EasyChair Preprint

№ 12862

Hybrid Quantum-Classical Framework for Clustering: a Comprehensive Approach with QMeans

Antonin Lefevre

April 1, 2024

Chapter 1

QMeans Algorithm

Abstract:

The Kmeans algorithm is a cornerstone in unsupervised learning for clustering, with a temporal complexity of $O(iknm)$, where i represents the number of iterations, k the number of clusters, n the number of points, and m the dimensionality of the observation space. The quantum-inspired variant, QMeans [1] , was introduced to address these limitations, albeit primarily on a theoretical front. This chapter bridges this gap by elucidating and implementing QMeans within a hybrid quantum-classical framework. Initially, a comprehensive overview of Kmeans and δ-Kmeans clustering models is provided. Subsequently, the paper covers quantum distance computation, quantum minimum finding in a list, and the quantum version of the Kmeans++ initialization method [10] , QMeans++, along with their respective mathematical formulations, circuit designs, and implementations with Qiskit. Finally, these elements are assembled to formulate the QMeans algorithm.

Keywords: Clustering; Kmeans; Quantum computing.

1.1 Introduction

Quantum computing emerges as a revolutionary paradigm, extending the frontiers of computational science by harnessing the principles of quantum mechanics. Unlike classical computing, which relies on bits as the smallest unit of information, quantum computing utilizes qubits. These qubits exhibit unique properties such as superposition - allowing them to represent multiple states simultaneously - and entanglement, which enables them to maintain a connection regardless of distance. This fundamental shift in information

processing opens new avenues for tackling computational problems that are intractable for classical computers, by performing complex calculations at significantly accelerated rates.

Among the practical applications of quantum computing principles is the domain of machine learning, where the QMeans algorithm exemplifies the fusion of classical methodologies with quantum enhancements. Drawing inspiration from the classical KMeans, QMeans incorporates quantum subroutines for specific tasks such as distance estimation and minimum value search, enabling the determination of new centroids in quantum states. Furthermore, QMeans introduces a quantum-centric centroid initialization method, QMeans++, paralleling the effectiveness of Kmeans++ in classical settings but executed in quantum superposition.

This article navigates the complexities of implementing QMeans in the context of current quantum computing limitations, such as the absence of QRAM [8] and the qubit availability on contemporary quantum machines. Through this exploration, we aim to illuminate the potential of quantum computing to revolutionize data analysis and machine learning by leveraging its inherent advantages for optimizing and accelerating computational tasks.

1.2 Related work

This article introduces a hybrid implementation of the QMeans algorithm for unsupervised clustering, building on the foundations established by Kerenidis et al. [1] in their proposition of a quantum version of the classic KMeans algorithm. Their innovative approach demonstrates an exponential speedup in the number of data points compared to the classical KMeans algorithm, offering a new paradigm in applying machine learning algorithms to increasing data volumes.

The necessity for a hybrid approach stems from the practical constraint related to the use of QRAM, which remains theoretical to this day. While QRAM allows for efficient quantum access to classical data, our hybrid approach seeks to circumvent this limitation by combining the computational advantages of quantum systems with the robustness and availability of classical technologies.

It is worth noting that quantum clustering algorithms have been explored prior to Kerenidis et al. [1] , as highlighted by Lloyd et al. [11] who proposed algorithms for supervised and unsupervised learning using quantum states. However, these earlier works faced the challenge of retrieving a classical model from quantum computations, a limitation our method aims to overcome.

Furthermore, research in quantum and quantum-inspired algorithms for machine learning, such as the works by Arrazola et al. [12] and Tang [13] [14] , has underscored the importance of developing techniques that can be efficiently applied on quantum computers or simulated with advantages on classical systems. These investigations lay the groundwork for broader applications of quantum technologies in the field of machine learning.

1.3 Introduction to Kmeans

The Kmeans algorithm solves an optimization problem by attempting to minimize the objective function, often referred to as *inertia* or cost *function*, defined as:

$$J(C, c_1, c_2, \dots, c_k) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x\text{-}c_i\|^2$$

Where C is the set of k clusters $C_1, C_2, \dots, C_k$ and c_i is the centroid of cluster C_i.

Step 1. **Initialization of centroids:** Several initialization methods exist, ranging from the random initiation of centroids to more sophisticated techniques like Kmeans++ [10] .

Step 2. Assignment of points to clusters: The Euclidean distance $\|x\text{-}c_i\|$ is commonly used, although other metrics such as Manhattan distance can also be utilized. The point x is assigned to the cluster that minimizes this distance, thus seeking:

$$\arg \min_{c_i \in C} \|x\text{-}c_i\|^2$$

Step 3. Updating centroids: Each new centroid is the centroid of all points in the corresponding cluster. Formally:

$$c_i = \frac{1}{\#C_i} \sum_{x \in C_i} x$$

With $\#C_i$ being the number of data points in cluster C_i.

Step 4. Convergence: The algorithm iterates steps 2 and 3 until the variation in the positions of centroids is below a predefined threshold ε, or until a maximum number of iterations is reached.

1.4 Introduction to δ-Kmeans

The δ-KMeans is a variant of the KMeans algorithm that introduces a noise parameter δ to the steps of label assignment and centroid updating. This method proves particularly beneficial in scenarios involving noisy data or outliers. The QMeans algorithm serves as the quantum analogue to this approach, leveraging the non-deterministic and noisy nature of quantum computations. This enables QMeans to embody the robustness features of δ-KMeans while harnessing the computational advantages provided by quantum physics.

Here are the details of the two steps that differentiate it from KMeans.

Label Assignment: Unlike KMeans, the assignment is not deterministic. A set of potential labels $\mathrm{L}_\delta(x)$ is created:

$$\mathrm{L}_\delta(x) = p: \left\|\mathrm{c_i^*}\text{-x}\right\|^2 - \left\|\mathrm{c_p}\text{-x}\right\|^2 \leq \delta$$

With $\mathrm{c_i^*}$ being the closest centroid to the data point x. A label is randomly chosen from this set.

To clarify, $\mathrm{L}_\delta(x)$ encompasses all centroid indices p for which the squared distance between point x and centroid c_p is not significantly different from the squared distance between x and the nearest centroid $\mathrm{c_i^*}$. The δ quantity represents the tolerance allowed for this difference.

The idea is that if a point x is almost equidistant from two or more centroids, it could reasonably belong to any cluster associated with these centroids. The δ parameter introduces flexibility in assigning points to clusters, making the algorithm more robust to data variations and especially useful in scenarios where clusters are not distinctly separated.

Centroid Update: Gaussian noise with variance $\frac{\delta}{2}$ is added to the centroid during its update:

$$c_i = \frac{1}{\#\mathrm{C_i}} \sum_{\mathrm{x} \in \mathrm{C_i}} \mathrm{x} + \mathrm{N}\left(0, \frac{\delta}{2}\right)$$

This introduction of noise can help uncover subtle structures in the data that standard KMeans might overlook, providing a more nuanced approach to clustering in complex datasets.

1.5 Quantum subroutines

1.5.1 Quantum Estimation of the Euclidean Distance Between Points

On a classical computer, Euclidean distances can be directly calculated. However, on a quantum computer, this task is complicated due to the probabilistic nature of qubits. Despite this complexity, it is essential to have a way to estimate the distances between points. For this, one can use the dot product, as introduced by Stephen DiAdamo et al. [2] and others, as an indicator of their distance.

The principle is based on the fact that we do not need to know the exact distance between the points, but rather to have an estimation that can allow for the comparison of several distance values. Thus, the dot product, although not proportional to the distance, is positively correlated with it, which allows for efficiently determining which point is closest to another given point.

5.1.1 Relationship between the dot product and distance

Imagine we have an auxiliary qubit initialized to the state $|0\rangle$ and a quantum state $|\psi\rangle = |x\rangle|y\rangle$ representing the two normalized vectors whose distance we want to estimate. Here are the steps to highlight the relationship between the probability of measuring the auxiliary qubit in the state $|1\rangle$, denoted as $P(1_{aux})$, and the dot product of the vectors:

Definition of | ψ⟩ and application of the Hadamard gate: Begin with an auxiliary qubit initialized to $|0\rangle$ and a quantum state $|\psi\rangle$, storing the normalized vectors whose distance we aim to estimate:

$$|\psi\rangle = |x\rangle|y\rangle$$

After applying a Hadamard gate to the auxiliary qubit, and using the controlled-SWAP gate (C-SWAP) on the auxiliary qubit, the state becomes:

$$\frac{1}{\sqrt{2}}(|0\rangle|\psi\rangle + |1\rangle F(|\psi\rangle))$$

Where $F(|\psi\rangle) = |y\rangle|x\rangle$.

Manipulation with a second Hadamard gate: Another Hadamard gate is applied to the auxiliary qubit, resulting in:

$$\frac{1}{2}(|0\rangle|(I+F)\psi\rangle + |1\rangle|(I-F)\psi\rangle)$$

Where I is the identity operator.

Introducing the probability that the auxiliary qubit is at $|1\rangle$: We can substitute $(I-F)$ and $(I+F)$, yielding:

$$(|0\rangle|A_0\psi\rangle + |1\rangle|A_1\psi\rangle)$$

Where $A_0 = \frac{I+F}{2}$ and $A_1 = \frac{I-F}{2}$

Now, utilizing the concept of quantum expectation value, we can express the probability that the auxiliary qubit is measured in the state $|1\rangle$ with an inner product:

$$P(1_{aux}) = \langle\psi|A_1|\psi\rangle$$

Simplifications: Simplifying by substitution with I and F:

$$P(1_{aux}) = \left\langle \text{xy} \left| \frac{I-F}{2} \right| \text{xy} \right\rangle$$

Expanding and knowing that $\langle xy, xy\rangle = 1$ and $\langle xy, yx\rangle = \langle x, y\rangle^2$, we get:

$$P(1_{aux}) = \frac{1}{2} - \frac{1}{2}\langle x, y\rangle^2$$

Finally, we establish a relationship between the probability of measuring the auxiliary qubit in the state $|1\rangle$ denoted as $P(1_{aux})$ and the dot product of the vectors.

Theorem: The probability of measuring the auxiliary qubit in the state $|1\rangle$ (i.e., $P(1_{aux})$) is positively correlated with the distance between the vectors $|x\rangle$ and $|y\rangle$. This allows for comparing distances without explicitly calculating them.

Proof: Let's demonstrate that the previous calculated probability $P(1_{aux})$ is positively correlated with the Euclidean distance between $|x\rangle$ and $|y\rangle$.

The square of the Euclidean distance between two vectors is defined as:

$$d^2 = \|x - y\|^2$$

Expanding the vector subtraction, since the vectors are normalized, we get:

$$d^2 = \langle x - y, x - y\rangle = 2 - 2\langle x, y\rangle$$

If you look closely, $P(1_{aux})$ and the square of the Euclidean distance have forms that are similar in terms of the inner product $\langle x, y\rangle$. In particular, $P(1_{aux})$ is positively correlated with d^2 because as the distance between the vectors increases (i.e., they become more orthogonal), the inner product $\langle x, y\rangle$ decreases, and consequently, $P(1_{aux})$ increases.
It's important to note that although, we are comparing the square of the Euclidean distance d^2 rather than the Euclidean distance d itself, this poses no issue since the square function is monotonic for positive Euclidean distances. Thus, an increase in d^2 always indicates an increase in d, and vice versa.

□

5.1.2 Encoding vector coordinates

To apply the quantum algorithm, it's essential to encode the points' coordinates into qubits.

This is achieved by converting classical rectangular coordinates into spherical coordinates, which are suitable for representation on the Bloch sphere:

- Start with a qubit initialized to $|0\rangle$.

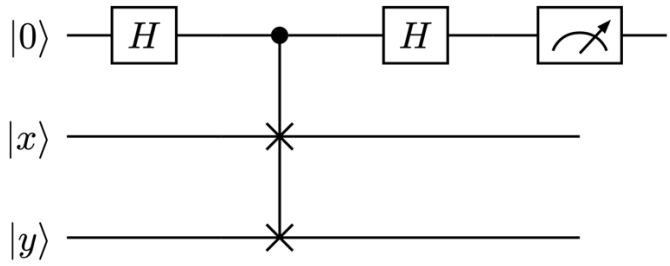
- Apply a Hadamard gate H to orient the qubit along the X axis of the Bloch sphere. (We will use $\lambda = 0$ in a U gate instead of an H gate)
- The angles φ and θ are used to represent the values of the two features of the point. Here, θ and φ can vary between 0 and π radians to avoid representation issues on the Bloch sphere.
- The following relations allow mapping the point's features to the angles θ and φ:

$$\varphi = (d_0 + 1)\frac{\pi}{2} \quad \theta = (d_1 + 1)\frac{\pi}{2}$$

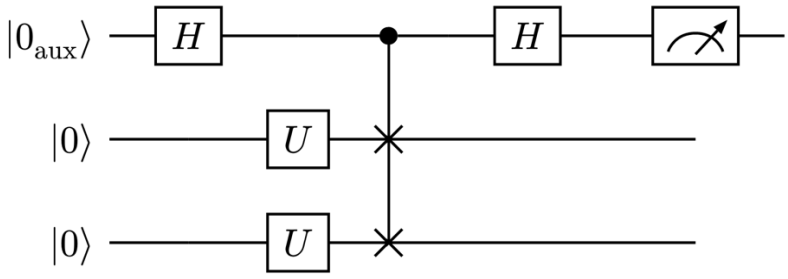
Where d_0 and d_1 correspond to the values of the two features of the point.

5.1.3 Associated quantum circuit

The theoretical quantum circuit corresponding to this distance estimation step is as follows:



The qubit $|0\rangle$ being the auxiliary qubit and the qubits $|x\rangle$ and $|y\rangle$ representing the two vectors whose distance we want to estimate. However, considering the 0, we can detail the circuit a bit more, which gives:



Where $|0_{aux}\rangle$ is the auxiliary qubit, and the two qubits $|x\rangle$ and $|y\rangle$ are created from a qubit $|0\rangle$, a U gate with parameters θ and φ calculated in 0 and λ with value of 0 to align the qubit along the X axis of the Bloch sphere. The measurement on the auxiliary qubit allows calculating the probability of finding $|0_{aux}\rangle$ in the state $|1\rangle$ after n circuits are run.

This U gate is defined by:

$$U(\theta,\varphi,\lambda) = \begin{pmatrix} cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}sin\left(\frac{\theta}{2}\right) \\ e^{i\varphi}sin\left(\frac{\theta}{2}\right) & e^{i(\varphi+\lambda)}cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

We obtain the result of the pseudo distance by measuring the auxiliary qubit, which is the first qubit, and counting the number of times it is measured in the state $|1\rangle$. To now measure the distance between 1 point and n other points, we parallel this operator (thus this circuit) to perform the algorithm on the n points at the same time. Here is the form of the circuit:

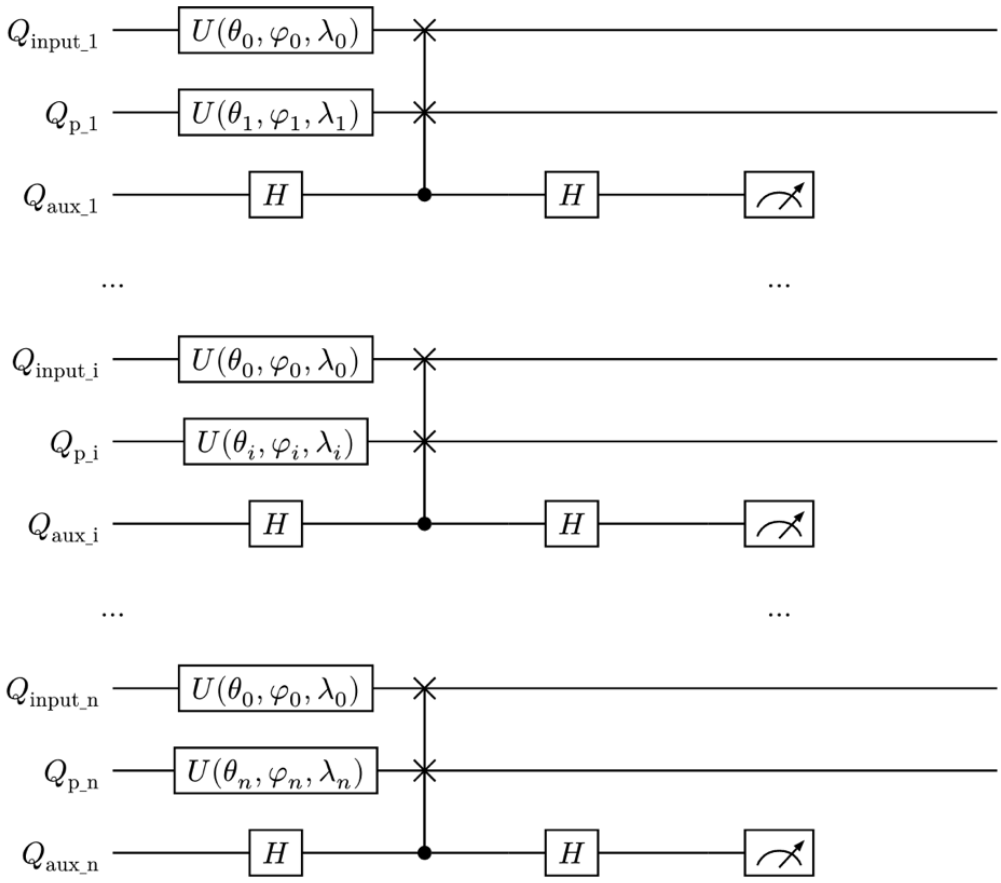


Figure 1. Quantum circuit for calculating distances with n points.

We have n registers of 3 qubits, each containing a qubit that corresponds to the duplication of the point whose distance with the n others needs to be found, a qubit representing one of the n points, and an auxiliary qubit. We can express this circuit as an operator $\widehat{K}_n$ such that:

$$\widehat{K}_n = \bigotimes_{i=0}^{n} \widehat{K}_i$$

Where $\widehat{K_i}$ is defined by:

$$\widehat{K_i} = \left(\widehat{H} \otimes \hat{I} \otimes \hat{I}\right).C - SWAP_{aux_i,(x,p_i)}.\left(\widehat{H} \otimes \widehat{U}(\theta_0, \varphi_0, \lambda_0) \otimes \widehat{U}(\theta_i, \varphi_i, \lambda_i)\right)$$

Where:

- $\theta_0, \varphi_0 \; and \; \lambda_0$ come from the encoding of the first point.
- θ_i, φ_i and λ_i come from the encoding of the i-th point among the n.
- $C - SWAP_{aux_i,(x,p_i)}$ is the SWAP gate between qubit x and p_i (the point and the i-th point) controlled by the qubit aux_i (the i-th auxiliary qubit)

The corresponding circuit is:

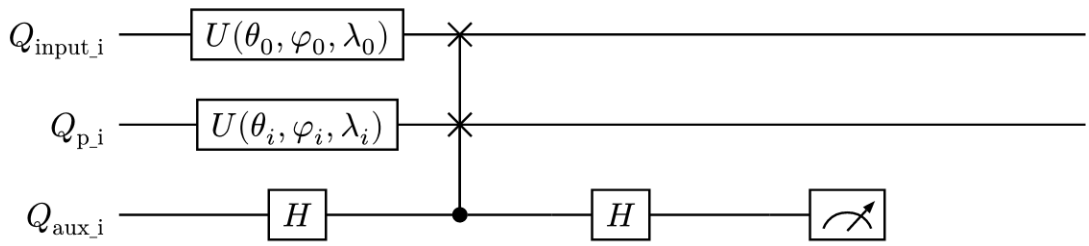


Figure 2. The *i*-th component of the distance calculation circuit.

Calculating distances between 1 data point and n points thus requires $3n$ qubits.

After measuring the n auxiliary qubits, each bit of the final states represents one of the n specific points (involved in the CSWAP gate in the circuit). To interpret the results, we count the occurrences for each bit position where a 1 appears, that is, the bit at index i. Thus, the index of the point associated with a reduced number of occurrences of 1 is considered as the most likely to be the closest to the point in question, due to the way distances are measured in this context.

Indeed, as demonstrated in 5.1.1, the probability of measuring the auxiliary qubit in the state $|1\rangle$ is positively correlated with the distance between the first and the second point. The auxiliary qubit being represented by the bit at position i in the circuit output.

5.1.4 Hand-on example

We consider the point X with coordinates $(x_0, y_0) = (0.1,0.2)$, and the points p_1, p_2 and p_3 with respective coordinates $(x_1, y_1) = (0.1,0.2)$, $(x_2, y_2) = (0.3,0.4)$ and $(x_3, y_3) = (0.5,0.6)$.

We start by encoding these points into qubits, according to 0.

For $X(x_0, y_0) = (0.1,0.2)$:

$$\varphi_X = (x_0 + 1)\frac{\pi}{2} = (0.1 + 1)\frac{\pi}{2} \qquad \theta_X = (y_0 + 1)\frac{\pi}{2} = (0.2 + 1)\frac{\pi}{2}$$

For $p_1(x_1, y_1) = (0.1,0.2)$:

$$\varphi_{C1} = (x_1 + 1)\frac{\pi}{2} = (0.1 + 1)\frac{\pi}{2} \qquad \theta_{C1} = (y_1 + 1)\frac{\pi}{2} = (0.2 + 1)\frac{\pi}{2}$$

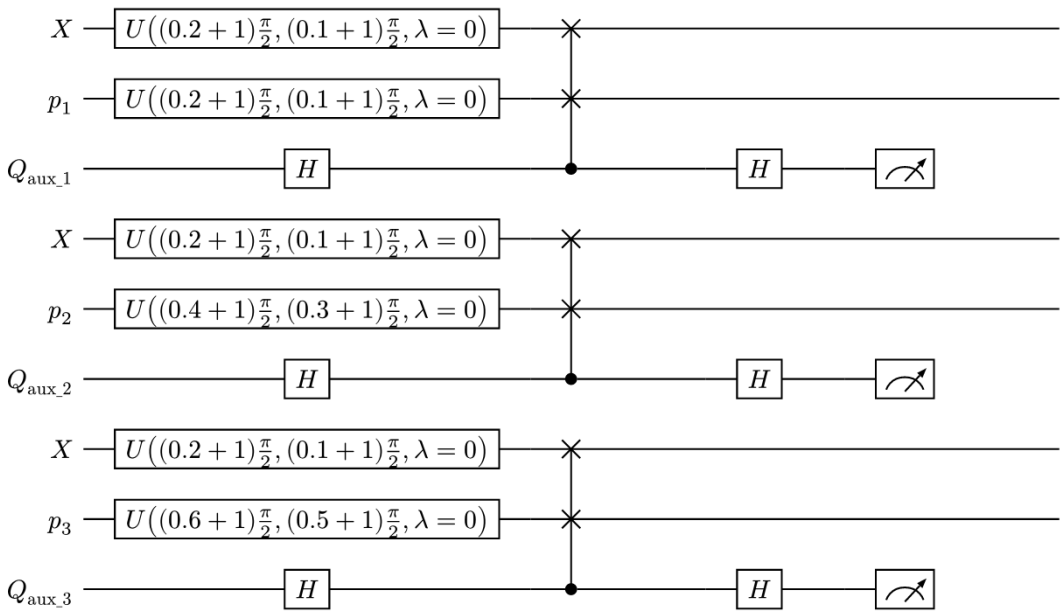
For $p_2(x_2, y_2) = (0.3,0.4)$:

$$\varphi_{C2} = (x_2 + 1)\frac{\pi}{2} = (0.3 + 1)\frac{\pi}{2} \qquad \theta_{C2} = (y_2 + 1)\frac{\pi}{2} = (0.4 + 1)\frac{\pi}{2}$$

For $p_3(x_3, y_3) = (0.5,0.6)$:

$$\varphi_{C3} = (x_3 + 1)\frac{\pi}{2} = (0.5 + 1)\frac{\pi}{2} \qquad \theta_{C3} = (y_3 + 1)\frac{\pi}{2} = (0.6 + 1)\frac{\pi}{2}$$

The resulting circuit is as follows:



Running this circuit 4096 times yields the following frequencies: {‘001’: 322, ‘011’:4, ‘000’:3689, ‘010’: 81}. As previously explained, we count the appearances of 1 for each position. This gives us the frequency table: [0,85,326]. That is, the first auxiliary qubit was measured in state $|1\rangle$ zero times, the second 85 times, and the third 326 times. Since the probability of each auxiliary qubit being in state $|1\rangle$ is proportional to the distance between the two points involved in the SWAP, we then look for the smallest value. Here, 0 is the minimum, which means that the first point p_1 is the closest to point X (which is accurate, looking at the coordinates of the points). This calculation of the minimum for this list is the subject of cluster assignment in the 1.5.3.

1.5.2 Comparing two integers

To compare two integers, we will use the Quantum Bit String Comparator [3] algorithm, introduced in 2007 by D. Oliveira and R. Ramos. We will start by comparing two bits and then extend this to comparing two integers of n bits. Ultimately, we will be able to determine, from two integers a and b, whether $a < b$ or $a \geq b$.

5.2.1 Encoding integers into qubits

To compare two integers using a quantum circuit, it is essential to encode these numbers into qubits. This is done by converting them into their binary representation. Thus, encoding numbers in a quantum circuit requires a number of qubits corresponding to the size of the numbers in question. For instance, one qubit can encode two values (0 and 1), while two qubits can encode four values ($00 = 0$, $01 = 1$, $10 = 2$, $11 = 3$). Ultimately, n qubits allow the encoding for 2^n numbers.

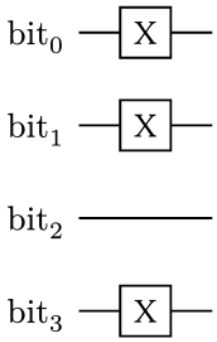


Figure 3: Encoding the integer 13, ‘1101’ in binary.

As we go through the binary representation of the integer, if the bit is 1, a X gate (NOT gate) is applied to the qubit, changing it from the state $|0\rangle$ to the state $|1\rangle$. If the bit is 0, no action is taken, and the qubit remains in the state $|0\rangle$. Thus, a classical bit is translated into a quantum state.

5.2.2 Comparing two integers

Now, the goal is to compare two integers encoded in the quantum circuit using the method described above in 5.2.1. The comparison of the integers is performed bit by bit and uses auxiliary qubits to indicate which bit is greater. The comparison starts with the most significant bits of the two integers. If these bits are equal, the process continues with the next bits. If the bits are different, the bit-by-bit comparison function determines which integer is larger.

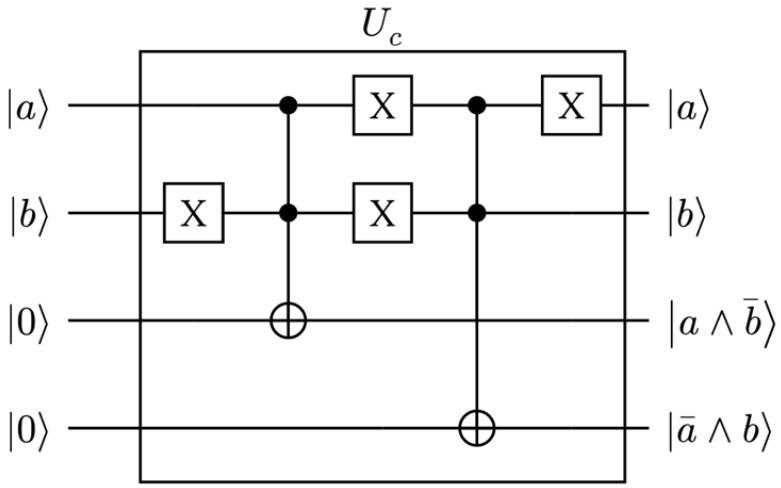


Figure 4: Operator to compare two bits.

The Figure 4. presents the circuit U_c for determining which bit is greater. It requires two qubits $|a\rangle$ and $|b\rangle$ to represent the two bits and two auxiliary qubits initialized to$|0\rangle$. We prepare the states $|a\rangle$ and $|b\rangle$ following 5.2.1. Then, a multi-controlled NOT gate is applied to the first auxiliary qubit with the pair $|a\rangle$ and $|b\rangle$ as control. This first auxiliary qubit is thus in the state $|a \wedge \bar{b}\rangle$. Next, a second multi-controlled NOT gate is applied to the second auxiliary qubit with the pair $|a\rangle$ and $|b\rangle$ as control, $|a\rangle$ having undergone a NOT gate. Thus, the second auxiliary qubit will contain the state $|\bar{a} \wedge b\rangle$.

These states can be elucidated according to the different values of $|a\rangle$ and $|b\rangle$ using a truth table:

$\|a\rangle$	$\|b\rangle$	$\|a \wedge \bar{b}\rangle$	$\|\bar{a} \wedge b\rangle$
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

We can conclude that if the auxiliary qubits are in the state $|01\rangle$ then $a < b$, otherwise $a \geq b$. We also define the operator U_c^{-1} which allows us to uncompute the qubits used by the operator U_c. We then construct the following operator:

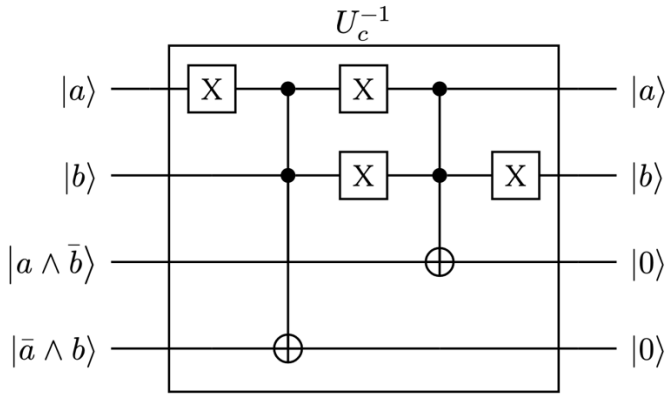


Figure 5. Operator to uncompute the qubits from the comparison operator.

Finally, here is the circuit for determining which bit is greater between $|a_0\rangle$ and $|b_0\rangle$. The last qubit is the qubit that contains the result of the comparison (as previously, the qubits $|a_0\rangle$ and $|b_0\rangle$ are prepared following 5.2.1).

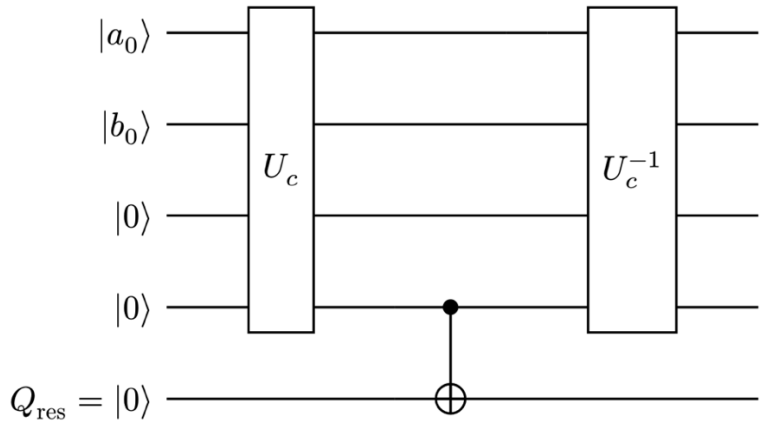


Figure 6. Complete circuit for comparing two bits $|a_0\rangle$ and $|b_0\rangle$.

Thus, by following the previous truth table and noting that the comparison is determined by the auxiliary qubits of the operator U_c, we can interpret the results as follows:

$$\begin{cases} a_0 < b_0 \;\; if \;\; Q_{res} = |1\rangle \\ a_0 \geq b_0 \;\; if \;\; Q_{res} = |0\rangle \end{cases}$$

To compare multiple bits at once, the comparison information generated by the operator U_c is conditionally propagated through the circuit, from the most significant bits to the least significant ones. For this, we use the multi-controlled X gate (MCX). This gate applies an operation to a target qubit only if certain conditions on the control qubits are met. In this context, it allows the propagation of the comparison results from previous bit-by-bit comparisons through the circuit. As soon as we find that one of the bits is greater than the other, we can conclude which integer is larger.

To begin, we present the circuit for comparing two integers of 2 bits.

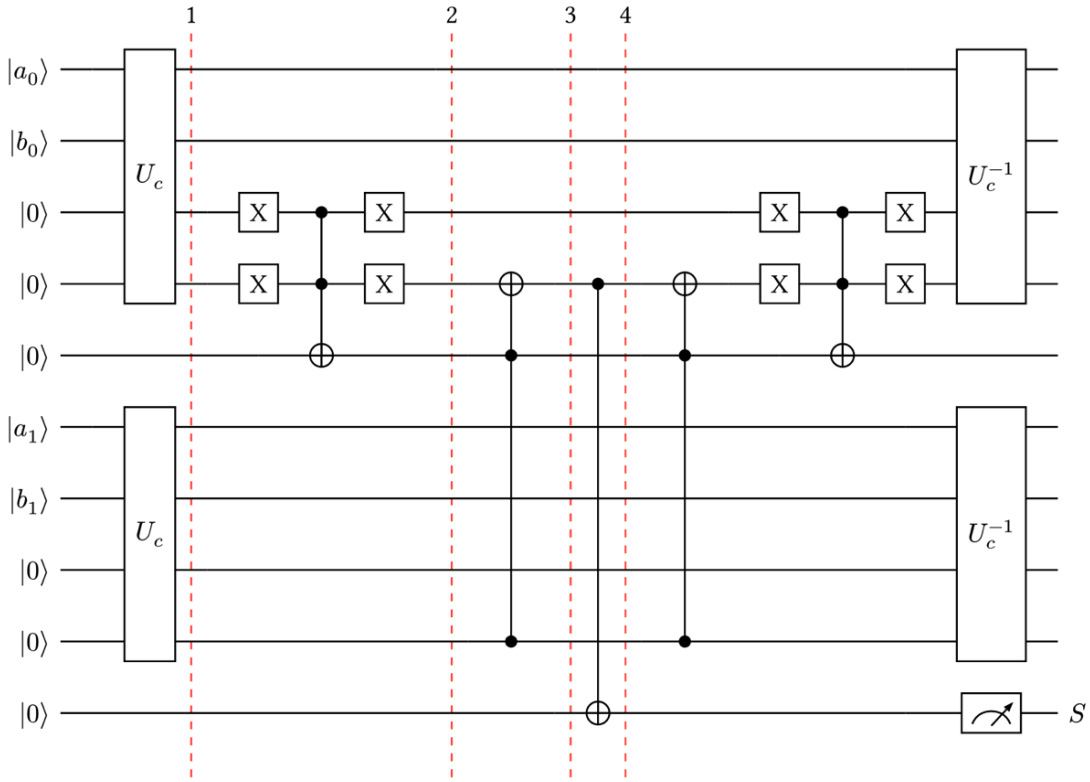


Figure 7. Circuit for comparing two integers *a* and *b* over 2 bits.

The comparison circuit is broken down into several parts. We use the dashed red lines to locate them. Again, qubits $|a_i\rangle$ and $|b_i\rangle$ are prepared following the 5.2.1.

- The U_c gates before line 1 perform the bit-by-bit comparisons.
- Between line 1 and line 2, a multi-controlled X gate is applied to modify the auxiliary qubit for each pair of qubits (except the last one) when the auxiliary qubits from the U_c gate are in $|00\rangle$.
- Between line 2 and line 3, the comparison results from the least significant qubits are propagated to the more significant qubits.
- Between line 3 and line 4, the result qubit is updated.
- After line 4, all the qubits used, except for the result qubit S, are uncomputed.

To obtain the results, as previously, we measure the auxiliary qubit S and observe its state over multiple shots. The most probable state allows us to conclude the comparison between the two integers. If $S = |0\rangle$, then $a \geq b$; otherwise, if $S = |1\rangle$, then $a < b$.

In the Figure 7., we encode two integers a and b. The integer a is written in binary as a_0a_1 and b as b_0b_1, with the most significant bit being the bit at index 0. We first compare the most significant bits of the two integers; if we can deduce at this point which is smaller, subsequent comparisons will not impact, otherwise, we continue with the next bit in each integer. Finally, we measure a 1-bit string S.
This circuit can be generalized for comparing two integers of n bits while maintaining the same logic for the result, that is, if $S = |0\rangle$ then $a \geq b$; otherwise, if $S = |1\rangle$, then $a < b$.

We then obtain the following circuit:

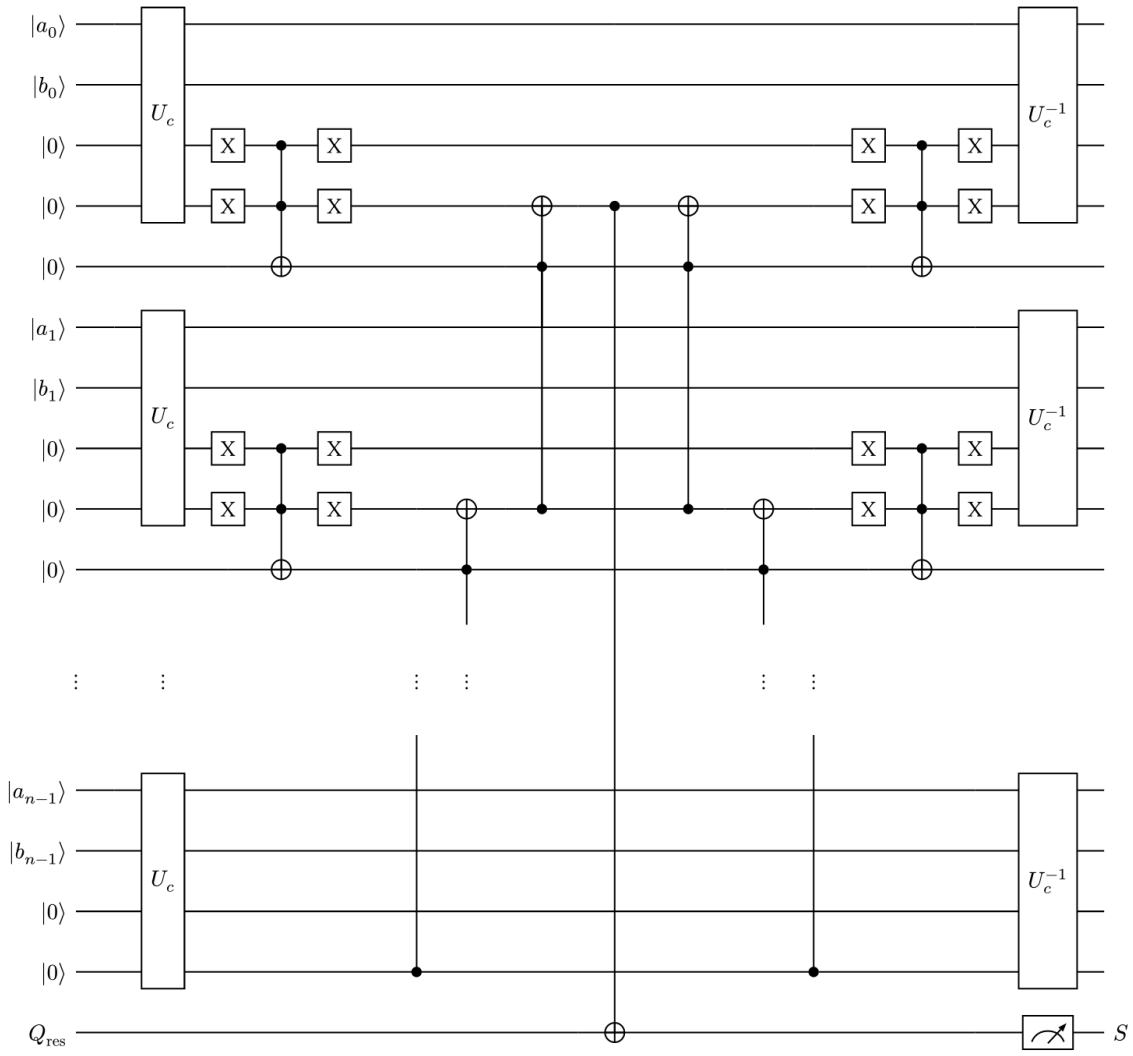


Figure 8. Circuit for comparing two integers a and b over n bits.

To compare two integers over n bits, $5n$ qubits are required.

5.2.3 Hand-on example

Here we take $a = 4$ and $b = 6$. Thus, a in binary is $a_0a_1a_2 = 100$ and b is $b_0b_1b_2 = 110$. We get $S = |1\rangle$ which means that a is strictly less than b.

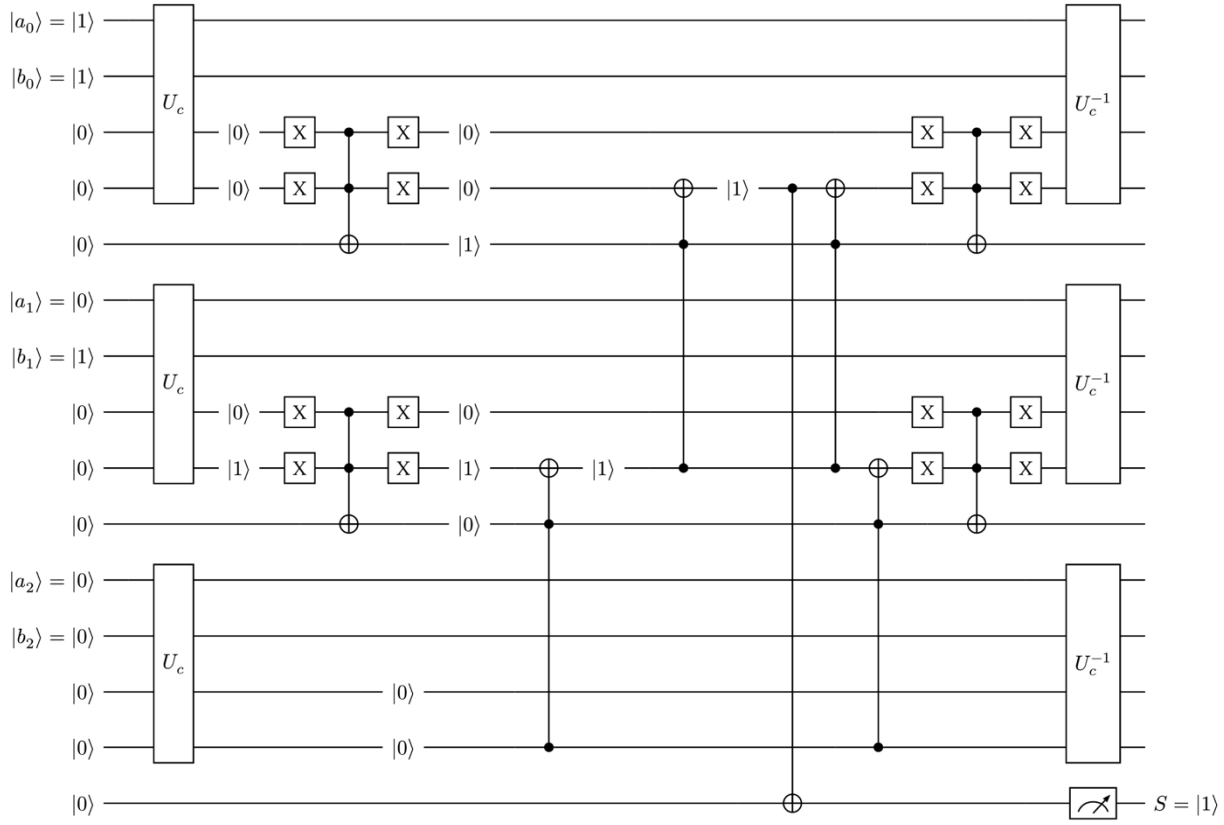


Figure 9. Circuit for comparing $a = 4$ and $b = 6$.

1.5.3 Computation of the minimum of a list of integers

To find the minimum of a list of integers, several quantum algorithms are available, such as the *Dürr-Hoyer* algorithm [4] or the *Quantum Minimum Search* algorithm [5] , both based on Grover's algorithm [6] . In this article, we will use the final circuit constructed in 5.2.2 to create an Oracle to approximate the *Dürr-Hoyer* algorithm.

5.3.1 Algorithm steps

Step 1. Choice of the reference value y_i: If $i = 0$, then we choose y_0 randomly from the list; otherwise, we use the output value from the previous iteration. The binary representation of this value requires n qubits and will be the reference value for comparisons in the integer comparison Oracle.

Step 2. Quantum machine initialization: We initialize a quantum machine in the state:

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n} |x\rangle |0\rangle^{\otimes n} |-\rangle$$

This state is obtained by the operation $H^{\otimes n} I^{\otimes n}$, which creates a superposition over the first register (the $|x\rangle$) and keeps the second register in the state $|0\rangle^{\otimes n}$. The state $|-\rangle$ for the auxiliary qubit is obtained by applying a X gate followed by a Hadamard gate to a $|0\rangle$ qubit. This preparation exploits the phase kickback effect [7] .

Step 3. Preparation of superposed states: The superposition creates 2^n states, but not all these states necessarily correspond to elements of the list. Thus, we will amplify the states present in the list by using Grover's operator applied to multiple solutions (The number of values in the list).

Step 4. Initialization of the value y_i in the quantum machine: We initialize the n qubits $|0\rangle$ so they represent the value y_i on n bits, following the logic described in 5.2.1. We thus obtain the state:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n} |x\rangle |y_i\rangle |-\rangle$$

Step 5. Application of the Integer Comparison Oracle and Diffusion: The integer comparison Oracle is applied to mark states corresponding to integers less than y_i. If a state satisfies this condition, the Oracle will invert the phase of this state through the phase kickback mechanism. That is, a Z gate will be applied to the auxiliary qubit.

The diffusion operator is then applied to amplify the probabilities of the marked states, thereby increasing the chance of measuring a state that represents an integer less than y_i.

Step 6. Measurement and Iteration Logic: Measurements are performed on the first n qubits. The most frequently observed states are those representing integers less than y_i. If the value with the highest probability is in the list and the maximum number of iterations $\sqrt{N}$ has not been reached, it becomes the new reference y_i. If it is not in the list, then either we choose a new value y_0 if it is the first iteration, or we conclude and exit the loop, providing the last value of y_i belonging to L as the answer.
In the Dürr-Hoyer algorithm version [4] , QRAM [8] is used to dynamically select and change the reference value y_i. However, to implement and execute our algorithm, we cannot use QRAM, because it's still theoretical. This is how our algorithm differs from that of Dürr-Hoyer. We need to run the circuit multiple times, changing the reference value y_i each time.

The algorithm for finding the minimum of a list of integers encoded on n bits requires $5n$ qubits. Indeed, it iterates through a circuit based on the circuit for comparing two integers presented in 5.2.2, which also uses $5n$ qubits.

5.3.2 Oracle for superposition state preparation

When the n qubits are in superposition, 2^n states are created, however, not all these states necessarily correspond to the integers in the list. Therefore, we will use Grover's operator, to amplify the states corresponding to the elements of the list. According to Grover's algorithm and as demonstrated in [9] , the number of iterations to be performed is on the order of $\frac{\pi}{4}\sqrt{\frac{2^n}{N}} - \frac{1}{2}$ where 2^n corresponds to the set in which we are searching for values and N is the number of values to find and amplify. The Oracle of Grover's operator to amplify the states corresponding to the values on the list is a series of multi-controlled Z gates, each recognizing an integer from the list.

Here, for example, is the Oracle for the list $L = [5,1,3]$:

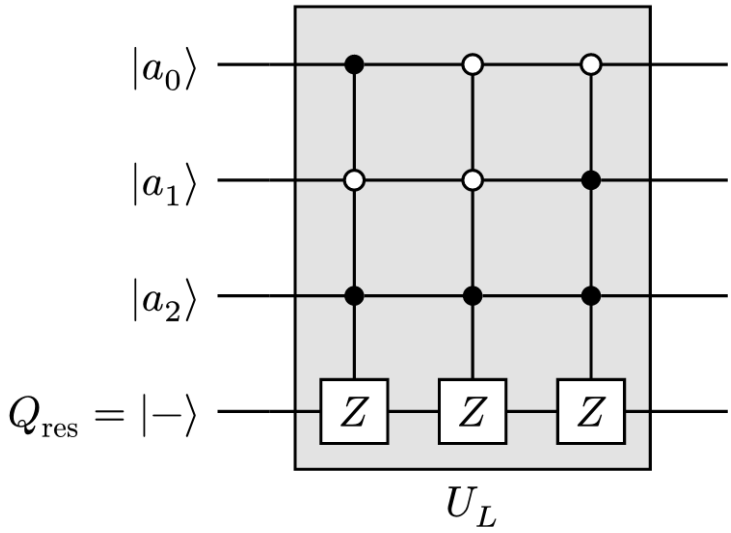


Figure 10. Oracle circuit for state preparation with $L = [5, 1, 3]$

Thus, we indeed have $5 = a_0a_1a_2 = 101$ for the first MCZ gate. Then $1 = a_0a_1a_2 = 001$ for the second and finally $3 = a_0a_1a_2 = 011$ for the last. As such, as soon as a state corresponds to one of the elements of L, its phase will be inverted by the application of a Z gate on the auxiliary qubit, and its probability will be amplified by the diffusion operator.

5.3.3 Integer comparison oracle

Before creating the complete circuit of our algorithm, we will revisit the description of the Oracle for comparison. Thus, the Figure 11. shows what the Oracle circuit looks like for comparing two integers over n bits by applying a Z gate on the auxiliary qubit if $a < b$.

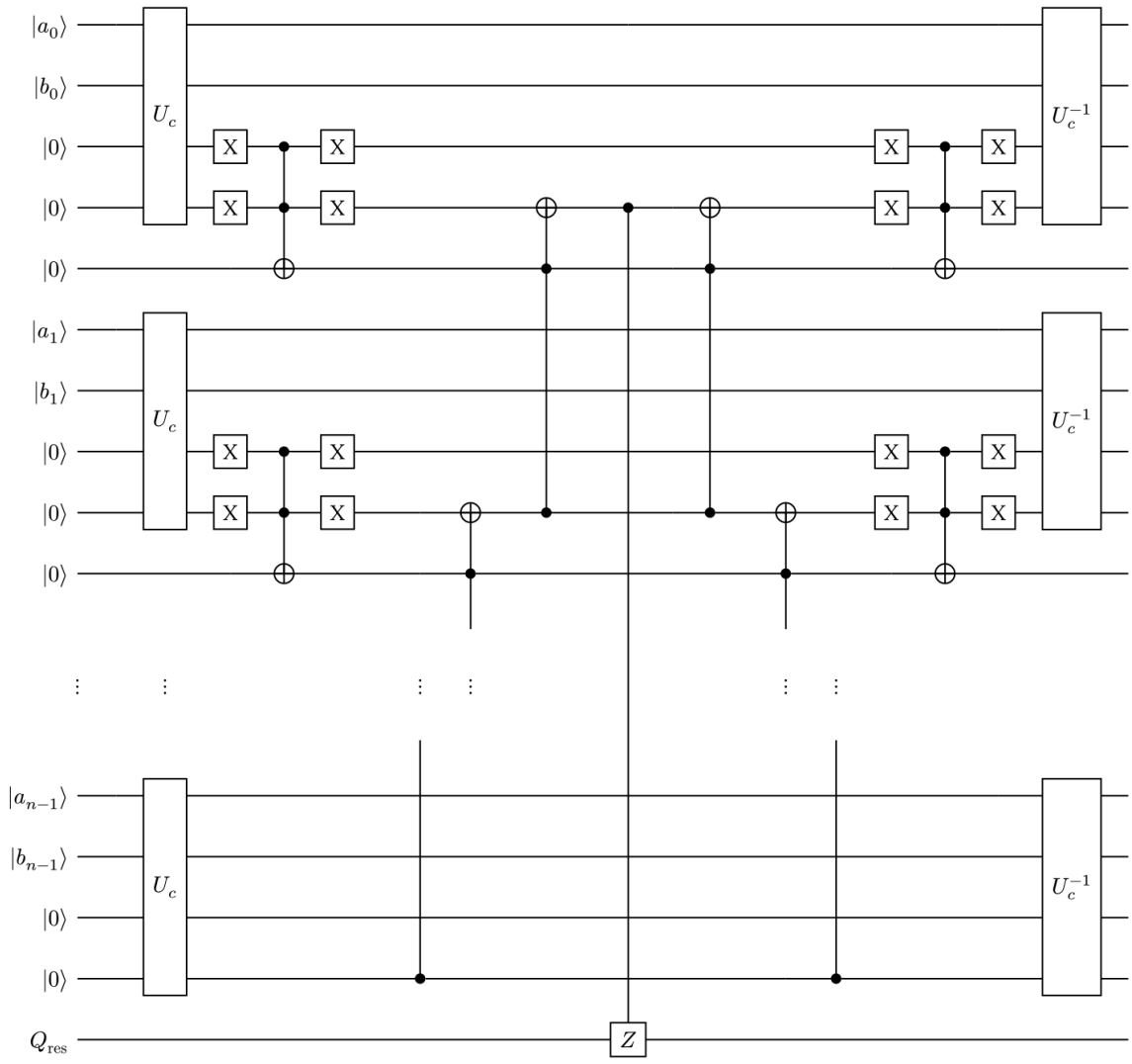


Figure 11. Complete circuit of the comparison Oracle.

As previously mentioned, the Oracle must apply a Z gate on the Q_{res} qubit of the circuit if $a < b$, i.e., when the second auxiliary qubit of the first U_c gate (of the most significant bit in binary notation) is in $|1\rangle$. We consider that a is the integer represented by the superposed states and b is the reference value y_i. We then change the $CNOT$ gate from Figure 8. on the qubit Q_{res} to a CZ gate. Afterward, we apply the diffusion operator to amplify the states that meet the comparison condition.

5.3.4 Definition of the diffusion operator

The diffusion operator is defined for a state $|\psi_H\rangle = H^{\otimes n}|0^n\rangle = |+^n\rangle$ by:

$$\begin{aligned} S_{\psi_H} &= 2|\psi_H\rangle\langle\psi_H| - \hat{I} \\ &= 2|\psi_H\rangle\langle\psi_H| - (\hat{H}\hat{I}\hat{H})^{\otimes n} \\ &= 2\hat{H}^{\otimes n}|0^n\rangle\langle 0^n|\hat{H}^{\otimes n} - (\hat{H}\hat{I}\hat{H})^{\otimes n} \\ &= \hat{H}^{\otimes n}(2|0^n\rangle\langle 0^n| - \hat{I})\hat{\mathrm{H}}^{\otimes n} \end{aligned}$$

Here is an example of its implementation in a circuit with a superposition of 4 qubits. The qubits $|a_i\rangle$ have previously undergone H gates to superpose them, then the Oracle to mark those that need to be marked and which should be amplified by this operator:

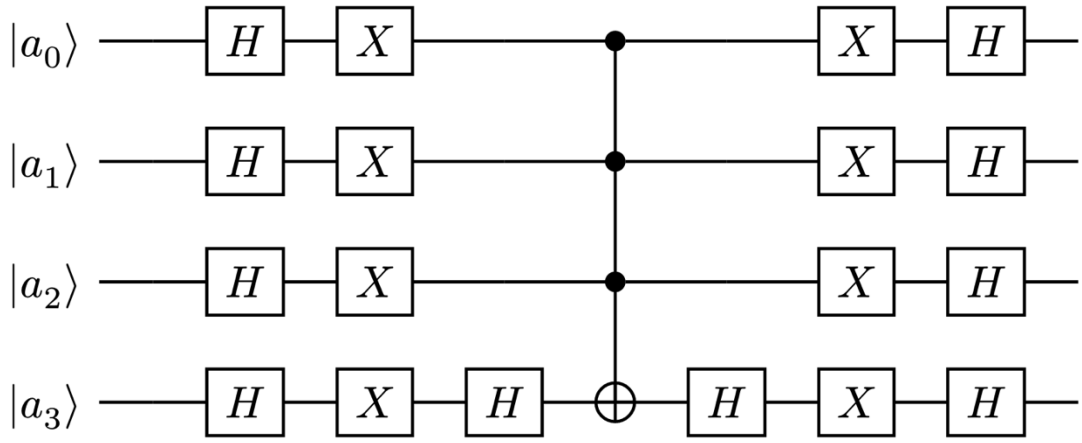


Figure 12. Circuit of the diffusion operator for $n = 4$.

This diffusion operator is Grover's, and it will be used both to amplify states after the Oracle for state preparation and by the Oracle for integer comparison.

5.3.5 Complete circuit

The circuit presented in this section is used in each iteration of the minimum search algorithm. We denote $\hat{G}$ as the operator to be repeated approximately $g \approx \frac{\pi}{4}\sqrt{\frac{2^n}{N}} - \frac{1}{2}$ times, which allows finding the N states that correspond to an integer from the list among the 2^n values created by the superposition and amplifies them with S_{ψ_H}, the diffusion operator. We also denote $\hat{P}$ as the operator to be repeated approximately $p \approx \frac{\pi}{4}\sqrt{\frac{2^n}{N}} - \frac{1}{2}$ times, which contains the integer comparison oracle presented in 5.3.3 and the diffusion operator.

The value y_i, the reference value for comparison during the $\hat{P}$ Oracle, is initialized as described in 5.2.1 and plays the role of b in it. Thus, $\hat{G}$ applies a Z gate on the auxiliary qubit $|-\rangle$ when a is an element of the list L, and $\hat{P}$ does the same when a is less than b, meaning the compared state a is smaller than the reference state $b = |y_i\rangle$.

Note that y_i is encoded on n qubits, and due to its construction, the states created by superposition are also states on n qubits. Furthermore, we do not show the auxiliary qubits of the Oracle in the circuit inputs for clarity.

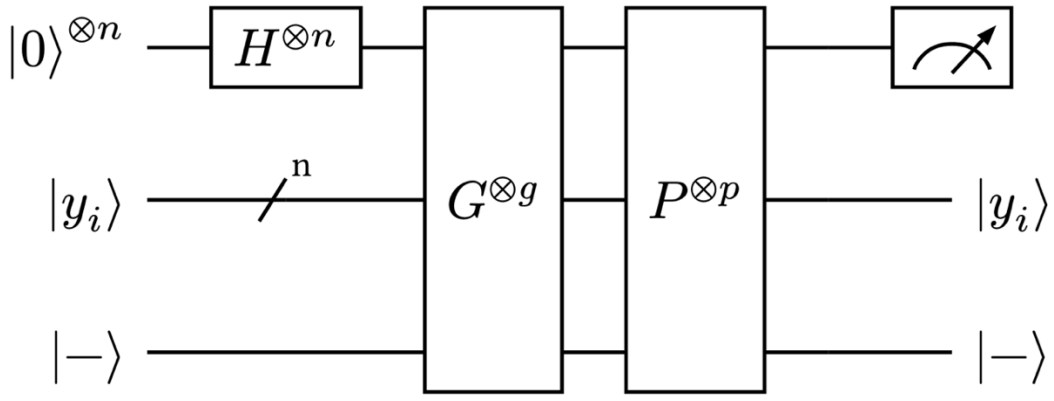


Figure 13. Complete circuit for marking elements of L less than y_i

With the $\hat{G}$ operator applied to the 2^n superposed states, applying a Z gate on the auxiliary qubit to mark the states present in the list. After diffusion, the auxiliary qubit is uncomputed for the next operator $\hat{P}$. Note that $U_L = U_L^{-1}$.

And the $\hat{P}$ operator is applied to all qubits in the circuit and, with the help of internal auxiliary qubits, allows applying a Z gate on the auxiliary qubit when the comparison condition is validated. The comparison oracle is detailed in 5.3.3. Here is a detail of both operator $\hat{G}$ and $\hat{P}$.

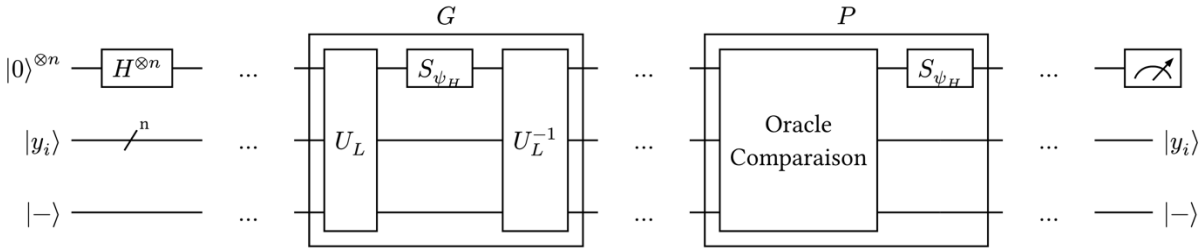


Figure 14. Detail of operators $\hat{G}$ and $\hat{P}$

This circuit thus marks integers less than a certain value y_i. There are n qubits for the superposition of integers from the list, another n qubits for the reference value y_i, and $3n$ auxiliary qubits, of which $3n-1$ are for the integer comparison Oracle and 1 to invert the sign of the states marked by the various Oracles. Measurements allow recovering n-qubits, whose most probable states are integers less than y_i.

As the operators $\hat{G}$ and $\hat{P}$ are repeated about $\frac{\pi}{4}\sqrt{\frac{2^n}{N}}-\frac{1}{2}$ times, the circuit's complexity can be estimated as $C(n,N)=\frac{\pi}{4}\sqrt{\frac{2^n}{N}}-\frac{1}{2}$. Thus, the larger the list, the lower the complexity.

5.3.6 Algorithm specification

Given that this algorithm for finding the minimum in a list of integers encoded on n bits requires $5n$ qubits, then on a quantum computer with k qubits, one could handle a list with integers ranging from 0 to 2^k-1.

5.3.7 Complete algorithm test

The complete algorithm involves initially selecting a random value from the list, denoted as y_0, and providing the circuit presented in 5.3.5 with the list L and this element y_0. The circuit is run t times, each time using the previous value y_i. The algorithm stops when the found minimum y_i is no longer an element of the list, or we have reached the maximum number of iterations set to $\sqrt{N}$ in which case we choose the last computed value as the minimum. All tests of these circuits are also performed on the noiseless *simulator_mps* simulator of the IBM Cloud platform.

Taking the list L such as:

$$L = [3, 4, 7, 8, 10, 13, 5, 12, 1, 11, 6, 2, 9]$$

And choosing a random value $y_0 = 4$, a number of iterations of $\hat{G}$ equal to 1, and a number of iterations of $\hat{P}$ equal to 1, here is the circuit output in terms of probability:

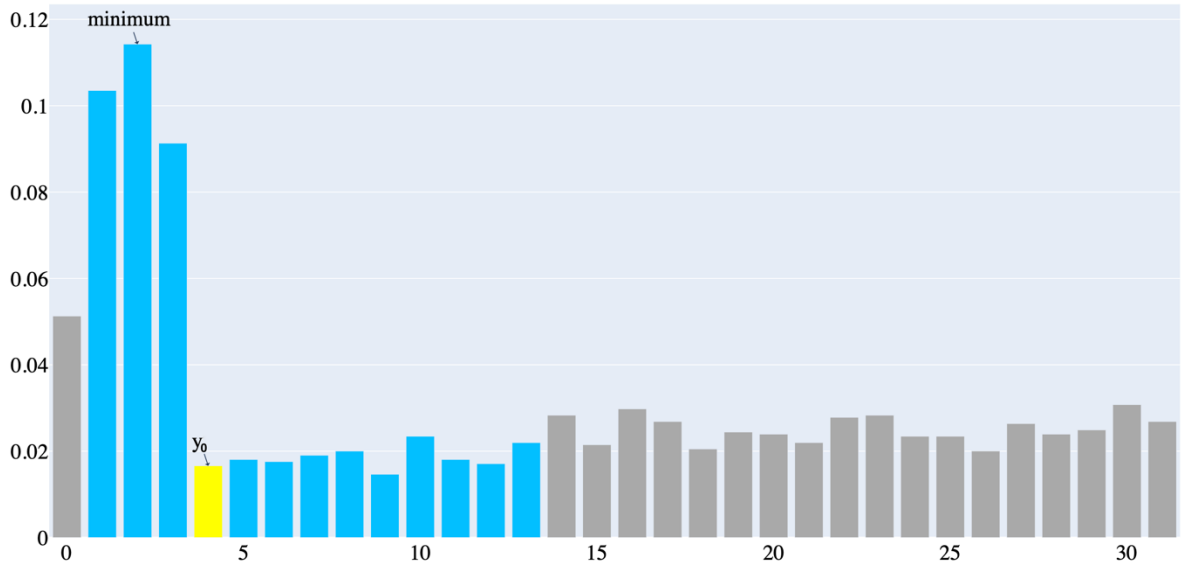


Figure 15. First iteration of the minimum search algorithm.

After this first iteration, the most probable states are those in L and strictly less than $y_0 = 4$. For the next iteration, we thus choose $y_1 = 2$.

Here are the results of the second iteration:

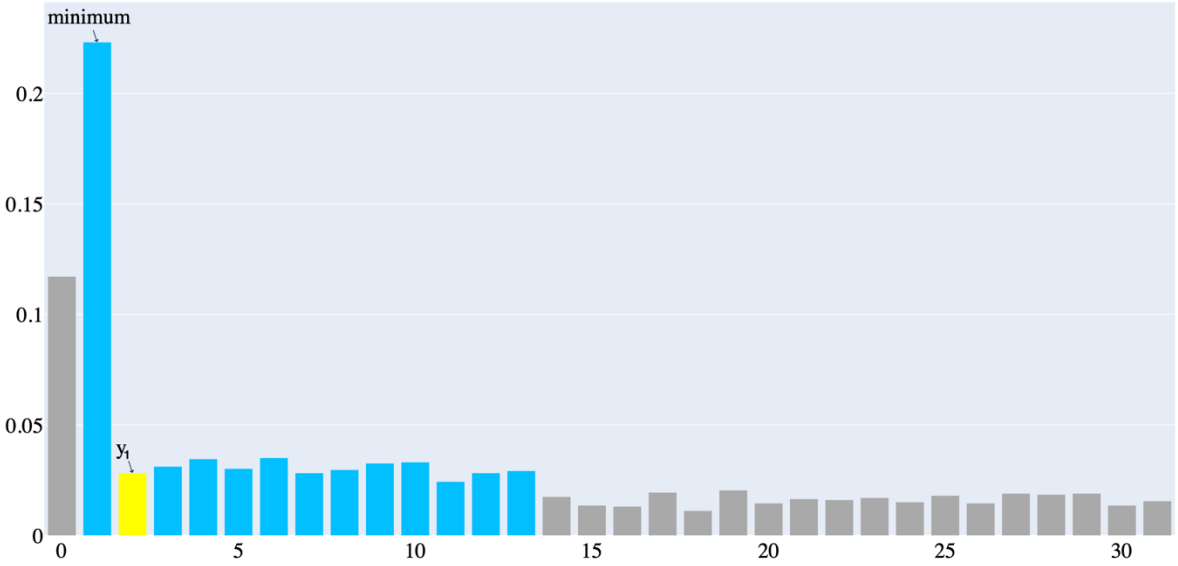


Figure 16. Second iteration with $y_1 = 2$

After this second iteration, the most probable states are limited to $|1\rangle$. We will therefore choose $y_2 = 1$ as the new value. Here are the results of the third iteration:

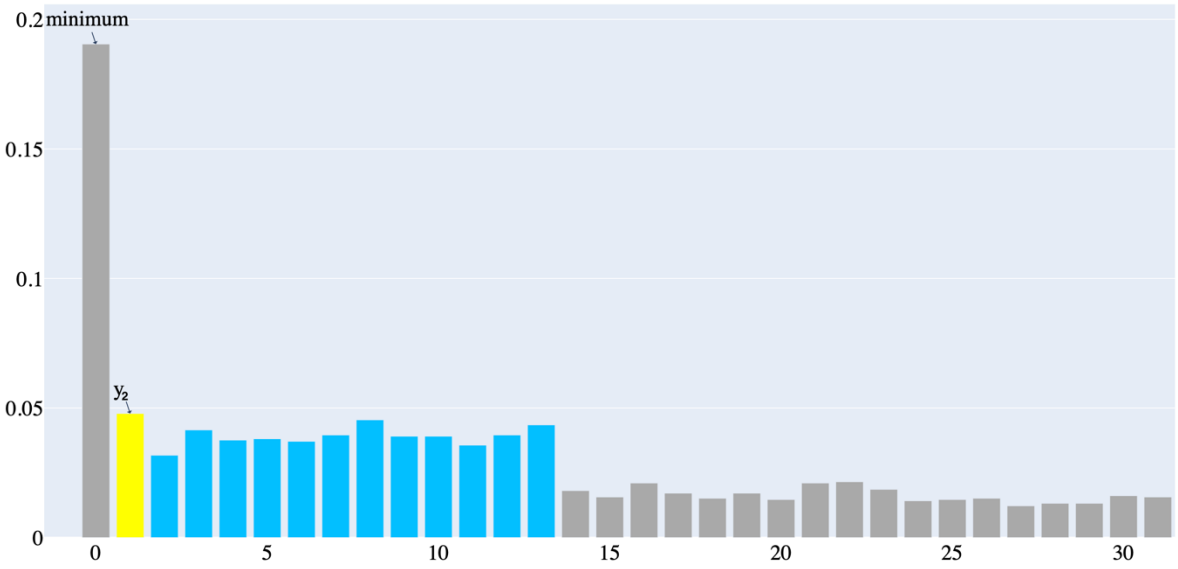


Figure 17. Third iteration with $y_2 = 1$

At this stage, the most probable state corresponds to $|0\rangle$, but since 0 is not in L, we conclude that the minimum of the list is the last found value, thus 1 is the result of the algorithm. In this example, 1 is indeed the true minimum.

1.6 QMeans algorithm

We will now assemble the quantum subroutines developed in the previous section to construct the QMeans algorithm. This version is a hybrid one, alternating between classical and quantum computations.

6.1.1 QMeans++ initialization

The QMeans algorithm comes with its initialization derived from Kmeans++ [10] , named QMeans++. The principle is the same as its classical counterpart, except that the distance calculation and the minimum finding are done quantumly.

We start by choosing a random point from the data, which will be the first centroid of the model. Then, for each new centroid to select, the process is a bit more complex and uses the capabilities of quantum computing. Specifically, for each data point, we calculate the distance between this point and the current set of centroids using the circuit detailed in 5.1.3. And we calculate the minimum of this list with the algorithm detailed in 5.3.1.

We thus obtain the set of minimum distances from each point as follows:

$$distances = [d(x_1, C), d(x_2, C), \dots, d(x_n, C)]$$

Where $d(x, C)$ is the distance from point x to the closest centroid C.

After obtaining these minimum distances for all points, they are classically normalized to obtain a probability distribution. This distribution is used to probabilistically select the next centroid, favoring points that are further away from the already selected centroids. We thus calculate the probability for a point x_i to be selected as a cluster.

$$p(x_i) = \frac{d(x_i, C)}{\sum_{j=1}^{n} d(x_j, C)}$$

By repeating this procedure until the required number of centroids is reached, we finalize the initialization of centroids for the QMeans algorithm.

6.1.2 Estimation of centroid distances

At each iteration of QMeans, we estimate the distance between each point and the cluster centroids. For this, we will use the quantum procedure described in 1.5.1. It allows, from a point X and n centroids C_i, to estimate the distance between X and each of the C_i. For each of the data points X, we assign it to a cluster, following the next section.

6.1.3 Cluster assignment

For each data point X, we thus obtain an estimation of the distance with all centroids. All that remains is to calculate the minimum distance to know the closest centroid and thus assign the point to the correct cluster. For the calculation of the minimum, we use the procedure detailed in 5.3.1.

1.7 QMeans pseudo code

QMeans++ initialization:

Centroids are initialized following QMeans++ algorithm described in 6.1.1.

For n clusters, $3n$ qubits will be needed for distance estimation. For a list of integers of n bits, $5n$ qubits will be required to find the minimum.

While the centroids continue to change positions**:**

Step 1: Distance estimation to centroids (Quantum)

To estimate distances quantumly between a point $x$and a centroid y, the dot product is used as a distance indicator, as explained in 5.1.1.

Thus calculated:

$$P(1_{aux}) = \frac{1}{2} - \frac{1}{2}\langle x, y\rangle^2$$

So, for each point, an estimation of the distance with all centroids is obtained.

For n clusters, $3n$ qubits will be needed.

Step 2: Cluster assignment (Quantum)

For each point, the minimum of the distances with the centroids is calculated to associate them with the correct cluster using the Quantum Bit String Comparator algorithm associated with the *Dürr-Hoyer* algorithm, as explained in 1.5.3.

For a list of integers of n bits, $5n$ qubits will be required to find the minimum.

Step 3: Updating centroids (Classical)

Each new centroid is the centroid of all points in the corresponding cluster. Thus:

$$c_i = \frac{1}{\#C_i}\sum_{x \in C_i} x$$

With $\#C_i$ being the number of points in the cluster C_i.

Result:

Ultimately, the positions of the centroids are obtained.

1.8 Implementation

This article is associated with code, where Kmeans and δ-kmeans have been coded for comparison purposes, as well as QMeans with all the subroutines presented in the previous sections. However, simulating such an algorithm is not feasible with a large number of qubits because it requires too much computational power. And the use of real quantum computers is limited by the waiting time to access IBM's quantum computing platform. Despite this, each quantum subroutine can be used independently with a reasonable number of qubits.

1.9 Future research directions

Testing on Larger Datasets with Increased Quantum Bits (Qubits): A significant direction for future work involves the application of the QMeans algorithm to much larger datasets as the number of qubits in quantum computers increases. The current limitation in qubit availability restricts the scale at which quantum algorithms can be effectively deployed. As quantum hardware continues to advance, enabling the processing of larger datasets, it will be crucial to evaluate the scalability and performance of QMeans in handling big data scenarios. This exploration could unveil the algorithm's practicality for real-world applications, where massive datasets are the norm rather than the exception.

Development of Practical Quantum Random Access Memory (QRAM): The theoretical concept of QRAM plays a pivotal role in the efficiency of quantum algorithms, including QMeans. However, the physical realization of QRAM remains a challenge. Future research should focus on developing practical and scalable QRAM solutions, which would significantly reduce the gap between theoretical quantum algorithms and their practical implementation. Achieving a breakthrough in QRAM technology would not only enhance the performance of QMeans but also unlock new possibilities for quantum computing in processing and analyzing large-scale datasets.

Integration into Machine Learning Frameworks: Another vital area of future research is the integration of QMeans, and quantum algorithms in general, into existing machine learning frameworks. This integration would facilitate the adoption of quantum algorithms by the broader machine learning and data science communities. Efforts should be directed

towards creating quantum-enhanced versions of popular machine learning libraries, thereby making quantum algorithms more accessible to practitioners. Additionally, developing guidelines and best practices for incorporating quantum computing techniques into traditional machine learning pipelines can accelerate the practical applications of quantum machine learning.

By addressing these future research directions, we aim to bridge the gap between the current state of quantum computing and its potential to revolutionize machine learning. Advancements in these areas could significantly impact the efficiency, scalability, and applicability of machine learning algorithms, paving the way for groundbreaking discoveries in various fields.

1.10 Conclusion

In summary, this article has delved into the potential offered by the QMeans algorithm, a quantum variant of the classical Kmeans clustering approach. Emerging at the crossroads of quantum computing and machine learning, this work diverges from the initial theoretical approach proposed by Kerenidis et al. [1] , as we utilize a sequential method rather than a single quantum circuit. While this approach allows for simpler implementation and aligns with the current constraints of quantum computers, it does not realize the full speed potential suggested by theory.

Anecdotally, this project required the deployment of no less than 100,000 quantum circuits on IBM's servers. Despite these challenges, our exploration highlights the transformative potential of quantum computing in enhancing machine learning algorithms. By leveraging quantum subroutines for specific tasks such as distance estimation, minimum value search, and quantum-centric centroid initialization, QMeans++ introduces a novel method that parallels the effectiveness of Kmeans++ in classical settings but executed in quantum superposition.

References

[1] I. Kerenidis, J. Landman, A. Luongo and A. Prakash, q-means: A quantum algorithm for unsupervised machine learning, in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, (December 2019).

[2] S. DiAdamo, C. O'Meara, G. Cortiana, and J. Bernabe-Moreno, Practical Quantum K-Means Clustering: Performance Analysis and Applications in Energy Grid Classification, in *IEEE Transactions on Quantum Engineering*, 3:1-16, (June 2022).

[3] D. S. Oliveira and R. V. Ramos, Quantum bit string comparator: Circuits and applications, *Quantum Computers and Computing*. 7(1):17-26, (2007).

[4] C. Dürr and P. Hoyer, A Quantum Algorithm for Finding the Minimum, *ArXiv*, quant-ph/9607014, (July 1996).

[5] A. S. Albino, L. Q. Galvão, E. Hansen, M. Q. N. Neto and C. Cruz, Quantum algorithm for finding minimum values in a Quantum Random Access Memory, *ArXiv,* abs/2301.05122, (January 2023).

[6] Grover, K. Lov, A fast quantum mechanical algorithm for database search, *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96),* 212-219, (July 1996).

[7] J. Ossorio-Castillo, U. Pastor–Díaz and J. Tornero, A generalisation of the Phase Kick-Back, *Quantum Inf Process* **22**, 143, (March 2023).

[8] V. Giovannetti, S. Lloyd and L. Maccone, Quantum random access memory, *Physical review letters,* 100(160501), (April 2008).

[9] E. Borbély, Grover search algorithm, *ArXiv,* abs/0705.4171, (May 2007).

[10] D. Arthur and S. Vassilvitskii, K-means++: the advantages of careful seeding, *Proceedings of the 19th annual ACM-SIAM symposium on Discrete algorithms (SODA '07)*, 1027-1035, (January 2007).

[11] S. Lloyd, M. Mohseni, and P. Rebentrost, Quantum algorithms for supervised and unsupervised machine learning, *arXiv*, 1307.0411:1-11, (July 2013).

[12] J. M. Arrazola, A. Delgado, B. R. Bardhan, and S. Lloyd, Quantum-inspired algorithms in practice, *arXiv preprint arXiv:1905.10415*, (May 2019).

[13] E. Tang, A quantum-inspired classical algorithm for recommendation systems. *arXiv preprint arXiv:1807.04271*, (July 2018).

[14] E. Tang, Quantum-inspired classical algorithms for principal component analysis and supervised clustering, *arXiv preprint arXiv:1811.00414*, (October 2018).