Neural, Parallel, and Scientific Computations 22 (2014) 127-148

# QUANTUM $K$-MEANS CLUSTERING

SRIMANTA PAL[1] AND PIJUSH KUMAR KOLEY[2]

[1]Electronics and Communication Sciences Unit, Indian Statistical Institute,
203 B T Road, Kolkata 700 108, India
[2]Department of Instrumentation and Electronics Engineering
Jadavpur University, Kolkata, India
*e-mail*: [1]srimanta@isical.ac.in [2]pijush-kolay@yahoo.com

**ABSTRACT.** In this paper we first propose a quantum subtraction algorithm along with its quantum circuit and the quantum circuit for computation of the inverse Fourier transform which is used in computation of a quantum mean. Finally we present a quantum mechanical version of the $K$-means clustering algorithm. Quantum $K$-means clustering algorithm takes only $O(LNK)$ time while classical the $K$-means takes $O(LNpK)$ after $L$ iterations with $N$ data points of dimension $p$. This algorithm, thus, performs better than classical algorithm when dimension $p$ of data is large enough.

**AMS (MOS) Subject Classification.** 65C60, 68M15, 94C12

## 1. Introduction

Feynman observed that simulation of a quantum mechanical system on an ordinary computer would entail an exponential slowdown in the efficiency. This is due to the fact that the description of the size of the quantum system would become exponential [21]. Feynman wrote [22] *"But the full description of quantum mechanics for a large system with R particles is given by a function $\psi(x_1, x_2, \cdots, x_R, t)$ which we call the amplitude to find the particles $x_1, x_2, \cdots, x_R$, and therefore, because it has too many variables, it cannot be simulated with a normal computer with a number of elements proportional to R ...."* He suggested that the only way to overcome this shortcoming was to simulate the quantum mechanical system on a computer governed by quantum laws. Quantum computing has now emerged as a synthesis of ideas from fields like computer science and quantum mechanics.

Deutsch [15] in 1985 first established a solid ground for quantum computation. After this, research on quantum computing remained at a low profile until 1994 when Shor [38] proposed quantum algorithms for factoring integers and extracting discrete logarithms in polynomial time and in 1996 Grover [23] proposed a quantum search algorithm for a database, which is quadratically faster than known classical algorithm.

Received December 18, 2013 

Over the last few years there has been a very rapid development of methods for processing quantum information [2, 3, 4, 5, 8, 9, 10, 11, 12, 14, 16, 17, 18, 24, 25, 35, 37, 39, 42].

In this paper we present a quantum $K$-means clustering algorithm. This algorithm is well known and widely used in pattern recognition. The time complexity of its classical version is $O(LNpK)$ but its quantum mechanical version takes $O(LNK)$. This will save enormous time for data mining type applications where dimension $p$ is large. This paper is organized as follows. In Section 2, the classical model of $K$-means clustering algorithm is described with its time-complexity $O(LNpK)$. Section 3 presented few basic concepts of quantum computing techniques. In Section 4, details for the design of quantum $K$-means clustering algorithm is described. This also includes representation of data and centroides as well as quantum computations of squared distance, minimum, mean, discrete logarithm, Fourier transform, inverse Fourier transform, exponentiation. Time-complexity $O(LNK)$ of quantum $K$-means clustering algorithm is presented in this section. A brief concluding remark is also given in Section 5.

## 2. Classical Model of K-Means Clustering Algorithm

The K-means clustering algorithm tries to partition a data set $X \subset R^p$ into $K$-groups $X_1, X_2, \ldots, X_K$ such that $X_i \bigcap X_j = \phi$, $\bigcup_{i=1}^{K} X_i = X$, $|X| = N$ and each $X_i$ is homogeneous [1, 19, 32, 36]. The partitioning algorithm is deriven by a set of centroides $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_K\}$, $\mathbf{v}_i \in R^p, i = 1, 2, \ldots, K$, generated from the data set $X$. The process requires a method for computing a centroid point $\mathbf{v}_j$, for some subset of the data points $X$, and a metric $D(\mathbf{x}_i, \mathbf{v}_j)$ between data point $\mathbf{x}_i$ and the centroid point $\mathbf{v}_j$.

The classical K-means clustering algorithm is described in Algorithm 1 which iterates between two major steps such as (1) updating of the clusters and (2) updating of the centroides.

We now provide an algorithmic description of a version of $K$-means algorithm.

**Algorithm 1.** Classical $K$-means clustering algorithm
**Input**: Data $X_i, i = 1, 2, \ldots, N$ of $p$ dimensions
**Output**: Centroid $\mathbf{v}_i, i = 1, 2, \ldots, K$
**Step 1**: [**Initialization**] Randomly generate a set of $K$ centroides $\mathbf{v}_i(0)$
for $i = 1, 2, \ldots, K$
$t = 0$

**Step 2**: [**Loop for classification of $X \subset R^p$ into $K$-groups for the updation of centroides**]
Repeat
$n_j = 0 \ \forall \ j = 1, 2, \ldots, K$

for $i = 1$ to $N$ do

begin

find $j = Augmin_l ||\mathbf{x}_i - \mathbf{v}_l(t)||^2$

$\mathbf{v}_j(t+1) = \mathbf{v}_j(t+1) + \mathbf{x}_i$

$n_j = n_j + 1$

end;

**Step 2.1**: [**Updating the Centroids**]

for $i = 1$ to $K$ do

$$\mathbf{v}_i(t+1) = \frac{\mathbf{v}_i(t+1)}{n_i}$$

$t \leftarrow t + 1$
until $||V(t) - V(t+1)||^2 \leq \epsilon$

**Step 3**: [**Termination**] Stop

2.1. **Analysis.** Now we discuss the complexity of classical K-means algorithm. Consider the data point $\mathbf{x}_i$ and determine its closest representation $\mathbf{v}_j$, where $i = 1, 2, \ldots, N$. The following steps are involved

**Step 1**: Compute distance $D(\mathbf{x}_i, \mathbf{v_j}),\ j = 1, 2, \ldots, K$, takes $(3p-1)K$ operations.

**Step 2**: Compute the minimum distance $D_l = \min_{1 \leq l \leq k} D_i$ requires $(K-1)$ operations.

The total time requirement for updating the cluster with $N$ points is

$$N[(3p-1)K + K - 1] = N(3pK - 1)$$

and for updating the center is $Np + N + K$.
Hence the time complexity for an iteration is

$$3NpK - N + Np + N + K = 3NpK + Np + K = O(NpK).$$

If the algorithm converges after $L$ number of iterations then the time complexity for Algorithm 1 is $O(LNpK)$.

Before going into the proposed quantum mechanical version of $K$-means clustering algorithm, basic concepts of quantum computing will be discussed in Section 3.

## 3. **Basic Concepts of Quantum Computing Techniques**

In this section we discuss quantum gate arrays or quantum acyclic circuits, which are analogous to acyclic circuits in classical computer science. We also discuss about reversible computation. Besides the network model of quantum computer, there are two other types of models such as quantum Turing machine [8, 15, 43] and quantum cellular automata [22, 29, 30, 31, 33, 34]. Quantum Turing machine and quantum gate arrays can compute some function with a small probability of error in polynomial time [43].

Suppose a system has $n$ components and each component can have two states. Classically we can represent the system with $n$ bits, but in quantum mechanics we need $2^n$ complex numbers for a complete description of the system, i.e., the state of the quantum system is a point in a $2^n$-dimensional Hilbert space. A quantum state is represented by the *ket* notation (first used by Dirac). The Hilbert space associated with this quantum system is the vector space with these $2^n$ states as basis vectors. A unit-length vector in this Hilbert space represents a state of the system at any instant of time. In quantum computation the superposition of a state is represented by $\sum_{i=0}^{2^n-1} \alpha_i |\mathbf{x}_i\rangle$ where, $\alpha_i$ = amplitude of $i$th state such that $\sum_i |\alpha_i|^2 = 1$ and $|\mathbf{x}_i\rangle$ = a basis vector of the Hilbert space.

Quantum circuits allow only local unitary transforms, i.e., unitary transforms on a fixed number of qubits [2, 4, 5]. Two qubit transforms are more useful than any general unitary transform which takes place on $n$-qubits, because it is not easy to implement $n$-qubit transforms, whereas two-bits transformations can be implemented by relatively simple physical systems. These two qubits transformations are the heart of a quantum computer. For quantum computation there are two well known quantum gates, NOT and controlled-NOT gates. NOT gate has a single qubit input and controlled-NOT gate has two qubits input. The input and output of a NOT gate and a controlled-NOT gate are shown below.

A NOT gate has a single qubit input and a controlled-NOT gate has two qubits input. The input and output of a NOT gate and a controlled-NOT gate are shown in Table 1 and Table 2 respectively.

Table 1: Input and output relation of a NOT gate

| Input qubit | Output qubit |
|---|---|
| $\|0\rangle$ | $\|1\rangle$ |
| $\|1\rangle$ | $\|0\rangle$ |

Table 2: Input and output relation of a controlled-NOT gate

| Input qubit | Output qubit |
|---|---|
| $\|00\rangle$ | $\|00\rangle$ |
| $\|01\rangle$ | $\|01\rangle$ |
| $\|10\rangle$ | $\|11\rangle$ |
| $\|11\rangle$ | $\|10\rangle$ |

As mentioned earlier quantum circuits allow only local unitary transforms. So to realize these gates we need unitary matrices. It is not difficult to construct the unitary matrices to achieve the desired goals. The rows of such a matrix correspond to input basis vectors and the columns represent output basis vectors. If the $i$th basis vector, when applied as an input to the gate, produces the $j$th basis vector as the output, then the $(i, j)$th entry of the matrix is set to the amplitude of the output vector (in this case the amplitude is 1); otherwise, it is set to 0. Thus the matrix $M_{NOT}$ corresponding to the NOT gate is shown in Table 3.

Table 3: Unitary matrix corresponding to a NOT gate

| | $\|0\rangle$ | $\|1\rangle$ |
|---|---|---|
| $\|0\rangle$ | 0 | 1 |
| $\|1\rangle$ | 1 | 0 |

$$M_{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Similarly, for a controlled-NOT gate the unitary matrix $M_{CNOT}$ is shown in Table 4.

Table 4: Unitary matrix corresponding to a controlled-NOT gate

| | $\|00\rangle$ | $\|01\rangle$ | $\|10\rangle$ | $\|11\rangle$ |
|---|---|---|---|---|
| $\|00\rangle$ | 1 | 0 | 0 | 0 |
| $\|01\rangle$ | 0 | 1 | 0 | 0 |
| $\|10\rangle$ | 0 | 0 | 0 | 1 |
| $\|11\rangle$ | 0 | 0 | 1 | 0 |

$$M_{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

However for simulating AND, OR, NAND, NOR gates we need Taffoli gate [40]. The input-output relation for the Taffoli gate is Table 5.

Table 5: Unitary matrix corresponding to a NOT gate

| Input | Output | Input | Output |
|---|---|---|---|
| $\|000\rangle$ | $\|000\rangle$ | $\|001\rangle$ | $\|001\rangle$ |
| $\|010\rangle$ | $\|010\rangle$ | $\|011\rangle$ | $\|011\rangle$ |
| $\|100\rangle$ | $\|100\rangle$ | $\|101\rangle$ | $\|101\rangle$ |
| $\|110\rangle$ | $\|111\rangle$ | $\|111\rangle$ | $\|110\rangle$ |

Taffoli gate is a two controlled-output (taget input) gate. The schematic diagram of a Taffoli gate is shown Fig. 1(a). In this figure there are two controlling inputs, $|x_1\rangle$ and $|x_2\rangle$, and one input qubit. If two controlling qubits are $|1\rangle|1\rangle$ then the input qubit flips, i.e., if the input qubit is $|0\rangle$, then it is changed to $|1\rangle$ and vice versa.

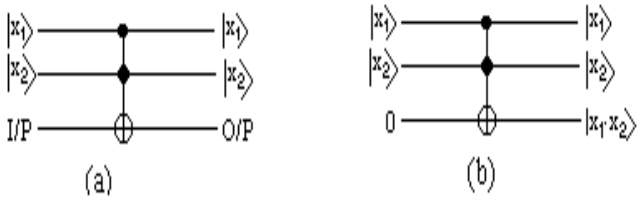


FIGURE 1. (a) Taffoli gate and (b) Quantum AND gate.

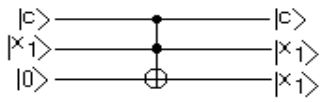


FIGURE 2. Quantum copy gate.

Using Taffoli gate we can implement other gates also. An AND gate can be easily realised from Taffoli gate by simply keeping the input qubit always to $|0\rangle$. Its implementation is shown in Fig. 1(b). Similarly, we can implement controlled copy gate using Taffoli gate as shown in Fig. 2. In this case one control qubit controls the copy operation and the control qubit is the qubit that is to be copied and target qubit is set to 0.

A quantum gate array is a set of quantum gates with logical "wires" connecting their inputs and outputs. When quantum gate arrays are associated with a quantum Turing machine, these gate arrays must be in a uniform complexity class. To make it uniform two additional requirements are added to the definition of quantum gate arrays. The first one is that the design of the gate array be produced by a polynomial-time computation and the second one is the standard part of the definition of analog complexity classes. The entries in the unitary matrix for gate representation must be computable numbers.

An important aspect of quantum computation is reversibility. The quantum gate arrays perform reversible computations. That is if we know the quantum state on the wires leading out of a gate, we can tell unambiguously the input state of the gate. This reveals that the laws of physics are reversible despite the macroscopic arrow of time. In a classical computer energy is dissipated so the computations are thermodynamically irreversible. But in case of a quantum computer we have to maintain the superposition of quantum states throughout the computation, so it uses reversible computation. This imposes extra costs when a classical computation is performed on a quantum computer. This raises another issue, if we want to perform a deterministic computation, we have to convert it into a reversible one. In classical paradigm a Turing machine computes output from input according to its control function non-reversibly, i.e., we cannot get back the input from its output. In this

case there are two types of tapes associated with the Turing machine, one is input tape and other is output tape. To convert the computation on Turing machine from irreversible to reversible we add another tape (initially blank) in which it saves all information of intermediate states. We can call these information history and the tape as record tape. History must be of sufficient detail so that the preceding state can be uniquely determined by the present state. At the end of the computation only the input and output should be present and the record tape must be erased. A tape full of random data cannot be erased except by an irreversible process. But the data on the record tape are not random, there exists mutual redundancy between the data and the machine that produced it, which will be helpful to erase the data reversibly from the record tape. For example, suppose we get an output from an input and in the record tape there is sufficient history of the operation performed on the input. If we reverse the operation then the record tape returns to its original blank condition. But the backward steps would convert the output into the original input. As a result the whole operation will be useless. Annihilation of desired output can be prevented simply by making an extra copy of it on a separate tape after the completion of forward computation and before backward computation. During the copying operation the data on record tape remain unaltered. This copying operation must be done reversibly. Now if backward or inverse operation is applied, the output will be erased out not the copy of the output. As a result at the end of whole computation there will be only the input and output. Lecerf [27], Bennett [6, 7] already showed that any non-reversible deterministic computation can be converted into a reversible deterministic computation. In 1989 Bennett [7] has given a method to convert a non-reversible deterministic computation to a reversible deterministic computation using a Turing machine. Table 6 shows Bennett's proposed method.

Table 6: Reversible deterministic computation using Bennett's method [7]

| Step | Input tape | Record tape | Working tape | Output tape |
|---|---|---|---|---|
| 1 | input | —— | —— | —— |
| 2 | input | record($R$) | output | —— |
| 3 | input | record($R$) | output | output |
| 4 | input | —— | —— | output |
| 5 | input | record($R^{-1}$) | input | output |
| 6 | —— | record($R^{-1}$) | input | output |
| 7 | —— | —— | —— | output |

Bennett's algorithm uses four tapes (Table 6) associated with the Turing machine: *input* tape, *record* tape, *working* tape and *output* tape. To convert a non-reversible deterministic computation to a reversible deterministic computation sufficient history must be kept to reverse back to previous state. In case of classical computation we do not keep any history and we cannot go back to previous state. Record tape is used in

Bennett's method to store the intermediate state of the computation and final output is obtained from the output tape. According to Table 3, the first 4 steps, i.e., steps 1 to 4 compute reversibly the output from the input using function $R$ and next 3 steps, i.e., steps 4 to 7 computes reversibly input from output using function $R^{-1}$. After the completion the output is obtained from the output tape. Now we describe the action of each step.

**Algorithm 2.** To convert a non-reversible deterministic computation to a reversible deterministic computation (Bennett's method [7]).

**Input**: Input record

**Output**: Output record

**Step 1**: Input is copied to input tape.

**Step 2**: Output is computed from input and sufficient history of computation is stored in the record tape and the output is stored in working tape.

**Step 3**: Output is copied into output tape from working tape.

**Step 4**: History and output are erased from record tape and working tape respectively.

**Step 5**: Input is computed from output using inverse function and sufficient history of computation is stored in record tape and input is stored in working tape respectively.

**Step 6**: Input is erased from input tape.

**Step 7**: Input and record are erased from working tape and record tape respectively. Only output remains in output tape.

**Step 8**: [**Termination**] Stop

In the above computation we need to compute $R(input)$ and $R^{-1}(input)$. These two functions must be computable in polynomial time otherwise computation time will increase exponentially with the increase of input.

Now computations can be made reversible with a cost of constant factor time and space. Even if classical computation takes much less space and time, when we make it reversible, by Algorithm 2 it may take a lot of space. Levine and Sherman [28] proposed another method which takes more time but less space. There is no general method to ensure that a method will not take much time and space.

If we try to convert a non-reversible deterministic computation to reversible deterministic computation through a network all classical gates (AND, OR, NAND, NOR) have to be converted into reversible gates using Taffoli gate and for this we need additional input bits which must be reset to 0 after the completion of computation, otherwise they will affect the interference pattern in quantum computation. A quantum network is a quantum computing device consisting of quantum logic gates and these gates operate synchronously. The network inputs are stored in qubits where

a qubit is prepared appropriately at the beginning of each computation performed by the network. Inputs are encoded in a binary form as a computational basis for selected qubits. This is known as quantum register. For example the binary form of number $4_{10}$ or 4 is $(100)_2$ and it is represented in a quantum register with three qubits in state $|1\rangle \bigotimes |0\rangle \bigotimes |0\rangle$.

Here we provide a compact notation:

$$|b\rangle = |b_n\rangle \bigotimes |b_{n-1}\rangle \bigotimes \cdots \bigotimes |b_1\rangle \bigotimes |b_0\rangle$$

which represents a quantum register of $n$ qubits prepared with the value

$$b = 2^0 b_0 + 2^1 b_1 + \cdots + 2^n b_n.$$

Similarly, if we require more qubits then expand it as we do in a binary representation. Each computation is a unitary transform which takes input as initial state and produces an output as the final state.

## 4. **Design of Quantum K-Means Clustering Algorithm**

Now we realize the clustering algorithm described in Algorithm 1, we need mechanisms to realize the following in a quantum paradigm:

1. Representation of a data point, $\mathbf{x}_i \in R^p$, so that it can be identified and processed.
2. Representation of centroids so that they can be manipulated.
3. Computation of squared distances.
4. Finding the minimum of a set of real values.
5. Addition of real vectors.
6. Subtraction of real vectors.

Now we shall explain how each of the above issues can be addressed in a quantum computing paradigm.

### 4.1. **Representation of Data.**

Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset R$ and $n$ is the smallest integer such that $N \geq 2^n$. $k$ is the smallest integer such that $K \geq 2^k$. Suppose to store a real value we need $r$ qubits (this will depend on the desired precision) then to store a data point in $R^p$ (i.e., $\mathbf{x}_i \in R^p$) we need at least $pr$ qubits. In addition to this we need $n$ qubits to store the index of a data points and $k$ qubits to store the index of the centroid closest to a data point. Hence, to store $\mathbf{x}_i \in R^p$ we need a composite register with $(n + k + rp)$ qubits as shown in Fig. 3(a).

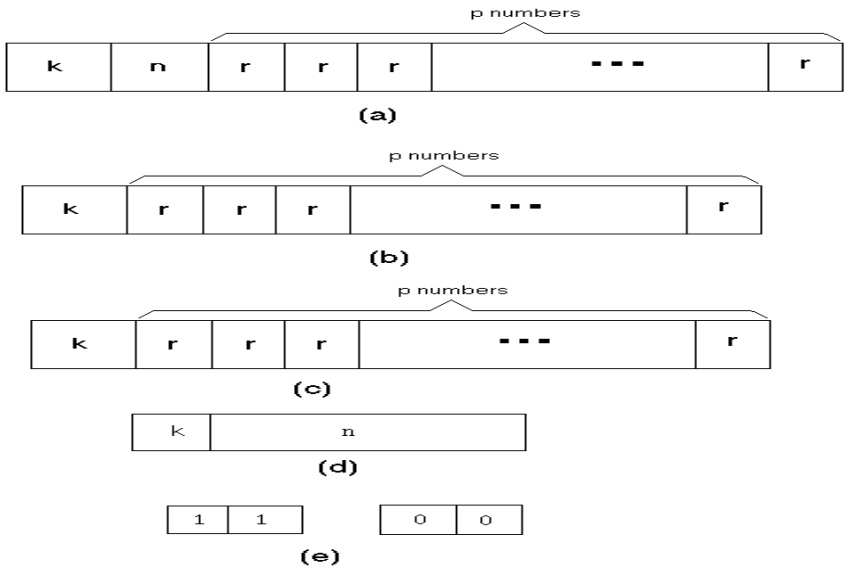


FIGURE 3. (a) Data register (b) $centroid_{new}$ register (c) $centroid_{old}$ register (d) integer register (e) constant register

4.2. **Representation of Centroids.** As stated earlier the number of centroids is $K$ and $k$ is a smallest number such that $K \geq 2^k$. Centroids are also in $p$ dimension. So to store a centroid at least $pr$ qubits are needed. Again we need another $k$ qubits to store the index of centroids. Hence, to store a centroid we need another type of composite register with $(k + rp)$ qubits as shown in Fig. 3(b). For our proposed algorithm we will need two sets of such composite register, one set for storing the new set of cetroids and other set for storing the old set of centroids.

In addition to these there will be temporary composite registers to hold the intermediate results of computation and two constant registers each with only two qubits. These two registers will be used to store 0 and 1. The first qubit will be needed for identification, while the second qubit for the value. These two registers are shown in Fig. 3(e). These two registers will be used for quantum arithmatic operations. There will be another integer type composite register with $(k+n)$ qubits, (Fig. 3(d)). The first $k$ qubits will be used for indicating the centroid index and the next $n$ qubits will be used to indicate number of data points associated to a cluster.

4.3. **Quantum Squared Distance Computation.** The squared distance computation requires the following basic operations: (a) subtraction, (b) addition and (c) squaring.

4.3.1. *Quantum Subtraction.* In the classical paradigm, subtraction operation is performed using logic gate network which is irreversible. But in quantum computing

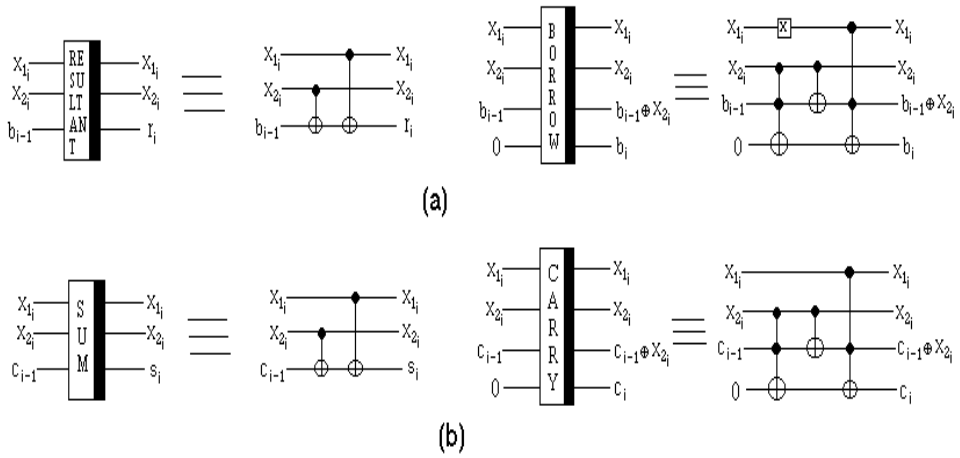


FIGURE 4. (a) Quantum subtraction operation and (b) quantum addition operation.

paradigm subtraction operation must be reversible. A quantum subtraction operation was first proposed by Vedral et al. [41] and was modified by Koley and Pal [26].

Consider two quantum registers $R_1$ and $R_2$. $R_1$ contains data $\mathbf{x}_1$ and $R_2$ contains $\mathbf{x}_2$. Let us define the $i$th qubit of $R_1$ as $x_{1_i}$ and that of $R_2$ as $x_{2_i}$. If $a$ is a qubit then $\overline{a}$ denotes NOT $a$.

To realize subtraction, we compute the borrow $b_i$ as

$$b_i \leftarrow (\overline{x}_{1_i}\ AND\ x_{2_i})\ OR\ (\overline{x}_{1_i}\ AND\ b_{i-1})\ OR\ (x_{2_i}\ AND\ b_{i-1}) \tag{4.1}$$

and the resultant output is computed by

$$r_i = x_{1_i}\ XOR\ x_{2_i}\ XOR\ b_{i-1}.$$

Subsequently we reverse all these operations in order to restore every qubit of the temporary register to its initial state 0 as we use this network for subtraction repeatedly.

The quantum version of the above procedure can be written as

$$|x_1\rangle|x_2\rangle|0\rangle \rightarrow |x_1\rangle|x_2\rangle|x_1 - x_2\rangle$$

and its quantum circuit is shown in Fig. 4(a). This figure has two parts, one is for resultant computation and other is for computation of the borrow. The resultant computation module accepts three inputs and produces three outputs. Here resultant is generated by XOR-ing all inputs. The borrow calculation is slightly difficult. It accepts four inputs and produces four outputs as shown in Fig. 4(a).

This subtraction operation is defined for two scalar numbers. We can easily compute subtraction operation between two vectors using the same circuit. To do this

we need two composite registers which will store the data and for each component we perform the subtraction operation and store the result in another composite register.

4.3.2. *Quantum Addition.* In the classical paradigm, addition is performed using logic gate network which is irreversible. But in the quantum paradigm addition must be reversible, so we use quantum addition as described in [41] and [26].

Consider two quantum registers $R_1$ and $R_2$. $R_1$ contains $x_1$ and $R_2$ contains $x_2$. The $i$th qubit of $R_1$ is denoted by $x_{1_i}$ and that for $R_2$ is $x_{2_i}$.
To realize addition, we compute carry $c_i$ as

$$c_i = (x_{1_i}\ AND\ x_{2_i})\ OR\ (c_{i-1}\ AND\ (x_{1_i}\ OR\ x_{2_i}))$$

and the sum (result) is computed as

$$s_i = x_{1_i}\ XOR\ x_{2_i}\ XOR\ c_{i-1}.$$

Subsequently we reverse all these operations in order to restore every qubit of the temporary register to its initial state 0 as we use this network for repeated addition.

Therefore, the quantum addition operation can be done by the following operation

$$|x_{1_i}\rangle|x_{2_i}\rangle|0\rangle \rightarrow |x_{1_i}\rangle|x_{2_i}\rangle|x_{1_i} + x_{2_i}\rangle$$

and its quantum circuit is shown in Fig. 4(b). It has two parts, one is for sum computation and other is for carry computation. The sum computation module accepts three inputs and produces three outputs. Here sum is generated by an XOR operation with all inputs. The carry computation module accepts four inputs and produces four outputs resulting from the operations shown in Fig. 4(b).

This quantum addition scheme is defined for two scalars. To perform quantum addition of two vectors, we have to store these two vectors in two composite quantum registers and for each component we have to perform the quantum addition operation. The results are then stored in another composite quantum register.

4.3.3. *Quantum Squaring.* In case of quantum squaring the computation algorithm must be reversible, so we use the procedure as described in [41]. Square of a number can be computed by simple multiplication operation between two such numbers. And multiplication is equivalent to repeated additions.

The quantum squaring of a number can be performed by the following operations:

Suppose $x$ is stored in a quantum register $R$ of $n$ qubits and we try to compute $x^2$. We can write $x$ as $x = 2^0x_0 + 2^1x_1 + \cdots + 2^{n-1}x_{n-1}$. Let us denote the $i$th qubit of $R$ as $x_i$. Initially, the quantum register is in state $|0\rangle$. The number $2^ix_i$ is added conditionally depending on the $i$th qubit $x_i$ and another control bit $c$ (Fig. 5). In

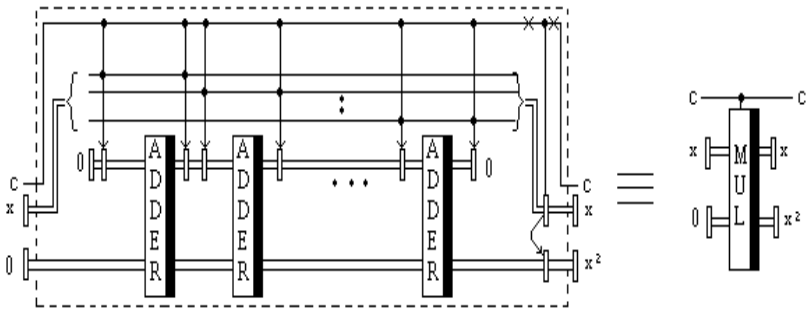


FIGURE 5. Quantum multiplication operation.

other words we implement the following.

$$|c\rangle|x\rangle|0\rangle = \begin{cases} |c\rangle|x\rangle|x \times x\rangle & \text{if } |c\rangle = |1\rangle \\ |c\rangle|x\rangle|x\rangle & \text{if } |c\rangle = |0\rangle \end{cases} \tag{4.2}$$

If $|c\rangle|x_i\rangle = |1\rangle|1\rangle$ then the $i$th qubit of the register is loaded with $2^i x$, otherwise with 0. To do this we use a Taffoli gate in which $|c\rangle$ and $|x_i\rangle$ act as control bits. If $|c\rangle = |0\rangle$ only 0 is added at each stage to the result register. But as the desired state must be $|c\rangle|x\rangle|x\rangle$, so a copy operation is performed. The copy operation is implemented by the rightmost elements of the network as shown in Fig. 5. The conditional copy is implemented using an array of Taffoli gates as shown in Fig. 2.

The above squaring operation is performed for a scalar $x$. If we want to compute the squaring operation on different components of a vector, then the vector data point must be stored in a composite quantum register and for each component we use the squaring operation and store the results in another composite quantum register.

4.4. **Quantum Minimum Square Distance Computation.** For conventional computer there are many algorithms to find minimum from a given set of scalars. In quantum computing paradigm to find the minimum of a set of scalars an algorithm is suggested in [20]. The minimum computation is basically a searching operation

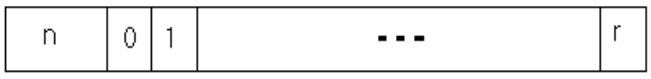


FIGURE 6. Data register used in minimum computation.

on a given scalar data set. Suppose, $N$ scalars are stored in an array of registers in unsorted manner. Registers are composite type and shown in Fig. 6. Register consists two parts, in one part data index is stored and in other part the value is stored. The data index part consists of $n$ qubits where $n$ is the smallest integer such that $N \geq 2^n$ and the other part has $r$ qubits (this will depend on the precision required). So each register consists of $(n+r)$ qubits. We denote this set of register by $T$, i.e., if we write


FIGURE 7. Integer register for storing $N$

$T[i]$, then it indicate $i$th data. The quantum search algorithm [13, 14] will find the index $i$ such that $T[i]$ is minimum.

4.4.1. *Quantum Search.* The minimum computation [20] is described in Algorithm 3.

**Algorithm 3.** Determination of minimum item and its index.
**Input**: Data items and their index
**Output**: Minimum item and its index

**Step 1**: Initialize the threshold index $0 \leq \theta \leq N-1$ using an uniformly distributed random number generator.

**Step 2.1**: Set the register as $\sum_{j=0}^{N-1} \frac{1}{\sqrt{N}}|j\rangle|\theta\rangle$, $\forall\ j = 0, 1, \ldots, N-1$ and mark every register $j$ for which $T[j] < T[\theta]$, where $T$ denotes set of register.

**Step 2.2**: Perform the quantum search algorithm as in [13].

**Step 2.3**: Measure the first register: let $\theta'$ be the result.
If $T[\theta'] < T[\theta]$ then threshold index change to $\theta'$.

**Step 3**: Return $\theta$ as the desired index for minimum item.

**Step 4**: [**Termination**] Stop

4.5. **Quantum Mean Computation.** In the computation of the mean of $N$ numbers, first we add these numbers and then divide that sum by $N$. For the quantum computing paradigm we propose a new algorithm to find mean of a given set of scalars. In this algorithm we use "discrete logarithm algorithm" [38] and "exponentiation algorithm" [41] as subroutines. To store the real data we need registers. We use the same type of register as shown in Fig. 6. Again to store the value of $N$ we need an integer type register. This register will be composite type and one part consists of one qubit by which we identify it and other part stores the value of $N$. There is $n$ qubits in this part, where $n$ is smallest integer such that $N \geq 2^n$. The register is shown in Fig. 7.

The mean of $N$ real values computation is described in Algorithm 4.

**Algorithm 4.** Quantum mean computation.
**Input**: $N$ real values.

**Output**: Mean

**Step 1**: Compute $S$, the sum of $N$ real values. This can be done by addition algorithm [41, 26].

**Step 2**: Compute $\log_2 S$ and $\log_2 N$ using "discrete logarithm" algorithm described next.

**Step 3**: Compute $\log_2 S - \log_2 N$ and round off the result into integer.

**Step 4**: Compute antilogarithm of $\log_2 S - \log_2 N$ using "exponential algorithm" described later.

**Step 5**: [**Termination**] Stop

We have used two algorithms, "discrete logarithm" and "exponential algorithm". Now we describe them one after another.

4.5.1. *Quantum Discrete Logarithm.* In quantum computing paradigm we compute discrete logarithm using the procedure suggested in [38]. The discrete logarithm problem can be described as follows: given, a real number $s$, we find another real number $\gamma_1$ such that $s = 2^{\gamma_1}$. Then $\gamma_1$ is the logarithm of $s$ with respect to the base 2. The discrete logarithm algorithm uses two exponentiations and two quantum Fourier transforms. Inputs for the algorithm are $y_1$,$y_2$ and $f(y_1, y_2) = s^{y_1} 2^{y_2}$ is computed reversibly. We store this inputs into three registers. To store $y_1$ and $y_2$ we need registers with $O\left([\log t] + \log\left(\frac{1}{\epsilon}\right)\right)$ qubits where $t$ is the period of the function and $\epsilon$ is the precision required. The size of the third register depends on the desired precision of the function. All these registers are initially set to zero.

The procedure is as follows.

**Algorithm 5.** Quantum discrete logarithm.

**Input**: $y_1$, $y_2$

**Output**: $\gamma_1$, $\gamma_2$

**Step 1**: Set the initial state of all three registers to 0, i.e., $|0\rangle|0\rangle|0\rangle$.

**Step 2**: In the first two registers, put $y_1$ and $y_2$ and create superposition using Walsh-Hadamard gate. So the contents of the three registers become
$\frac{1}{2^\alpha} \sum_{y_1=0}^{2^\alpha-1} \sum_{y_2=0}^{2^\alpha-1} |y_1\rangle|y_2\rangle|0\rangle$

**Step 3**: Compute function $f(y_1, y_2)$ and store into the third register. Hence, contents of the three registers become
$\frac{1}{2^\alpha} \sum_{y_1=0}^{2^\alpha-1} \sum_{y_2=0}^{2^\alpha-1} |y_1\rangle|y_2\rangle|f(y_1, y_2)\rangle$.

**Step 4**: Apply Fourier transform [38] on the third register and contents of the register now changed to
$\frac{1}{2^\alpha\sqrt{\gamma_2}} \sum_{\beta=0}^{\gamma_2-1} \sum_{y_1=0}^{2^\alpha-1} \sum_{y_2=0}^{2^\alpha-1} e^{2\pi j \frac{(\gamma_1\beta y_1 + \beta y_2)}{\gamma_2}} |y_1\rangle|y_2\rangle|F(\gamma_1\beta, \beta)\rangle$
$= \frac{1}{2^\alpha\sqrt{\gamma_2}} \sum_{\beta=0}^{\gamma_2-1} [\sum_{y_1=0}^{2^\alpha-1} e^{2\pi j \frac{(\gamma_1\beta y_1)}{\gamma_2}} |y_1\rangle][\sum_{y_2=0}^{2^\alpha-1} e^{2\pi j \frac{(\beta y_2)}{\gamma_2}} |y_2\rangle]|F(\gamma_1\beta, \beta)\rangle$

**Step 5**: Apply inverse Fourier transform on the first and second registers. The contents of the three registers become
$\frac{1}{\sqrt{\gamma_2}} \sum_{\beta=0}^{\gamma_2-1} |\widetilde{\frac{\gamma_1\beta}{\gamma_2}}\rangle |\widetilde{\frac{\beta}{\gamma_2}}\rangle |F(\gamma_1\beta, \beta)\rangle$
where $\widetilde{A}$ denotes a good approximation of $A$.

**Step 6**: Measure the first two registers and from the first register get $\widetilde{\frac{\gamma_1\beta}{\gamma_2}}$ and from the second register get $\widetilde{\frac{\beta}{\gamma_2}}$.

**Step 7**: Apply generalized continued fraction method [38] to compute $\gamma_1$, $\gamma_1$ is the desired logarithm of a number.

**Step 8**: [**Termination**] Stop

In the above algorithm we use Fourier transform and inverse Fourier transform. So we first describe how Fourier transform is performed in quantum computation domain.

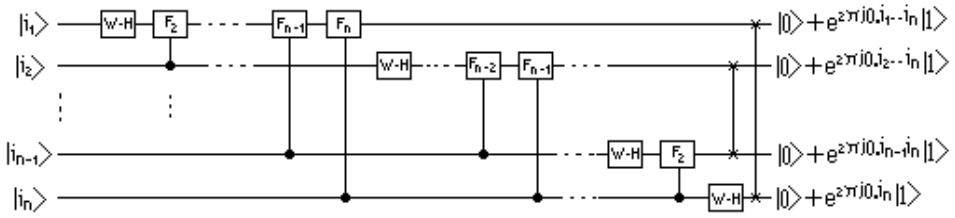


FIGURE 8. Quantum Fourier transform operation.

4.6. **Quantum Fourier Transform** [38]**.** First we consider the classical paradigm. Given $x_0, x_1, \ldots, x_{N-1}$, each $x_i$ a complex number the transforms $y_0, y_1, \ldots, y_{N-1}$ are given by $y_k \equiv \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} x_i e^{\frac{2\pi jik}{N}}$, $k = 0, 1, \ldots, N-1$.

The quantum discrete Fourier transform is the same transform, but the computation is done in a different manner. We consider a set of orthogonal basis $|0\rangle, |1\rangle, \ldots,$ $|N-1\rangle$ where $i$th basis $|i\rangle$ is $|i\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{2\pi jik}{N}} |k\rangle$.

Equivalently, for arbitrary state we may write $\sum_{i=0}^{N-1} x_i|i\rangle \rightarrow \sum_{k=0}^{N-1} y_k|k\rangle$. For simplicity we consider $N = 2^n$, where $n$ is some integer and the basis $|0\rangle, |1\rangle, \ldots, |2^{n-1}\rangle$ are the computational basis for a $n$ qubits quantum computer. So, $i = i_1 i_2 \ldots i_n$ or $i = i_1 2^{n-1} + i_2 2^{n-2} + \cdots + i_n 2^0$.

Now we do a little manipulation to rearrange the relation to get a simpler structure that can be computed by gate structure.

$$
\begin{aligned}
|i\rangle &\rightarrow \frac{1}{2^{\frac{n}{2}}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi jik}{2^n}} |k\rangle \\
&= \frac{1}{2^{\frac{n}{2}}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi ji \left( \sum_{m=1}^{n} k_m 2^{-m} \right)} |k_1 \cdots k_n\rangle
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{2^{\frac{n}{2}}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{m=1}^{n} e^{2\pi j i k_m 2^{-m}} |k_m\rangle \\
&= \frac{1}{2^{\frac{n}{2}}} \bigotimes_{m=1}^{n} \left[ \sum_{k_m=0}^{1} e^{2\pi j i k_m 2^{-m}} |k_m\rangle \right] \\
&= \frac{1}{2^{\frac{n}{2}}} \bigotimes_{m=1}^{n} [|0\rangle + e^{2\pi j i 2^{-m}} |1\rangle] \\
&= \frac{(|0\rangle + e^{2\pi j O.i_n}|1\rangle)(|0\rangle + e^{2\pi j O.i_{n-1} i_n}|1\rangle) \cdots (|0\rangle + e^{2\pi j O.i_1 i_2 \cdots i_n}|1\rangle)}{2^{\frac{n}{2}}}
\end{aligned}
$$

In the above expression we use the notation $0 \cdot i_m i_{m-1} \cdots i_n$ to represent the binary fraction $\frac{i_m}{2} + \frac{i_{m+1}}{4} + \cdots + \frac{i_n}{2^{n-m+1}}$.

The above expression can be implemented by gate structure and it is shown in Fig. 8. In Figure 8 we use $F_k$ and the matrix associated with it is

$$F_k \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi j}{2^k}} \end{pmatrix}.$$

$F_k$ expression can be implemented easily by phase gates. Above we computed Fourier transform for scalar numbers. If the data points are in $p$ dimensions, then for each component we have to perform the same operation. The data must be stored in a composite quantum register and the result of the operation will also be stored in another composite quantum register.

4.7. **Quantum Inverse Fourier Transform.** The inverse Fourier transform is the reverse action of the Fourier transform and similarly we can write as

$$|i\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{-2\pi j i k}{N}} |k\rangle.$$

We propose a gate structure of the circuit to perform the inverse Fourier transform is shown in Fig. 9. Here we use gate IF which is a combination of two gates, one is $Z$ and other is $F$. The unitary operator corresponding to each gate is shown below:

$$Z \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The operator $F_k$ is the same as used with the Fourier transform algorithm. If we want to compute inverse Fourier transform operation for a sequence of vectors then the vector quantity must be stored in a composite quantum register and we need to perform the above mentioned operation for every component.

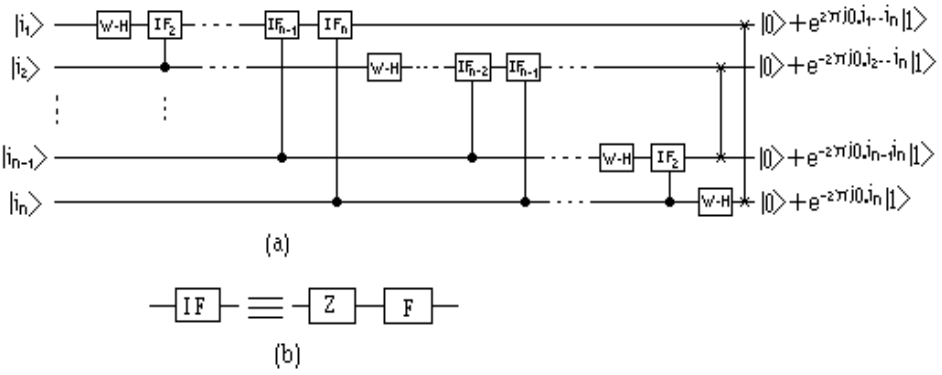


FIGURE 9. Quantum inverse Fourier transform operation.

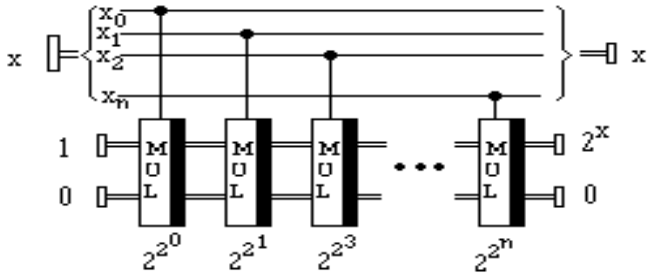


FIGURE 10. Quantum exponential operation.

4.8. **Quantum Exponentiation Algorithm.** To compute antilogarithm in quantum computing paradigm we follow the procedure described in [41]. Suppose $x$ is stored in a quantum register $R$ and our objective is computation of $2^x$. We define the $i$th qubit of $R$ as $x_i$. We can write a number $x$ in binary form as $x = 2^0x_0+2^1x_1+2^2x_2+\cdots+2^nx_n$. So $2^x$ can be written as $2^x = 2^{2^0x_0}\cdot 2^{2^1x_1}\cdot 2^{2^2x_2}\cdots 2^{2^nx_n}$. As a result exponentiation can be computed by setting initially the result register to $|1\rangle$ and computing $n$ multiplication by $2^{2^i}$ depending on the value of the qubit $|x_i\rangle$. The whole operation is as follows.

$$|2^{2^0x_0+\cdots+2^{i-1}x_{i-1}}\rangle|0\rangle \rightarrow \begin{cases} |2^{2^0x_0+\cdots+2^{i-1}x_{i-1}}\rangle|2^{2^0x_0+\cdots+2^{i-1}x_{i-1}}\rangle & \text{if } |x_i\rangle{=}0 \\ |2^{2^0x_0+\cdots+2^{i-1}x_{i-1}}\rangle|2^{2^0x_0+\cdots+2^ix_i}\rangle & \text{if } |x_i\rangle{=}1 \end{cases}$$

The above operation can be implemented by the gate structure shown in Fig. 10.

We have devised algorithm for computation of mean of $N$ number of real values. If we want to compute the mean of $N$ data points in $R^p$, then the algorithm has to be applied $p$ times to compute the mean vector.

Now we describe the quantum mechanical $K$-means algorithm.

4.9. **Quantum Mechanical $K$-means Algorithm.** The quantum mechanical version of $K$-means algorithm is almost similar to the classical $K$-means algorithm which

is described in Section 2 but the representation of data and operations applied on the data are different. The algorithm is described below.

**Algorithm 6.** Quantum $K$-means clustering.

**Input**: $N$ data points, $X_1$, $i = 1, 2, \cdots, N$

**Output**: $K$ centroides, $\mathbf{v}_i$ for $i = 1, 2, \cdots, K$.

**Step 1**: Set the data points into the registers. Set the first $k$ qubits of every register to zero, next $n$ qubits store data index and next every $r$ qubits represent a component of a data point. The registers are represented as

$$\sum_{i=0}^{N-1}\sum_{j=0}^{K-1}\sum_{l=0}^{p-1}|i\rangle|j\rangle|x_l\rangle.$$

**Step 2**: Randomly set $K$ number of centroides in $K$ registers. This set of registers will be denoted by $\mathbf{v}_i^{old}; i = 1, 2, \cdots, K$. Also set all $K$ registers for the new centroides $\mathbf{v}_i^{new}; i = 1, 2, \cdots, K$ to zero. The first $k$ qubits of $\mathbf{v}_i^{old}$ and $\mathbf{v}_i^{new}$ store the value of $i$.

**Step 3**: Repeat

**Step 3.1**: For $j = 1, 2, \cdots, k$ set the first $k$ qubits of the integer register $IR_j$ to $j$ and the remaining $n$ qubits to zero.

**Step 3.2**: for i=1 to N

**Step 3.2.1**: for j=1 to K

Compute squared distance between ($\mathbf{x}_i$ and $\mathbf{v}_j^{old}$).

Compute minimum of the $K$-squared distances.

The index of closest centre is marked in the first $k$-qubits of the data point $\mathbf{x}_i$.

Add $\mathbf{x}_i$ to $\mathbf{v}_l^{new}$ when the first $k$-qubits of data register for $\mathbf{x}_i$ contains $l$.

Add 1 to the integer register $IR_l$.

**Step 3.3**: For $i = 1$ to $K$

$\mathbf{v}_i^{new} = \frac{\mathbf{v}_i^{old}}{IR_i}$.

**Step 3.4**: $\epsilon = ||V^{old} - V^{new}||$

**Step 3.5**: $V^{old} \leftarrow V^{new}$.

**Step 4**: Until ($\epsilon < \epsilon_0$)

**Step 5**: [**Termination**] Stop

The above algorithm requires three major computations: squared distance, minimum and mean. All these operations can be performed on the data points efficiently if we store these properly. We have already discussed in Section 4.1 how we store the data points. Finding of the squared distance between two points is discussed in Section 4.3. Minimum squared distance can be found using Algorithm 3 described in Section 4.4. After finding the minimum squared distance we add the data points to

the appropriate centroid. To find the new cluster centroid we just divide $\mathbf{v}_i^{new}$ by the corresponding integer register($IR_i$).

4.10. **Analysis of the Algorithm.** The time complexity for this quantum K-means clustering algorithm depends on the complexity of the different modules such as (1) squared distance computation, (2) minimum computation, and (3) mean computation using quantum computing techniques.

In squared distance computation we have performed subtraction, squaring and addition operations serially. As we have designed a quantum network to perform these operations, so operations will take constant time. We have considered a set of data $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ in $p$ dimensions. For each components we have to compute subtraction and squaring operations. The computation of squared distance takes $K$ operations. Time requirement for the computation of minimum squared distance is $O(\sqrt{K})$ as described in [20]. The time requirement for the process of updating the cluster is $N[K + O(\sqrt{K})]$. Mean computation consists of discrete logarithm and exponentiation. Exponentiation takes constant time as it is performed through gate structure. Discrete logarithm computation time depends on the computation of Fourier transform and inverse Fourier transform. Since these operations are performed through gate structures, time requirement is also constant. The time requirement for updating centroides is $N + K$. Hence time complexity for an iteration is $N[K + O(\sqrt{K})] + N + K = O(NK)$. If the algorithm converges after $L$ number of iterations then the time complexity for the algorithm is $O(LNK)$.

## 5. Conclusion

In this paper a $K$-means clustering algorithm using quantum computing technique is proposed. Unlike traditional $K$-means clustering, which takes $O(LNpK)$ time, its quantum mechanical version takes only $O(LNK)$ time. This could save a lot of time for data mining type applications where often dimension of the data is on the order of hundreds. The proposed algorithm can be improved if we use exponential search [14] algorithm. In this case minimum finding will take only $O(2(\lceil \log_4 K \rceil))$ iterations. However, building quantum gates or quantum computer is a difficult task. There are five different techniques namely, simple harmonic oscillator, photons and nonlinear optical media, ion trap, NMR and cavity quantum electrodynamics. These are of course, still in the experimental stage.

## REFERENCES

[1] G. H. Ball and D. J. Hall (1967). A clustering technique for summarizing multivariate data. *Behavioral Science*, **12**, pp. 153–155.

[2] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin and H. Weinfurther (1995). Elementary gates for quantum computation, *Phys. Rev. A.*, **52**, pp. 3457–3467.

[3] P. Benioff (1980). The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines.*Journal of Stat. Phys.*, **22**, pp. 563–591.

[4] P. Benioff (1982). Quantum mechnical Hamiltonian models of Turing machines, *J. Statist. Phys.*, **29**, pp. 515–546.

[5] P. Benioff (1982). Quantum mechnical Hamiltonian models of Turing machines that dissipate no energy, *Phys. Rev. Letter*, **48**, pp. 1581–1585.

[6] C. H. Bennett (1973). Logical reversibility of computation, *IBM J. Res. Develop.*, **17**, pp. 525–532.

[7] C. H. Bennett (1989). Time/space trade-offs for reversible computation, *SIAM J. Comput.*, **18**, pp. 766–776.

[8] E. Bernstein and U. Vazirani (1993). Quantum complexity theory, *Proc.* $25^{th}$ *Annual ACM Symposium on Theory of Computing*, Association for computing Machinery, New York, pp. 11–20.

[9] A. Berthiaume and G. Brassad (1992). The quantum challenge to structural complexity theory, *Proc. Seventh Annual Structure in Complexity theory conference*, IEEE comp. Scociety Press, Los Alamitos, CA, pp. 132–137.

[10] A. Berthiaume and G. Brassad (1994). Oracle quantum computing, *J. Mod. Optics*, **41**, pp. 2521–2535.

[11] M. Biafore (1994). Can quantum computers have simple Hamiltonians, *Proc. Workshop on Physics Computation: PhysComp '94, IEEE Computer Society Press*, Los Alamitos, CA, pp. 63–68.

[12] P. van Emde Boas (1990). Machine models and simulations, in Handbook of theoretical computer science vol. A. J. van Leeuwen ed. Elsevier, Amsterdam, pp. 1–66.

[13] M. Boyer, G. Brassard, P. Hoyer and A. Tapp (1998). Tight bounds on quantum searching. *arXiv:quant-ph/9605034*.

[14] G. Chen and Z. Diao (2000). Exponentially fast quantum search algorithm. *arXiv:quant-ph/0011109*.

[15] D. Deutsch (1985). Quantum theory, the Church-Turing principle and universal quantum computer, *Proc. Roy. Soc. London Ser. A.*, **400**, pp. 96–117.

[16] D. Deutsch, A. Barenco and A. Ekert (1995). Universality of quantum computation, *Proc. Roy. Soc. London. Ser. A.*, **449**, pp. 669–677.

[17] D. Deutsch and R. Jozra (1992). Rapid solution of problems by quantum computation, *Proc. Roy. Soc. London Ser. A.*, **439**, pp. 553–558.

[18] D. P. DiVincenzo (1995). Two-bit gates are universal for quantum computations, *Phys. Rev. A.*, **51**, pp. 1015–1022.

[19] R. O. Dudo and P.E. Hart (1973). Pattern classification and Scene analysis, *John Wiley and Sons*.

[20] C. Durr and P. Hoyer (1996). A quantum algorithm for finding the minimum. *arXiv:quant-ph/9607014*.

[21] R. P. Feynman (1982). Simulating physics with computers. *Intl. J. Of Theoretical Physics*, **21**, pp. 467.

[22] R. P. Feynman (1986). Quantum mechanical computers, *Found. Phys.*, **16**, pp. 507–531.

[23] L. K. Grover (1996). A fast quantum mechanical algorithm for database search. *Proceedings 28th Annual Symposium on the Theory of Computing (STOC)*, pp. 212–219.

[24] M. Hirvensalo (2002). Quantum computing - Facts and folklore, *Natural Computing*, **1**, pp. 135–155.

[25] M. Hirvensalo (2001). Quantum computing. *Springer*.

[26] P. K. Koley and S. Pal, Comments on "Quantum networks for elementary arithmetic operations", *Neural, Parallel and Scientific Computation* 20:11–16, 2012.

[27] Y. Lecref (1963). Machines de Turing reversibles. Recursive insolubilite en $n \in N$ de l'equation $u = \theta^n u$, ou $\theta$ est un isomorphisme de codes. *C. R. Acad. Francaise Sci.,* **257**, pp. 2597–2600.

[28] R. Y. Levine and A. T. Sherman (1990). A note on Bennett;s time-space tradeoff for reversible computations, *SIAM J. Comput.*, **19**, pp. 673–677.

[29] S. Lloyd (1993). A potentially realizable quantum computer. *Science*, **261**, pp. 1569–1571.

[30] S. Lloyd (1994). Envisioning a quantum supercomputer, *Science*, **263**, pp. 695.

[31] S. Lloyd (1995). Almost any quantum logic gate is universal. *Phys. Rev. Lett.*, **75**, pp. 346–349.

[32] J. B. MacQueen (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Barkeley Symposium*, **2**, pp. 281–297.

[33] N. Margolus (1986). Quantum computation. *Ann. New York Acad. Sci.*, **480**, pp. 346–349.

[34] N. Margolus (1990). Parallel quantum computation. *Complexity, Entropy and the Physics of Information, Santa Fe Institute Studies in the Sciences of Complexity*, **Vol. VIII**, W. H. Zurek, ed., Addision Wesley, Reading, MA, pp. 273–287.

[35] M. A. Nielsen and I. L. Chuang (2000). Quantum computation and quantum information. Cambridge University Press.

[36] S. Z. Selim and M. A. Ismail (1984). K-means type algorithms: A generalized Convergence theorem and characterization of local optimality. *IEEE Tr. on PAMI*, vol.**6(1)**, pp. 81–87.

[37] D. Simon (1994). On the power of quantum computation, *Proc.* $35^{th}$ *Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, pp. 116–123.

[38] P. W. Shor (1994). Algorithm for quantum computation: discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Fundamentals of Computer Science (FOCS)*, pp. 124–134.

[39] K. Steiglitz (1988). Two non-standard paradigms for computation: Analog machines and cellular automata, in Performance Limits in Communication Theory and practice. *Proc. NATO Advanced Study Institute, Il Ciocco, Castelvecchio Pascoli, Tuscany, Italy*, J.K. Skwirzynski, ed., Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 173–192.

[40] T. Taffoli (1980). Reversible computing, an Automata, Language and Programming, *Seventh colloquium, Lecture notes in computer science*, **84**, J.W. de Bakker and J. van Leeuwen eds., Springer-Verlag, Berlin, pp. 632–644.

[41] V. Vedral, A. Barenco and A. Ekert (1996). A Quantum Networks for Elementary Arithmatic Operations.*Phys. Rev.* **A**, pp. 147–153.

[42] A. Vergis, K. Steiglitz and B. Dickinson (1986). The complexity of analog computation, *Math. comput. Simulation*, **28**, pp. 91–113.

[43] A. Yao (1993). Quantum circuit complexity, *Proc.* $36^{th}$ *Annual Symposium on Foundations of Computer Science*, IEEE computer society press, Los Alamitos, CA, pp. 352–361.