), excluding the (*extension*, *avg*) - (*translation*, *diff*) comparison in terms of median.

Unexpectedly, the configuration that has achieved the highest classification accuracy differs from the configuration that has identified the best nearest neighbors. In fact, as shown in Figure 8.7b, the configuration that has performed the best in terms of Jaccard index is (*extension*, *avg*), and its difference with respect to the other configurations is almost always significant, as indicated in Table 8.10b. Overall, the *extension* encoding has performed better than the *translation* encoding in terms of Jaccard index. Instead, no distance estimate has clearly outperformed the other one in this case. Indeed, with the *extension* encoding, the *avg* distance estimate has achieved better results, while, with the *translation* encoding, the *diff* distance estimate has performed better. The majority of the differences are statistically significant in both median and mean, with a few exceptions such as the (*extension*, *avg*) - (*extension*, *diff*) comparison in mean, and the (*translation*, *avg*) - (*translation*, *diff*) comparison for both statistics. Concerning the Average Jaccard score (Figure 8.7c), a similar trend is observed, with (*extension*, *avg*) emerging as the best configuration. Nevertheless, in this case, the *extension* encoding with the *diff* distance estimate has achieved the worst results. This means that, among the k nearest neighbors returned by this configuration, the correct ones are the most dissimilar



(a)

(b)

(c)

Figure 8.7: Comparison of (*encoding*, *distance estimate*) configurations in terms of accuracy (a), Jaccard index (b), and Average Jaccard score (c) for the *simulation* execution modality. The number of shots is 1024, and each data point corresponds to the difference for a (*dataset fold*, *k value*) pair (taken from [127]).

Table 8.10: Wilcoxon signed-rank test and one-sample T-test applied to the distributions shown in Figure 8.7a (a), Figure 8.7b (b), and Figure 8.7c (c). Each column corresponds to a different comparison, and the first letter identifies the encoding (E=*extension*, T=*translation*), while the second letter identifies the distance estimate (A=*avg*, D=*diff*). The values reported in the tables are the p-values obtained ($\alpha = 0.05$) (taken from [127]).

(a)

	EA-ED	**EA-TA**	**EA-TD**	**ED-TA**	**ED-TD**	**TA-TD**
Wilcoxon	9.378E-26	3.019E-05	0.069	1.771E-24	1.539E-21	1.084E-09
T-test	1.236E-27	3.706E-07	0.001	1.404E-20	2.037E-18	2.492E-09

(b)

	EA-ED	**EA-TA**	**EA-TD**	**ED-TA**	**ED-TD**	**TA-TD**
Wilcoxon	0.003	8.888E-09	1.917E-07	0.010	0.043	0.104
T-test	0.054	6.775E-07	1.102E-05	0.001	0.005	0.054

(c)

	EA-ED	**EA-TA**	**EA-TD**	**ED-TA**	**ED-TD**	**TA-TD**
Wilcoxon	1.556E-07	0.002	0.008	0.315	0.111	0.043
T-test	6.693E-07	0.057	0.202	0.735	0.374	0.123

ones. Moreover, in the case of the Average Jaccard score, few differences turn out to be statistically significant, as detailed in Table 8.10c. Notably, the (*extension*, *avg*) configuration has statistically outperformed all the others in median and (*extension*, *diff*) also in mean.

8.3.4.3 Comparison with Baseline Methods

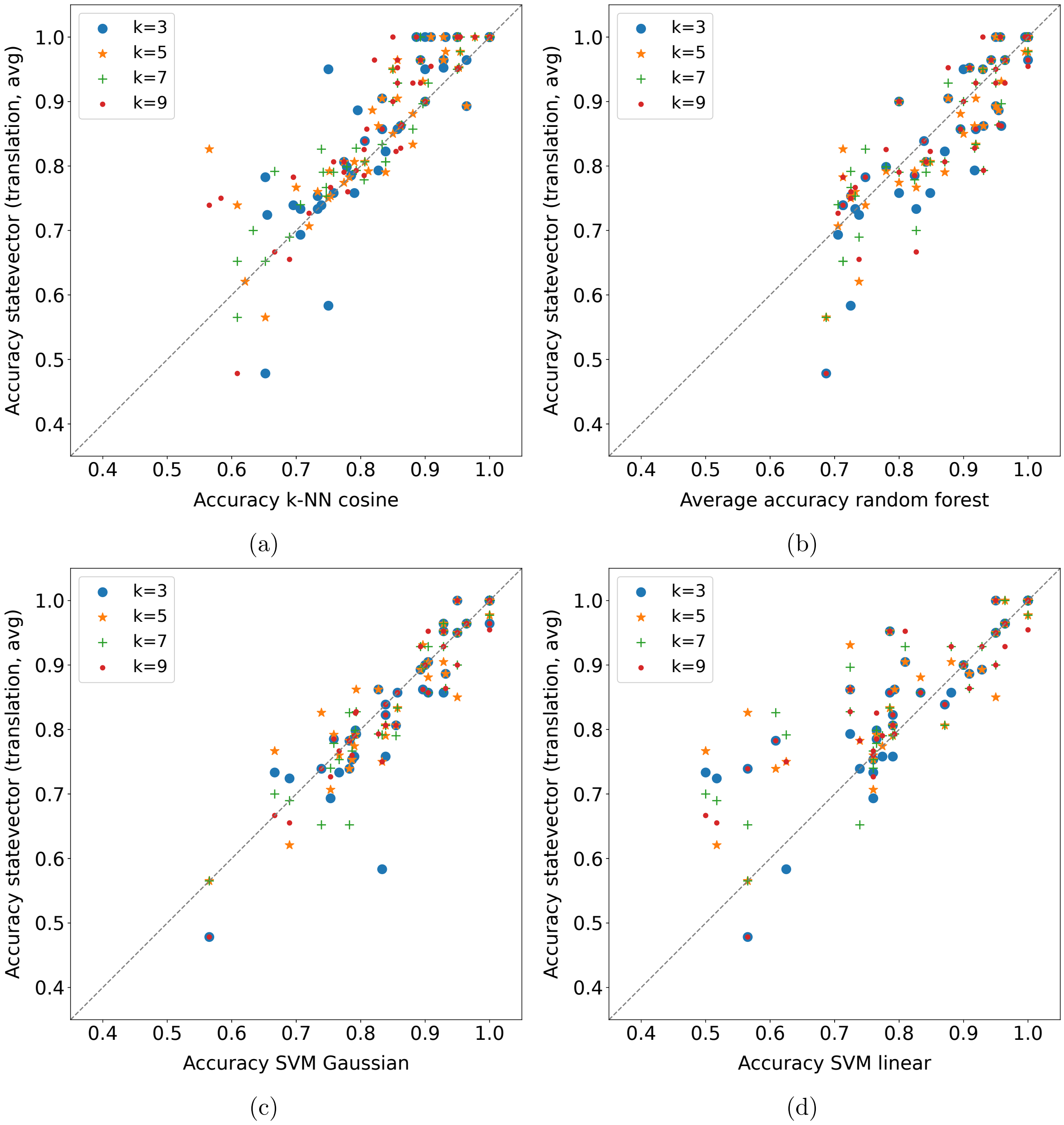
Several classical baseline methods have been considered for comparison. In particular, Figure 8.8 displays the comparison in terms of classification accuracy between the baseline methods and the *statevector* modality in the (*translation*, *avg*) configuration. However, it is worth noting that the configuration chosen for *statevector* is irrelevant, as elucidated in the preceding sections. Essentially, in the ideal case, the Euclidean distance quantum k-NN statistically outperforms both the classical k-NN with the cosine distance metric (Figure 8.8a) and the SVM with the linear kernel (Figure 8.8d), as reported in Table 8.11. Conversely, both the random forest (Figure 8.8b) and the SVM with the Gaussian kernel (Figure 8.8c) perform better than it. However, the only statistically significant difference observed is the one with the SVM with the Gaussian kernel, for $k = 3$.

Analogous plots for the *simulation* modality in the (*translation*, *avg*) configuration, which has obtained the best results in classification accuracy, are provided in Figure 8.9. In detail, all the considered baseline methods have statistically outperformed the proposed quantum k-NN in the *simulation* modality, as reported in Table 8.12.

8.3.4.4 Number of Shots Analysis

This final analysis explores the correlation between number of shots (measurements) and performance for the *simulation* modality. Specifically, in this case, only the best-performing quantum k-NN configuration, namely, the (*extension*, *avg*) configuration, has been taken into account. Indeed, the primary objective of the quantum k-NN is to accurately identify the k nearest neighbors, and the configuration in question has obtained the highest Jaccard index and Average Jaccard score, as illustrated in Section 8.3.4.2. The results are shown in Figure 8.10, utilizing difference boxplots with 512 as the baseline number of shots.

In practical terms, the higher the number of shots, the better the performance of the quantum k-NN. Specifically, this trend is more pronounced for both the Jaccard index (Figure 8.10b) and the Average Jaccard score (Figure 8.10c). However, the differences in absolute value are smaller when compared to those related to the accuracy (Figure 8.10a). Moreover, almost all differences are statistically significant in both median (Wilcoxon signed-rank test) and mean (one-sample T-test), as



(a)

(b)

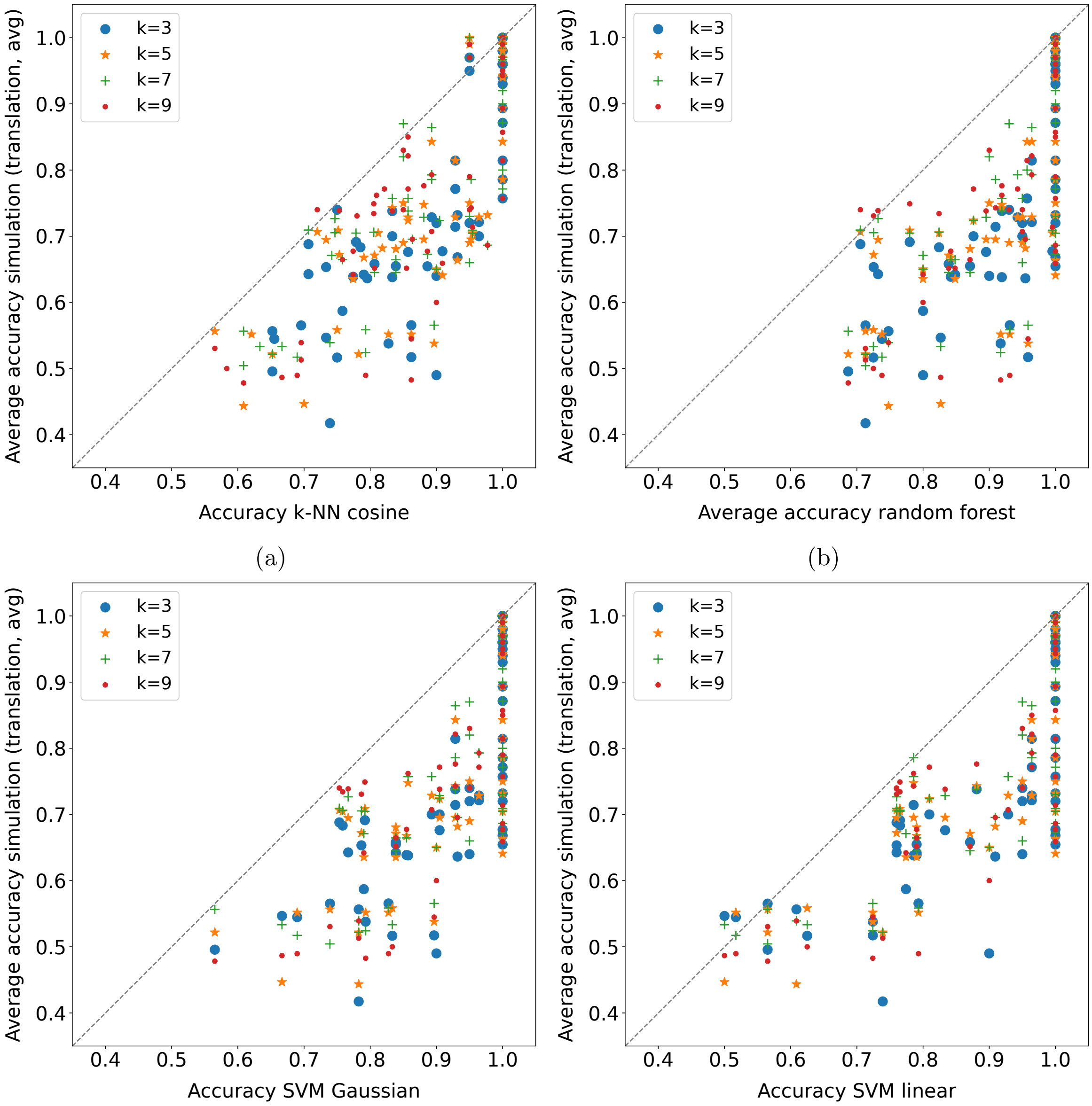
(c)

(d)

Figure 8.8: Comparison between some classical baseline methods and *statevector* in terms of accuracy. The configuration used for *statevector* is (*translation*, *avg*), but the results are the same for all configurations. Each point is related to a dataset fold (taken from [127]).

Table 8.11: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the distributions shown in Figure 8.8. The values reported in the table are the p-values obtained (taken from [127]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 8.8a	0.003	0.001	6.502E-05	8.149E-05
Figure 8.8b	0.074	0.165	0.095	0.258
Figure 8.8c	0.046	0.407	0.103	0.062
Figure 8.8d	0.045	0.005	0.012	0.007



(a)

(b)

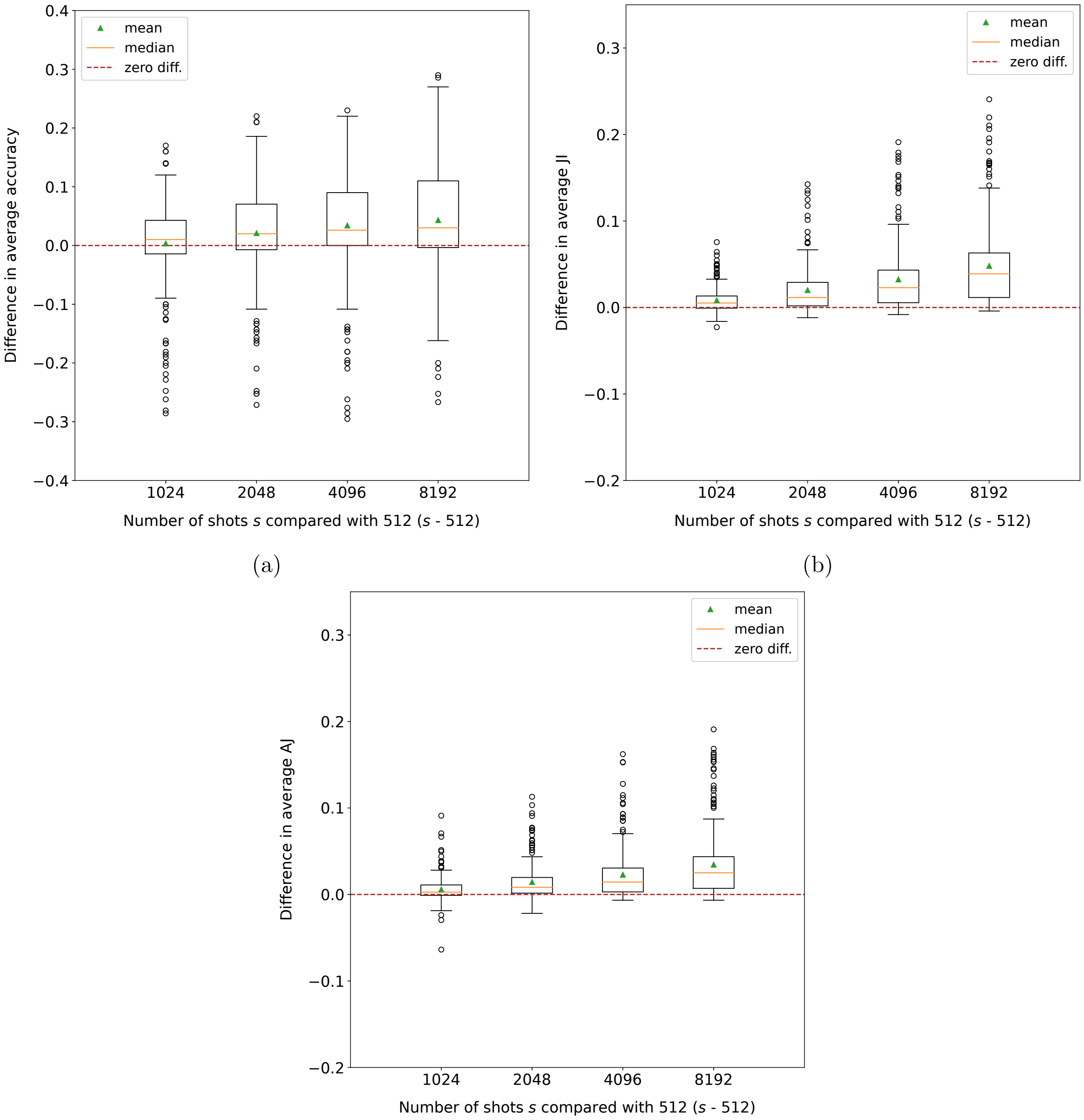
(c)

(d)

Figure 8.9: Comparison between some classical baseline methods and *simulation* (*translation*, *avg*) in terms of accuracy. The number of shots for *simulation* is 1024, and each point is related to a dataset fold (taken from [127]).

Table 8.12: Wilcoxon signed-rank test ($\alpha = 0.05$) applied to the distributions shown in Figure 8.9. The values reported in the table are the p-values obtained (taken from [127]).

	k=3	**k=5**	**k=7**	**k=9**
Figure 8.9a	2.147E-10	5.660E-10	8.949E-10	1.210E-09
Figure 8.9b	1.105E-10	2.521E-10	3.705E-10	6.632E-10
Figure 8.9c	1.105E-10	2.380E-10	3.494E-10	3.500E-10
Figure 8.9d	3.172E-10	3.976E-10	1.299E-09	3.502E-10



(a)

(b)

(c)

Figure 8.10: Comparison of different numbers of shots in terms of accuracy (a), Jaccard index (b), and Average Jaccard score (c) for the *simulation* execution modality in the (*extension*, *avg*) configuration. Each data point corresponds to the difference for a (*dataset fold*, *k value*) pair (taken from [127]).

Table 8.13: Wilcoxon signed-rank test and one-sample T-test applied to the distributions shown in Figure 8.10a (a), Figure 8.10b (b), and Figure 8.10c (c). The values reported in the tables are the p-values obtained ($\alpha = 0.05$) (taken from [127]).

(a)

	1024-512	**2048-512**	**4096-512**	**8192-512**
Wilcoxon	0.001	3.131E-07	2.432E-11	6.830E-11
T-test	0.473	5.648E-05	3.016E-08	7.904E-11

(b)

	1024-512	**2048-512**	**4096-512**	**8192-512**
Wilcoxon	1.151E-17	5.672E-33	4.761E-38	1.393E-40
T-test	4.330E-17	5.712E-26	1.955E-30	1.561E-38

(c)

	1024-512	**2048-512**	**4096-512**	**8192-512**
Wilcoxon	1.977E-11	8.804E-30	2.327E-37	1.030E-39
T-test	3.037E-09	6.719E-23	1.862E-27	3.622E-33

indicated in Table 8.13, with the only exception being the $1024 - 512$ comparison in terms of mean for the accuracy. Lastly, it is important to mention that, for a larger dataset, a higher number of shots is necessary to properly estimate the joint probability values.

8.4 Discussion

In this chapter, a new Euclidean distance quantum k-NN algorithm has been proposed and empirically evaluated. Specifically, two novel (and different) encodings of the input data into the quantum states amplitudes, not necessitating unit-norm normalization and requiring a low number of qubits, have been introduced. The quantum circuit used, not involving oracles, implements a SWAP-test-like procedure with a fixed number of elementary gates, enabling the parallel computation of quantities related to the pairwise Euclidean distances. In addition, two methods for estimating the Euclidean distance values, given the state counts resulting from the repeated measurements, have been presented. The final steps, namely, the training data sorting and the classification, are performed classically.

Concerning the empirical evaluation, the algorithm has been implemented in Python, using Qiskit, and tested on real-world datasets. The results have shown that, in the ideal case, the proposed quantum k-NN is equivalent to its classical counterpart in terms of both classification accuracy and nearest neighbors extraction, demonstrating the correctness of the formulation (the advantage of the proposed algorithm lies in the execution time). However, when simulating the algorithm with a finite number of measurements (1024), a significant drop in performance has been observed, regardless of the (*encoding*, *distance estimate*) configuration used. Among the tested configurations, (*extension*, *avg*) has turned out to be the best one. Indeed, although (*translation*, *avg*) has achieved the highest classification accuracy, the main objective of the algorithm is to identify the nearest neighbors, task in which (*extension*, *avg*) has obtained the best results (note that the distance estimate is *avg* in both cases). Concerning the classical baseline methods taken into account, half of them have outperformed the quantum k-NN in the ideal case, while all of them have outperformed the simulated version with 1024 shots. Lastly, the analysis regarding the number of measurements has confirmed that it is possible to enhance the algorithm's performance, when simulated, by increasing the number of shots.

Future investigations could involve evaluating the proposed model on different datasets, using a higher number of shots, and on real quantum devices, where noise is present.

9 Conclusion

In this thesis, several hybrid classical-quantum algorithms have been introduced and/or empirically evaluated. Indeed, the empirical evaluation is a fundamental part of the algorithmic research. Specifically, both quantum annealing and universal quantum computing have been addressed here, ranging from generic optimization to machine learning tasks.

On the quantum annealing side, the performance of QALS, a meta-heuristic approach for tackling QUBO optimization problems not directly mappable on the annealer topology, have been investigated first. Two efficient implementations, with different pros and cons, have been presented, and two complementary optimization problems have been considered for the empirical evaluation. The results have shown that, whenever a problem cannot be efficiently solved with classical methods, QALS allows processing larger instances than the standard embedding procedure provided by D-Wave, although the solutions found do not necessarily satisfy the problem constraints. Then, the QUBO formulation proposed by O'Gorman et al. for Bayesian network reconstruction has been implemented and empirically assessed on real-world problems. The formulation in question, with some minor improvements, has turned out to be successful for networks of small sizes and with no more than two parents per node, but good-quality solutions (in terms of energy) have also been found for larger networks. Additionally, for handling bigger BNSL instances, a divide et impera approach has been introduced. The experiments have demonstrated its superiority over the direct application of O'Gorman's formulation, although the solutions found were generally non-optimal, probably due to the non-ideal annealing setup used. Lastly, the local application of quantum-trained SVM models has been proposed and empirically evaluated in the remote sensing domain. To do this, FaLK-SVM, the framework for efficient local SVMs, has been extended, incorporating quantum-trained SVM models for binary and multiclass classification. The results have shown the efficacy of the approach, with the local quantum-trained methods achieving performance close to their classical counterparts and always better than their global counterparts. In addition, the scalability of the approach has been classically verified, and, using the multiclass models, its practical applicability in a large-scale scenario has been demonstrated.

On the universal quantum computing side, locality (of data samples in the feature space) has been the leitmotiv. Indeed, first, the application of a quantum locality technique as a preliminary step of a quantum machine learning model has been proposed and empirically evaluated. Specifically, a pipeline consisting of a quantum k-NN and a quantum binary classifier (based on the cosine similarity) has been considered and implemented. The approach has turned out to be effective, with the quantum pipeline consistently outperforming the quantum binary classifier in the experiments despite the high sensitivity of the considered quantum k-NN to probability fluctuations. Some classical baseline methods have achieved better results, but the actual usage of the second model in the pipeline was very low for the real-world datasets taken into account. Then, a quantum k-NN algorithm based on the Euclidean distance estimation, still missing in the literature and characterised by a really simple quantum circuit, has been introduced and empirically evaluated on similar datasets. In particular, two novel data encoding schemes and two ways of estimating the distances have been presented. The results have demonstrated the correctness of the formulation and the algorithm's competitiveness with respect to some classical baseline methods in the ideal case. However, a significant drop in performance when using a limited number of measurements has also been observed.

Future directions of research include evaluating the aforementioned quantum annealing algorithms using more performing annealing parameters. Additionally, the divide et impera approach for BNSL could be assessed on larger problems, considering different strategies for the subproblems generation. Concerning the local quantum-trained SVMs, datasets from different domains and different model parameters could be taken into account. Moreover, a local version of the quantum-trained support vector regression model, not considered here, could be developed. Regarding the algorithms for universal quantum devices, they could be assessed on real machines, considering more complex datasets and a higher number of measurements. Additionally, more complex models like the quantum SVM could be taken into account for the locality-based quantum machine learning pipelines. Lastly, the proposed quantum k-NN could be slightly modified in order to obtain a quantum binary classifier based on the Euclidean distance.

To conclude, most of the proposed hybrid approaches have turned out to be effective. However, in order to be competitive against fully classical methods, less noisy quantum architectures are required. Additionally, the physical realization of a QRAM would also be of great importance for universal quantum computing. Hence, these approaches are currently limited by the available hardware. Nevertheless, it is fundamental to carry on the algorithmic research so that when more mature hardware is available, the algorithms will be ready.

Bibliography

[1] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. “The power of quantum neural networks”. In: *Nature Computational Science* 1.6 (June 2021), pp. 403–409. ISSN: 2662-8457. DOI: 10.1038/s43588-021-00084-1. URL: https://doi.org/10.1038/s43588-021-00084-1.

[2] A. Afham, Afrad Basheer, and Sandeep K. Goyal. *Quantum k-nearest neighbor machine learning algorithm*. 2020. URL: https://arxiv.org/abs/2003.09187v1.

[3] Md S. Anis, Héctor Abraham, AduOffei, Rochisha Agarwal, Gabriele Agliardi, Merav Aharoni, et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[4] Ramin Ayanzadeh, Milton Halem, and Tim Finin. “Reinforcement Quantum Annealing: A Hybrid Quantum Learning Automata”. In: *Scientific Reports* 10.1 (May 2020), p. 7952. ISSN: 2045-2322. DOI: 10.1038/s41598-020-64078-1. URL: https://doi.org/10.1038/s41598-020-64078-1.

[5] Afrad Basheer, A. Afham, and Sandeep K. Goyal. *Quantum k-nearest neighbors algorithm*. 2021. URL: https://arxiv.org/abs/2003.09187.

[6] Alina Beygelzimer, Sham Kakade, and John Langford. “Cover Trees for Nearest Neighbor”. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML ’06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 97–104. ISBN: 1595933832. DOI: 10.1145/1143844.1143857. URL: https://doi.org/10.1145/1143844.1143857.

[7] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. “Quantum machine learning”. In: *Nature* 549.7671 (Sept. 2017), pp. 195–202. ISSN: 1476-4687. DOI: 10.1038/nature23474. URL: https://doi.org/10.1038/nature23474.

[8] Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. “Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results”. In: *Information and Computation* 275 (2020), p. 104609. ISSN: 0890-5401. DOI: https://doi.org/10.1016/j.ic.2020.104609. URL: https://www.sciencedirect.com/science/article/pii/S0890540120300973.

[9] Enrico Blanzieri and Farid Melgani. “An Adaptive SVM Nearest Neighbor Classifier for Remotely Sensed Imagery”. In: *2006 IEEE International Symposium on Geoscience and Remote Sensing*. 2006, pp. 3931–3934. DOI: 10.1109/IGARSS.2006.1008.

[10] Denis Bokhan, Alena S. Mastiukova, Aleksey S. Boev, Dmitrii N. Trubnikov, and Aleksey K. Fedorov. “Multiclass classification using quantum convolutional neural networks with hybrid quantum-classical learning”. In: *Frontiers in Physics* 10 (2022). ISSN: 2296-424X. DOI: 10.3389/fphy.2022.1069985. URL: https://www.frontiersin.org/articles/10.3389/fphy.2022.1069985.

[11] Andrea Bonomi. *Python Quantum Annealing Learning Search*. https://github.com/bonom/Quantum-Annealing-for-solving-QUBO-Problems. 2021.

[12] Andrea Bonomi, Thomas De Min, Enrico Zardini, Enrico Blanzieri, Valter Cavecchia, and Davide Pastorello. “Quantum annealing learning search implementations”. In: *Quantum Information and Computation* 22.3&4 (Feb. 2022), pp. 181–208. DOI: 10.26421/qic22.3-4-1. URL: https://doi.org/10.26421%2Fqic22.3-4-1.

[13] Sima E. Borujeni, Saideep Nannapaneni, Nam H. Nguyen, Elizabeth C. Behrman, and James E. Steck. “Quantum circuit representation of Bayesian networks”. In: *Expert Systems with Applications* 176 (2021), p. 114768. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.114768.

[14] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. “Quantum amplitude amplification and estimation”. In: *Contemporary Mathematics* 305 (2002), pp. 53–74.

[15] Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. “Quantum Fingerprinting”. In: *Phys. Rev. Lett.* 87 (16 Sept. 2001), p. 167902. DOI: 10.1103/PhysRevLett.87.167902. URL: https://link.aps.org/doi/10.1103/PhysRevLett.87.167902.

[16] Gabriele Cavallaro, Dennis Willsch, Madita Willsch, Kristel Michielsen, and Morris Riedel. “Approaching Remote Sensing Image Classification with Ensembles of Support Vector Machines on the D-Wave Quantum Annealer”. In: *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium.* 2020, pp. 1973–1976. DOI: 10.1109/IGARSS39084.2020.9323544.

[17] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM: A library for support vector machines”. In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

[18] David M. Chickering. “Learning Bayesian Networks is NP-Complete”. In: *Learning from Data: Artificial Intelligence and Statistics V.* New York, NY: Springer New York, 1996, pp. 121–130. ISBN: 978-1-4612-2404-4. DOI: 10.1007/978-1-4612-2404-4_12. URL: https://doi.org/10.1007/978-1-4612-2404-4_12.

[19] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. “Quantum algorithms revisited”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1969 (Jan. 1998), pp. 339–354. ISSN: 1471-2946. DOI: 10.1098/rspa.1998.0164. URL: http://dx.doi.org/10.1098/rspa.1998.0164.

[20] Elon S. Correa, Alex A. Freitas, and Colin G. Johnson. “Particle Swarm and Bayesian Networks Applied to Attribute Selection for Protein Functional Classification”. In: *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation.* GECCO ’07. London, United Kingdom: Association for Computing Machinery, 2007, pp. 2651–2658. ISBN: 9781595936981. DOI: 10.1145/1274000.1274081.

[21] Koby Crammer and Yoram Singer. “On the Algorithmic Implementation of Multiclass Kernel-Based Vector Machines”. In: *Journal of Machine Learning Research* 2 (Mar. 2002), pp. 265–292. ISSN: 1532-4435.

[22] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000. DOI: 10.1017/CBO9780511801389.

[23] Yijie Dang, Nan Jiang, Hao Hu, Zhuoxiao Ji, and Wenyin Zhang. “Image classification based on quantum K-Nearest-Neighbor algorithm”. In: *Quantum Information Processing* 17.9 (Aug. 2018), p. 239. ISSN: 1573-1332. DOI: 10.1007/s11128-018-2004-9. URL: https://doi.org/10.1007/s11128-018-2004-9.

[24] Amer Delilbasic. *QMSVM implementation.* https://gitlab.jsc.fz-juelich.de/sdlrs/qmsvm. Last access on 15 Dec 2023. 2023.

[25] Amer Delilbasic, Gabriele Cavallaro, Madita Willsch, Farid Melgani, Morris Riedel, and Kristel Michielsen. “Quantum Support Vector Machine Algorithms for Remote Sensing Data Classification”. In: *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS.* 2021, pp. 2608–2611. DOI: 10.1109/IGARSS47720.2021.9554802.

[26] Amer Delilbasic, Bertrand Le Saux, Morris Riedel, Kristel Michielsen, and Gabriele Cavallaro. “A Single-Step Multiclass SVM Based on Quantum Annealing for Remote Sensing Data Classification”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* (2023), pp. 1–12. DOI: 10.1109/JSTARS.2023.3336926.

[27] Sebastiano Dissegna. *Implementation of the divide et impera approach.* https://github.com/sebdisdv/BNSL. 2021.

[28] Jun Du. *Embedding Python in C/C++.* https://www.codeproject.com/Articles/11805/Embedding-Python-in-C-C-Part-I. Last access on 24 February 2021. 2005.

[29] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository.* 2017. URL: http://archive.ics.uci.edu/ml.

[30] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. "Quantum-Enhanced Machine Learning". In: *Phys. Rev. Lett.* 117 (13 Sept. 2016), p. 130501. DOI: 10.1103/PhysRevLett.117.130501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.117.130501.

[31] Christoph Dürr and Peter Høyer. *A Quantum Algorithm for Finding the Minimum.* Jan. 1999. URL: https://arxiv.org/abs/quant-ph/9607014.

[32] Dmitriy V. Fastovets, Yurii I. Bogdanov, Boris I. Bantysh, and Vladimir F. Lukichev. "Machine learning methods in quantum computing theory". In: *International Conference on Micro- and Nano-Electronics 2018.* Vol. 11022. International Society for Optics and Photonics. Zvenigorod, Russia: SPIE, 2019, pp. 752–761. URL: https://doi.org/10.1117/12.2522427.

[33] Congcong Feng, Bo Zhao, Xin Zhou, Xiaodong Ding, and Zheng Shan. "An Enhanced Quantum K-Nearest Neighbor Classification Algorithm Based on Polar Distance". In: *Entropy* 25.1 (2023). ISSN: 1099-4300. DOI: 10.3390/e25010127. URL: https://www.mdpi.com/1099-4300/25/1/127.

[34] Evelyn Fix and Joseph L. Hodges. *Discriminatory Analysis, Nonparametric Discrimination: Consistency Properties.* Tech. rep. 4. USAF School of Aviation Medicine, Randolph Field, 1951.

[35] Li-Zhen Gao, Chun-Yue Lu, Gong-De Guo, Xin Zhang, and Song Lin. "Quantum K-nearest neighbors classification algorithm based on Mahalanobis distance". In: *Frontiers in Physics* 10 (2022). ISSN: 2296-424X. DOI: 10.3389/fphy.2022.1047466. URL: https://www.frontiersin.org/articles/10.3389/fphy.2022.1047466.

[36] Amanuel T. Getachew. *Quantum K-medians Algorithm Using Parallel Euclidean Distance Estimator.* 2020. DOI: 10.48550/ARXIV.2012.11139. URL: https://arxiv.org/abs/2012.11139.

[37] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Quantum Random Access Memory". In: *Phys. Rev. Lett.* 100 (16 Apr. 2008), p. 160501. DOI: 10.1103/PhysRevLett.100.160501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.100.160501.

[38] Fred Glover, Gary Kochenberger, and Yu Du. "Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models". In: *4OR - A Quarterly Journal of Operations Research* 17.4 (Dec. 2019), pp. 335–371. ISSN: 1614-2411. DOI: 10.1007/s10288-019-00424-y. URL: https://doi.org/10.1007/s10288-019-00424-y.

[39] William S. Gosset. "The Probable Error of a Mean". In: *Biometrika* 6.1 (Mar. 1908). Originally published under the pseudonym "Student", pp. 1–25. URL: http://dx.doi.org/10.2307/2331554.

[40] Derek Greene, Derek O'Callaghan, and Pádraig Cunningham. "How Many Topics? Stability Analysis for Topic Models". In: *Machine Learning and Knowledge Discovery in Databases.* Ed. by Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 498–513. ISBN: 978-3-662-44848-9.

[41] Lov K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing.* STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[42] Robert Hable. "Universal Consistency of Localized Versions of Regularized Kernel Methods". In: *Journal of Machine Learning Research* 14.5 (2013), pp. 153–186. URL: http://jmlr.org/papers/v14/hable13a.html.

[43] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, et al. “Supervised learning with quantum-enhanced feature spaces”. In: *Nature* 567.7747 (Mar. 2019), pp. 209–212. ISSN: 1476-4687. DOI: 10.1038/s41586-019-0980-2. URL: https://doi.org/10.1038/s41586-019-0980-2.

[44] David Heckerman, Dan Geiger, and David M. Chickering. “Learning Bayesian Networks: The Combination of Knowledge and Statistical Data”. In: *Machine Learning* 20.3 (Sept. 1995), pp. 197–243. ISSN: 1573-0565. DOI: 10.1023/A:1022623210503.

[45] IBM. *International Business Machines Corporation.* https://www.ibm.com/quantum-computing. Last access on 29 Apr 2022. 2022.

[46] Anaconda Inc. et al. *Numba, compiling Python code with @jit.* https://numba.readthedocs.io/en/stable/user/jit.html. Last access on 12 August 2021.

[47] D-Wave Systems Inc. *D-Wave simulated annealing sampler.* https://docs.ocean.dwavesys.com/projects/neal/en/latest/reference/sampler.html. Last access on 29 November 2021.

[48] D-Wave Systems Inc. *D-Wave solver parameters.* https://docs.dwavesys.com/docs/latest/c_solver_parameters.html. Last access on 20 October 2021. 2021.

[49] D-Wave Systems Inc. *D-Wave Systems.* https://www.dwavesys.com. Last access on 29 Apr 2022. 2022.

[50] D-Wave Systems Inc. *Embedding Composite.* https://docs.ocean.dwavesys.com/en/stable/docs_system/reference/composites.html#embeddingcomposite. Last access on 20 October 2021. 2021.

[51] D-Wave Systems Inc. *Leap’s Hybrid Solvers.* https://docs.dwavesys.com/docs/latest/doc_leap_hybrid.html#doc-leap-hybrid. Last access on 25 October 2023. 2023.

[52] D-Wave Systems Inc. *Minor embedding.* https://docs.dwavesys.com/docs/latest/c_gs_3.html#minor-embedding. Last access on 20 October 2021. 2021.

[53] D-Wave Systems Inc. *Minor embedding example.* https://docs.dwavesys.com/docs/latest/c_gs_7.html#getting-started-embedding. Last access on 20 October 2021. 2021.

[54] D-Wave Systems Inc. *Operation and Timing.* https://docs.dwavesys.com/docs/latest/c_qpu_timing.html. Last access on 13 December 2021.

[55] D-Wave Systems Inc. *Pegasus topology.* https://docs.dwavesys.com/docs/latest/c_gs_4.html#pegasus-graph. Last access on 24 February 2021. 2021.

[56] Ridgeback Network Defense Inc. *D-Wave random coin flip.* https://github.com/ridgebacknet/dwave-tutorials/blob/master/fun/fun-coin.py. Last access on 26 February 2021. 2018.

[57] ISPRS. *2D Semantic Labeling Contest - Potsdam.* https://www.isprs.org/education/benchmarks/UrbanSemLab/2d-sem-label-potsdam.aspx. Last access on 19 Dec 2023.

[58] Thorsten Joachims. *SVM-Multiclass: Multi-Class Support Vector Machine.* https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html. Last access on 15 Dec 2023. 2008.

[59] Tadashi Kadowaki and Hidetoshi Nishimori. “Quantum annealing in the transverse Ising model”. In: *Phys. Rev. E* 58 (5 Nov. 1998), pp. 5355–5363. DOI: 10.1103/PhysRevE.58.5355. URL: https://link.aps.org/doi/10.1103/PhysRevE.58.5355.

[60] Narenda Karmarkar and Richard M. Karp. *The Differencing Method of Set Partitioning.* Tech. rep. UCB/CSD-83-113. EECS Department, University of California, Berkeley, 1983. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1983/6353.html.

[61] Phillip Kaye. *Reversible addition circuit using one ancillary bit with application to quantum computing.* 2004. DOI: 10.48550/ARXIV.QUANT-PH/0408173. URL: https://arxiv.org/abs/quant-ph/0408173.

[62] Scott Kirkpatrick, Charles D. Gelatt Jr., and Mario P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[63] Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, et al. “The unconstrained binary quadratic programming problem: a survey”. In: *Journal of Combinatorial Optimization* 28.1 (July 2014), pp. 58–81. ISSN: 1573-2886. DOI: 10.1007/s10878-014-9734-0. URL: https://doi.org/10.1007/s10878-014-9734-0.

[64] Richard E. Korf. “A complete anytime algorithm for number partitioning”. In: *Artificial Intelligence* 106.2 (1998), pp. 181–203. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(98)00086-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370298000861.

[65] Jing Li, Song Lin, Kai Yu, and Gongde Guo. “Quantum K-nearest neighbor classification algorithm based on Hamming distance”. In: *Quantum Information Processing* 21.1 (Dec. 2021), p. 18. ISSN: 1573-1332. DOI: 10.1007/s11128-021-03361-0. URL: https://doi.org/10.1007/s11128-021-03361-0.

[66] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. *Quantum algorithms for supervised and unsupervised machine learning.* 2013. DOI: 10.48550/ARXIV.1307.0411. URL: https://arxiv.org/abs/1307.0411.

[67] Andrew Lucas. “Ising formulations of many NP problems”. In: *Frontiers in Physics* 2 (2014). ISSN: 2296-424X. DOI: 10.3389/fphy.2014.00005. URL: https://www.frontiersin.org/articles/10.3389/fphy.2014.00005.

[68] Yan-zhu Ma, Hong-fei Song, and Jun Zhang. “Quantum Algorithm for K-Nearest Neighbors Classification Based on the Categorical Tensor Network States”. In: *International Journal of Theoretical Physics* 60.3 (Mar. 2021), pp. 1164–1174. ISSN: 1572-9575. DOI: 10.1007/s10773-021-04742-y. URL: https://doi.org/10.1007/s10773-021-04742-y.

[69] Jarrod R. McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. “The theory of variational hybrid quantum-classical algorithms”. In: *New Journal of Physics* 18.2 (Feb. 2016), p. 023023. ISSN: 1367-2630. DOI: 10.1088/1367-2630/18/2/023023. URL: http://dx.doi.org/10.1088/1367-2630/18/2/023023.

[70] Mona Meister and Ingo Steinwart. “Optimal Learning Rates for Localized SVMs”. In: *Journal of Machine Learning Research* 17.194 (2016), pp. 1–44. URL: http://jmlr.org/papers/v17/14-023.html.

[71] Thomas De Min. *C++ Quantum Annealing Learning Search.* https://github.com/tdemin16/QALS-cpp. 2021.

[72] Kosuke Mitarai, Masahiro Kitagawa, and Keisuke Fujii. “Quantum analog-digital conversion”. In: *Phys. Rev. A* 99 (1 Jan. 2019), p. 012301. DOI: 10.1103/PhysRevA.99.012301. URL: https://link.aps.org/doi/10.1103/PhysRevA.99.012301.

[73] Kohei Miyamoto, Masakazu Iwamura, and Koichi Kise. *A Quantum Algorithm for Finding k-Minima.* 2019. DOI: 10.48550/ARXIV.1907.03315. URL: https://arxiv.org/abs/1907.03315.

[74] Rajdeep K. Nath, Himanshu Thapliyal, and Travis S. Humble. “A Review of Machine Learning Classification Using Quantum Annealing for Real-World Applications”. In: *SN COMPUT. SCI* 365 (2 July 2021). URL: https://doi.org/10.1007/s42979-021-00751-0.

[75] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[76] Bryan O’Gorman, Ryan Babbush, Alejandro Perdomo-Ortiz, Alan Aspuru-Guzik, and Vadim Smelyanskiy. “Bayesian network structure learning using quantum annealing”. In: *The European Physical Journal Special Topics* 224.1 (Feb. 2015), pp. 163–188. ISSN: 1951-6401. DOI: 10.1140/epjst/e2015-02349-9. URL: https://doi.org/10.1140/epjst/e2015-02349-9.

[77] Maris Ozols, Martin Roetteler, and Jérémie Roland. "Quantum Rejection Sampling". In: *ACM Transactions on Computation Theory (TOCT)* 5.3 (Aug. 2013). ISSN: 1942-3454. DOI: 10.1145/2493252.2493256.

[78] Edoardo Pasetto, Amer Delilbasic, Gabriele Cavallaro, Madita Willsch, Farid Melgani, Morris Riedel, et al. "Quantum Support Vector Regression for Biophysical Variable Estimation in Remote Sensing". In: *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*. 2022, pp. 4903–4906. DOI: 10.1109/IGARSS46834.2022.9883963.

[79] Davide Pastorello and Enrico Blanzieri. "A Quantum Binary Classifier based on Cosine Similarity". In: *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2021, pp. 477–478. DOI: 10.1109/QCE52317.2021.00086.

[80] Davide Pastorello and Enrico Blanzieri. *A quantum binary classifier based on cosine similarity*. https://arxiv.org/abs/2104.02975. 2021. DOI: 10.48550/ARXIV.2104.02975.

[81] Davide Pastorello and Enrico Blanzieri. "Quantum annealing learning search for solving QUBO problems". In: *Quantum Information Processing* 18.10 (Aug. 2019), p. 303. ISSN: 1573-1332. DOI: 10.1007/s11128-019-2418-z. URL: https://doi.org/10.1007/s11128-019-2418-z.

[82] Davide Pastorello, Enrico Blanzieri, and Valter Cavecchia. "Learning adiabatic quantum algorithms over optimization problems". In: *Quantum Machine Intelligence* 3.1 (Jan. 2021), p. 2. ISSN: 2524-4914. DOI: 10.1007/s42484-020-00030-w. URL: https://doi.org/10.1007/s42484-020-00030-w.

[83] Miguel Patrício, José Pereira, Joana Crisóstomo, Paulo Matafome, Manuel Gomes, Raquel Seiça, et al. "Using Resistin, glucose, age and BMI to predict the presence of breast cancer". In: *BMC Cancer* 18.1 (Jan. 2018), p. 29. ISSN: 1471-2407. DOI: 10.1186/s12885-017-3877-1. URL: https://doi.org/10.1186/s12885-017-3877-1.

[84] Judea Pearl. "Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning". In: *Proceedings of the 7th conference of the Cognitive Science Society (CSS-7)*. 1985, pp. 15–17.

[85] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, O. Grisel, et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[86] João P. Pedroso and Mikio Kubo. "Heuristics and exact methods for number partitioning". In: *European Journal of Operational Research* 202.1 (2010), pp. 73–81. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2009.04.027. URL: https://www.sciencedirect.com/science/article/pii/S0377221709003002.

[87] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (2018), p. 79.

[88] Luis F. Quezada, Guo-Hua Sun, and Shi-Hai Dong. "Quantum Version of the k-NN Classifier Based on a Quantum Sorting Algorithm". In: *Annalen der Physik* 534.5 (2022), p. 2100449. DOI: https://doi.org/10.1002/andp.202100449. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.202100449.

[89] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. "Quantum Support Vector Machine for Big Data Classification". In: *Phys. Rev. Lett.* 113 (13 Sept. 2014), p. 130503. DOI: 10.1103/PhysRevLett.113.130503. URL: https://link.aps.org/doi/10.1103/PhysRevLett.113.130503.

[90] Patrick Rebentrost, Adrian Steffens, Iman Marvian, and Seth Lloyd. "Quantum singular-value decomposition of nonsparse low-rank matrices". In: *Phys. Rev. A* 97 (1 Jan. 2018), p. 012327. DOI: 10.1103/PhysRevA.97.012327. URL: https://link.aps.org/doi/10.1103/PhysRevA.97.012327.

[91] Rigetti. *Rigetti Computing*. https://www.rigetti.com/what-we-build. Last access on 29 Apr 2022. 2022.

[92] Massimo Rizzoli. *Implementation of O'Gorman's algorithm.* https://github.com/massimo-rizzoli/BNSL-QA-python. 2021.

[93] Ribana Roscher, Michele Volpi, Clément Mallet, Lukas Drees, and Jan D. Wegner. "SEM-CITY TOULOUSE: A BENCHMARK FOR BUILDING INSTANCE SEGMENTATION IN SATELLITE IMAGES". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-5-2020 (2020), pp. 109–116. DOI: 10.5194/isprs-annals-V-5-2020-109-2020. URL: https://isprs-annals.copernicus.org/articles/V-5-2020/109/2020/.

[94] Matthew Route. "Radio-flaring Ultracool Dwarf Population Synthesis". In: *The Astrophysical Journal* 845.1 (Aug. 2017), p. 66. DOI: 10.3847/1538-4357/aa7ede. URL: https://dx.doi.org/10.3847/1538-4357/aa7ede.

[95] Yue Ruan, Xiling Xue, Heng Liu, Jianing Tan, and Xi Li. "Quantum Algorithm for K-Nearest Neighbors Classification Based on the Metric of Hamming Distance". In: *International Journal of Theoretical Physics* 56.11 (Nov. 2017), pp. 3496–3507. ISSN: 1572-9575. DOI: 10.1007/s10773-017-3514-4. URL: https://doi.org/10.1007/s10773-017-3514-4.

[96] Abhijat Sarma, Rupak Chatterjee, Kaitlin Gili, and Ting Yu. "Quantum unsupervised and supervised learning on superconducting processors". In: *Quantum Information and Computation* 20.7–8 (2020), pp. 541–552. ISSN: 1533-7146. DOI: 10.26421/QIC20.7-8-1.

[97] Bernhard Schölkopf and Alexander J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2002.

[98] Maria Schuld, Mark Fingerhuth, and Francesco Petruccione. "Implementing a distance-based classifier with a quantum interference circuit". In: *Europhysics Letters* 119.6 (Dec. 2017), p. 60002. DOI: 10.1209/0295-5075/119/60002. URL: https://dx.doi.org/10.1209/0295-5075/119/60002.

[99] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. "Quantum Computing for Pattern Classification". In: *PRICAI 2014: Trends in Artificial Intelligence.* Ed. by Duc-Nghia Pham and Seong-Bae Park. Cham: Springer International Publishing, 2014, pp. 208–220.

[100] Ralf Schützhold. "Pattern recognition on a quantum computer". In: *Phys. Rev. A* 67 (6 June 2003), p. 062311. DOI: 10.1103/PhysRevA.67.062311. URL: https://link.aps.org/doi/10.1103/PhysRevA.67.062311.

[101] Scikit-learn. *Balanced accuracy.* https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html. Last access on 21 Dec 2023.

[102] Scikit-learn. *F1 score.* https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. Last access on 21 Dec 2023.

[103] Nicola Segata. *FaLKM-lib v1.0: a Library for Fast Local Kernel Machines.* Tech. rep. DISI-09-025. Software available at http://disi.unitn.it/~segata/FaLKM-lib. DISI, University of Trento, Italy, 2009.

[104] Nicola Segata and Enrico Blanzieri. "Fast and Scalable Local Kernel Machines". In: *Journal of Machine Learning Research* 11.64 (2010), pp. 1883–1926. URL: http://jmlr.org/papers/v11/segata10a.html.

[105] Nicola Segata, Edoardo Pasolli, Farid Melgani, and Enrico Blanzieri. "Local SVM approaches for fast and accurate classification of remote-sensing images". In: *International Journal of Remote Sensing* 33.19 (2012), pp. 6186–6201. DOI: 10.1080/01431161.2012.678947. URL: https://doi.org/10.1080/01431161.2012.678947.

[106] Bayes Server. *Bayesian network examples.* https://www.bayesserver.com/examples. Last access on 22 June 2021.

[107] Peter W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science.* 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[108] Pedro F. B. Silva, André R. S. Marçal, and Rubim M. Almeida da Silva. "Evaluation of Features for Leaf Discrimination". In: *Springer Lecture Notes in Computer Science* 7950 (2013), pp. 197–204.

[109] Peter Spirtes and Christopher Meek. "Learning Bayesian Networks with Discrete Variables from Data". In: *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. KDD'95. AAAI Press, 1995, pp. 294–299.

[110] M. Stęchły and P. Korponaić. *Quantum TSP*. `https://github.com/BOHRTECHNOLOGY/quantum_tsp`. Last access on 27 June 2021. 2018.

[111] Carlo A. Trugenberger. "Quantum Pattern Recognition". In: *Quantum Information Processing* 1.6 (Dec. 2002), pp. 471–493. ISSN: 1573-1332. DOI: `10.1023/A:1024022632303`. URL: `https://doi.org/10.1023/A:1024022632303`.

[112] user1413793. *C++ rand()*. `https://stackoverflow.com/questions/10984974/why-do-people-say-there-is-modulo-bias-when-using-a-random-number-generator`. Last access on 24 February 2021. 2016.

[113] Haibo Wang, Wendy Wang, Yi Liu, and Bahram Alidaee. "Integrating Machine Learning Algorithms With Quantum Annealing Solvers for Online Fraud Detection". In: *IEEE Access* 10 (2022), pp. 75908–75917. DOI: `10.1109/ACCESS.2022.3190897`.

[114] Yuxiang Wang, Ruijin Wang, Dongfen Li, Daniel Adu-Gyamfi, Kaibin Tian, and Yixin Zhu. "Improved Handwritten Digit Recognition using Quantum K-Nearest Neighbor Algorithm". In: *International Journal of Theoretical Physics* 58.7 (July 2019), pp. 2331–2340. ISSN: 1572-9575. DOI: `10.1007/s10773-019-04124-5`. URL: `https://doi.org/10.1007/s10773-019-04124-5`.

[115] Nathan Wiebe, Ashish Kapoor, and Krysta M. Svore. "Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning". In: *Quantum Information and Computation* 15.3–4 (Mar. 2015), pp. 316–356. ISSN: 1533-7146. DOI: `10.26421/QIC15.3-4-7`.

[116] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83. ISSN: 00994987. URL: `http://www.jstor.org/stable/3001968`.

[117] Dennis Willsch, Gabriele Cavallaro, and Madita Willsch. *QA SVM implementation*. `https://gitlab.jsc.fz-juelich.de/sdlrs/quantum-svm-algorithms-for-rs-data-classification/-/tree/master/experiments/QA_SVM?ref_type=heads`. Last access on 15 Dec 2023. 2021.

[118] Dennis Willsch, Madita Willsch, Hans De Raedt, and Kristel Michielsen. "Support vector machines on the D-Wave quantum annealer". In: *Computer Physics Communications* 248 (2020), p. 107006. ISSN: 0010-4655. DOI: `https://doi.org/10.1016/j.cpc.2019.107006`. URL: `https://www.sciencedirect.com/science/article/pii/S001046551930342X`.

[119] Edwin B. Wilson. "Probable Inference, the Law of Succession, and Statistical Inference". In: *Journal of the American Statistical Association* 22.158 (1927), pp. 209–212. DOI: `10.1080/01621459.1927.10502953`. URL: `https://www.tandfonline.com/doi/abs/10.1080/01621459.1927.10502953`.

[120] Joanna Wiśniewska and Marek Sawerwain. "Recognizing the pattern of binary Hermitian matrices by quantum kNN and SVM methods". In: *Vietnam Journal of Computer Science* 5.3 (Sept. 2018), pp. 197–204. ISSN: 2196-8896. DOI: `10.1007/s40595-018-0115-y`. URL: `https://doi.org/10.1007/s40595-018-0115-y`.

[121] Jindi Wu, Zeyi Tao, and Qun Li. "wpScalable Quantum Neural Networks for Classification". In: *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2022, pp. 38–48. DOI: `10.1109/QCE53715.2022.00022`.

[122] I-Cheng Yeh, King-Jang Yang, and Tao-Ming Ting. "Knowledge Discovery on RFM Model Using Bernoulli Sequence". In: *Expert Syst. Appl.* 36.3 (Apr. 2009), pp. 5866–5871. DOI: `10.1016/j.eswa.2008.07.018`. URL: `https://doi.org/10.1016/j.eswa.2008.07.018`.

[123] Kai Yu, Gong-De Guo, Jing Li, and Song Lin. "Quantum Algorithms for Similarity Measurement Based on Euclidean Distance". In: *International Journal of Theoretical Physics* 59.10 (Oct. 2020), pp. 3134–3144. ISSN: 1572-9575. DOI: 10.1007/s10773-020-04567-1. URL: https://doi.org/10.1007/s10773-020-04567-1.

[124] Enrico Zardini. *Euclidean Quantum k-NN Data.* 2023. DOI: 10.6084/m9.figshare.22598455.v1. URL: https://figshare.com/articles/dataset/Euclidean_Quantum_k-NN_Data/22598455.

[125] Enrico Zardini. *QML Pipeline Datasets.* 2023. DOI: 10.6084/m9.figshare.22333102.v1. URL: https://figshare.com/articles/dataset/QML_Pipeline_Datasets/22333102.

[126] Enrico Zardini. *QML Pipeline Raw Results.* 2023. DOI: 10.6084/m9.figshare.22333147.v1. URL: https://figshare.com/articles/dataset/QML_Pipeline_Raw_Results/22333147.

[127] Enrico Zardini, Enrico Blanzieri, and Davide Pastorello. *A quantum k-nearest neighbors algorithm based on the Euclidean distance estimation.* Accepted by *Quantum Machine Intelligence.* 2023. arXiv: 2305.04287 [cs.ET].

[128] Enrico Zardini, Enrico Blanzieri, and Davide Pastorello. "Implementation and empirical evaluation of a quantum machine learning pipeline for local classification". In: *PLOS ONE* 18.11 (Nov. 2023), pp. 1–28. DOI: 10.1371/journal.pone.0287869. URL: https://doi.org/10.1371/journal.pone.0287869.

[129] Enrico Zardini, Amer Delilbasic, Enrico Blanzieri, Gabriele Cavallaro, and Davide Pastorello. *Local Binary and Multiclass SVMs Trained on a Quantum Annealer.* 2024. arXiv: 2403.08584 [cs.ET].

[130] Enrico Zardini, Massimo Rizzoli, Sebastiano Dissegna, Enrico Blanzieri, and Davide Pastorello. "Reconstructing Bayesian networks on a quantum annealer". In: *Quantum Information and Computation* 22.15&16 (Nov. 2022), pp. 1320–1350. DOI: 10.26421/qic22.15-16-4. URL: https://doi.org/10.26421%2Fqic22.15-16-4.

[131] Nan-Run Zhou, Xiu-Xun Liu, Yu-Ling Chen, and Ni-Suo Du. "Quantum K-Nearest-Neighbor Image Classification Algorithm Based on K-L Transform". In: *International Journal of Theoretical Physics* 60.3 (Mar. 2021), pp. 1209–1224. ISSN: 1572-9575. DOI: 10.1007/s10773-021-04747-7. URL: https://doi.org/10.1007/s10773-021-04747-7.