Received: 7 July 2021 | Revised: 1 November 2021 | Accepted: 13 November 2021

IET Quantum Communication

DOI: 10.1049/qtc2.12029

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Circuit design for clique problem and its implementation on quantum computer

Arpita Sanyal Bhaduri[1] | **Amit Saha**[1,2] | **Banani Saha**[3] | **Amlan Chakrabarti**[1]

[1]A. K. Choudhury School of Information Technology, University of Calcutta, Kolkata, West Bengal, India

[2]Atos, Pune, India

[3]Computer Science and Engineering, University of Calcutta, Kolkata, West Bengal, India

Correspondence

Amit Saha, A. K. Choudhury School of Information Technology, University of Calcutta, Kolkata, West Bengal, India.
Email: abamitsaha@gmail.com

Abstract

Finding cliques in a graph has a wide range of applications due to its pattern matching ability. The k-clique problem, a subset of the clique problem, determines whether or not an arbitrary network has a clique of size k. Modern-day applications include a variation of the k-clique problem that lists all cliques of size k. However, the quantum implementation of such a variation of the k-clique problem has not been addressed yet. In this work, apart from the theoretical solution of such a k-clique problem, practical quantum-gate-based implementation has been addressed using Grover's algorithm. In a classical-quantum hybrid architecture, this approach is extended to build the circuit for the maximum clique problem. Our technique is generalised since the program automatically builds the circuit for any given undirected and unweighted graph and any chosen k. For a small k with regard to a big graph, the proposed solution to addressing the k-clique issue has shown a reduction in qubit cost and circuit depth when compared to the state-of-the-art approach. A framework is also presented for mapping the automated generated circuit for clique problems to quantum devices. Using IBM's Qiskit, an analysis of the experimental results is demonstrated.

KEYWORDS

Grover's algorithm, k-clique problem, maximum clique problem, NISQ devices, quantum circuit synthesis

1 | INTRODUCTION

In the early 1980s, quantum computers were postulated, and in the late 1980s, the description of quantum mechanical computers was formalised. Since the early 1990s, several efforts on quantum computers have developed gradually, as these computers have been proved to be more powerful than conventional computers on a variety of specialised issues, including computationally NP-problems [1, 2]. Several quantum algorithms, for example, **Shor's Algorithm** [3] for factoring integers, **Grover's Algorithm** [4] for searching an unstructured database, **Triangle finding** by Magniez et al. [5], **Matrix Product Verification** [6] have already been proposed and shown asymptotic improvements than their classical counterparts. Another computationally NP problem, the clique problem [7], has been addressed in a quantum environment in this study. The main goal of this paper is to provide an end-to-end framework for automatically implementing a clique problem so that anyone who can map their computational problem to the clique problem in polynomial time can implement it further, even if they have no prior knowledge of gate-based quantum circuit implementation.

A clique is a complete subgraph of an undirected graph in which every distinct vertex in the subgraph is connected to every other vertex through an edge. The k-clique problem is a subset of the clique problem that asks if an arbitrary graph has a clique of size k. A maximum clique, on the other hand, is a complete subgraph of a graph with the biggest size among all other complete subgraphs in the graph. In [8], the authors studied that the clique Problems can have applications in classical communications and signal processing. In the quantum domain, clique problems can also be useful to determine the maximum rate of classical information that can be sent through a quantum communication channel with zero error

Arpita Sanyal Bhaduri is the first author.

and without using entanglement [9]. Clique problems have several applications in various branches of computer science, for instance, pattern recognition [10], information retrieval [11], computer vision [10], analysis of financial networks [12], and spatial data mining [13]. Solving the clique problem especially the k-clique problem using a quantum algorithm [14], where the solution provides a clique of size k, is more efficient in terms of computation as compared to its classical counterpart due to its quantum mechanical properties. Quantum circuit design for the k-Clique problem has also been demonstrated in the literature [14]. Although, circuit designing in quantum setting for a variant of the k-clique problem, where the solution of the problem lists all cliques of size k is much more difficult in reality due to the computational complexity of this variant of the k-clique problem.

In this work, we solve the variant of the k-clique problem, which lists all the cliques of size k using a well-known quantum search algorithm, *i.e.* Grover's algorithm to get the immediate advantage of solving various modern-day applications like community detection [15–17], data mining in bioinformatics [18] and disease classification [19]. The circuit for a specific variant of the k-clique problem has been built in such a way that the engineering difficulty of implementing such a variant of the k-clique problem has been met. We also show that our approach to building the circuit for the k-clique problem outperforms the state-of-the-art approach [14] in terms of quantum cost in terms of the qubit and circuit depth for a modest value of k with a considerably bigger graph. This work also proposes a generalised solution for the maximum clique problem (MCP), which lists all the largest-sized cliques among all the cliques for a given graph, based on the presented approach to solving the k-clique issue.

Our key contributions in this paper can be summarised as follows:

- We propose an automated end-to-end framework for mapping the clique problem to any available quantum computer so that anyone can get the advantage of the gate-based implementation of a quantum algorithm without much prior knowledge.
- We propose an approach to implement the variant of the k-clique problem, where the output of the problem lists all cliques of size k using a quantum search algorithm for the first time to the best of our knowledge.
- Our approach of solving the k-clique problem outperforms the state-of-the-art approach with respect to qubit cost and circuit depth when $n \gg k$, where n is a large number of nodes of a given graph (G ($V = n$, $E = e$)) and k is, respectively, very small.
- We exemplify the triangle finding problem, which is an example of the k-clique problem, where $k = 3$ to establish our claim.
- Further, we extend our approach of solving the k-clique problem to implement the maximum clique problem via classical-quantum hybrid computation.
- We implement the generalised algorithm on arbitrary graph instances and simulated the same in the QASM simulator as well as in real quantum devices (IBMQX architecture [20, 21]) through a python-based programming interface called QISKit [22] with the noise model and study that for different graphs, how the error affects the resultant states further.

The paper has been organised as follows. Section 2 reviews related works in the classical and quantum domain. The general flow of the automated framework for the clique problem is illustrated in Section 3. Our proposed algorithm for the synthesis of the k-clique problem has been discussed in Section 4. The extension of the algorithm for solving MCP has also been depicted in this section. The performance of the circuit is analysed in Section 5. Section 6 deals with experimental results of the circuit generated for different exemplified graphs in the *IBMQ_qasm_simulator* and *IBMQ_16_melbourne*. The complexity of the algorithm has been analysed in Section 7. Finally, the paper concludes with a summary and future scope in Section 8.

2 | BACKGROUND

In this section, we will try to lay out some background knowledge on Quantum computing and clique problem.

2.1 | Preliminaries of quantum computation

The qubit is the quantum equivalent of a classic bit [23]. Instead of working with classical bits, quantum computers, or more precisely quantum processing units (QPUs), use qubits. Qubits are mathematically expressed using Dirac notation. This notation is also known as bra-ket notation. An example of complete bra-ket notation is $\langle\phi||\psi\rangle$, where ϕ represents ket part and ψ represents bra part. A bit is a binary information that can have two possible values: 0 or 1. On the contrary, a qubit can be expressed as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are amplitudes of the state $|0\rangle$ and $|1\rangle$ and $\alpha^2 + \beta^2 = 1$. The states $|0\rangle$ and $|1\rangle$ are the basis states, similar to bit values of 0 and 1 in classical computers. To denote a quantum state, only the ket part of the notation is used. For example, to represent the classical values of 0 and 1 in a quantum state, we can write as $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$, $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$. The column vector represents the amplitude of the quantum state. A quantum circuit is a model for quantum computation, where reversible quantum gates like Hadamard, CNOT, and Toffoli are imposed on qubits to evolve the quantum states towards the solution of a specific quantum algorithm [24]. The quantum gates that are used in the proposed approach is thoroughly discussed in the Appendix.

2.2 | NISQ devices

NISQ devices (Noisy Intermediate Scale Quantum devices) [25] have arisen in recent years that can execute quantum

computation with a short circuit length, but the scale and accuracy are insufficient to perform continuous and effective error correction. NISQ devices use a variety of physical systems, including superconductors, ion traps, quantum dots, NV centres, and optics. For example, IBM Q processors based on superconducting accept gates written in the QASM language [26], which is also used in this article for all of the tests. We used IBM's Melbourne quantum gadget in particular. Figure 1 shows the qubit topology of IBM's Melbourne.

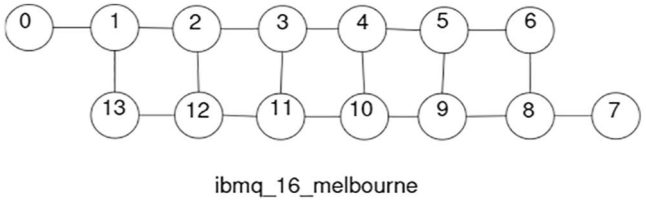

FIGURE 1 Qubit topology [27]

TABLE 1 Gate set for IBM Q devices

Gate type	Remarks
$U1(\lambda)$	No pulse. Rotation Z (R_Z) gate.
$U2\ (\phi, \lambda)$	One $\frac{\pi}{2}$ pulse. H Gate is $U2\ (0, \pi)$.
$U3\ (\theta, \phi, \lambda)$	Two $\frac{\pi}{2}$ pulses. $R_Y(\theta)$ gate is $U3\ (\theta, 0, 0)$.
CX	Cross-resonance pulses and one $\frac{\pi}{2}$ pulse.

To map onto the IBM Q processor, all multi-controlled logical gates in the resulting logical circuits must be broken into four types of one-qubit and two-qubit gates. Table 1 lists the gate sets and needed pulses for the IBM Q superconducting computers. We can do $U1$ at no expense because no pulse is required. $U3$ has double the error level of $U2$ and is about an order of magnitude less than the CX gate.

The circuit depth grows with the number of Toffoli controls when the MCT gate is implemented on the IBM Q device, as shown in Figure 2. The inaccuracy in quantum circuits grows exponentially as the depth of the circuit increases, as detailed in the Appendix.

2.3 | General hybrid architecture for quantum algorithms

The maximum clique issue is solved in this study using the concept of hybrid quantum and classical computing [28]. A classical algorithm is employed in a hybrid quantum-classical

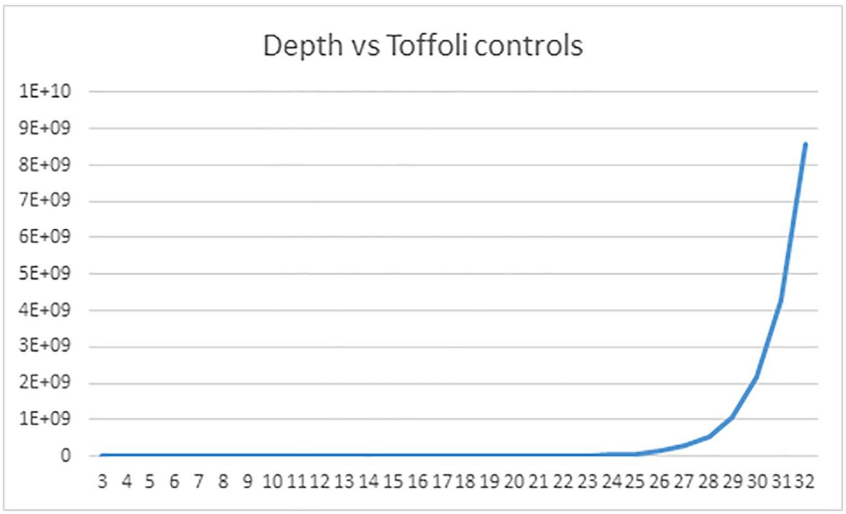

FIGURE 2 Circuit depth versus Toffoli controls on IBM Q: Toffoli controls along with the x axis and depth along with the y axis

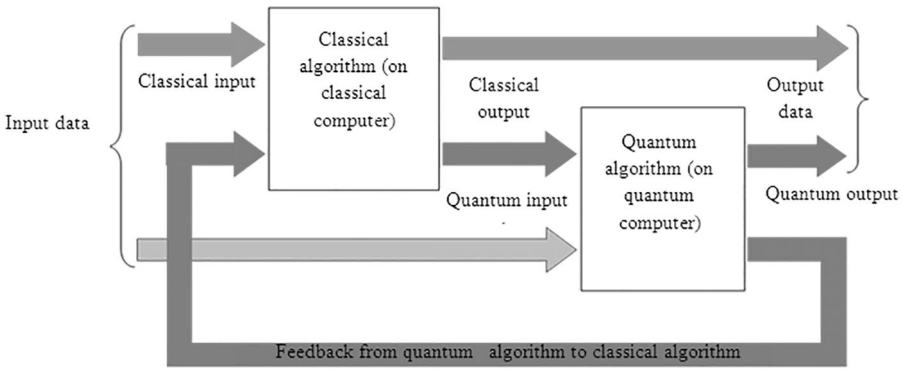

FIGURE 3 The relationship between classical part and quantum part of the hybrid algorithm [28]

architecture to accomplish some basic activities (preparation of inputs, feeding the output to the quantum machine) that the quantum algorithm may require. This architecture's quantum subsystem initialises quantum registers/nodes, prepares inputs using quantum gates, and executes the quantum oracle using quantum gates and unitary transformations. Finally, analyse the results in the quantum registers and send them as feedback to the classical computer for further processing. Figure 3 depicts the hybrid model's overall architecture. Next, we'll go through the core principles of Grover's algorithm and the clique problem, both of which are critical in our suggested strategy.

2.4 | Grover's algorithm

Grover's algorithm [4] is a way of finding an element in an unsorted list with N elements using quantum computers. Grover's algorithm is based on amplitude amplification of the basis state that specifies a position of a searched element in the list. It runs in time $O(\sqrt{N})$, where N is the number of elements in the list. The generalised structure of Grover's algorithm is shown in Figure 4.

Grover's algorithm works with a unitary operator O called the oracle function, which is defined by $|x\rangle|q\rangle \Rightarrow |x\rangle|q \oplus f(x)\rangle$, where $|x\rangle$ is the n qubit index register and $|q\rangle$ is an additional qubit, called the oracle qubit. The functional view of Grover's Search Algorithm is presented here. ∃ function oracle (O) such that

$$O|x\rangle = \begin{cases} -|x\rangle, & \text{if } x \text{ is Marked} \\ |x\rangle, & \text{otherwise} \end{cases}$$

More elaborately, the steps of the Grover's algorithm are as follows:

Initialisation: The algorithm starts with the uniform superposition of all the basis states on input qubits n. The last ancilla qubit is used as an output qubit, which is initialised to $H|1\rangle$. Thus, we obtain the quantum state $|\psi\rangle$.

Sign Flip: Flip the sign of the vectors for which the oracle gives output 1.

Amplitude Amplification: We need to perform the inversion about the average of all amplitudes of the quantum state for a certain number of iterations to make the amplitude of the marked state large enough so that it can be obtained from a measurement with probability close to 1. This phenomenon is known as amplitude amplification, which is performed by using a diffusion operator.

Number of Iterations: Iterations of Grover's algorithm are the number of times that the oracle and amplification stages are performed. Each iteration of the algorithm increases the amplitude of the marked state by $O(\sqrt{1/N})$. In this way, Grover's search algorithm requires $\sqrt{N/M}$ iterations to get the probability of one of the marked states M out of the total N number of states set. If the number of iterations of Grover's algorithm is more than the optimal number, the probability of measuring the desired state actually goes down. Therefore, the right number of iterations is important for getting proper results.

Diffusion operator: This diffusion operator of Grover's algorithm [4] inverts the amplitude of the input states about their mean value of amplitude. The generalised matrix representation of the diffusion operator is shown in Table 2. A six-qubit diffusion operator is also presented in Table 2. As shown in the six-qubit diffusion operator, a 6-qubit Toffoli gate is required. Therefore, for the n-qubit diffusion operator, n-qubit Toffoli gate is needed. This n-qubit Toffoli gate needs to be realised into one-qubit or two-qubit gates as discussed. While decomposing the n-qubit Toffoli gate, if the depth and the ancilla qubits increase arbitrarily, then the time complexity of the algorithm also increases, which is undesirable.

2.5 | Clique problem

This subsection deals with the definition of a clique, k-clique, maximum clique of a graph, and the clique problem in the quantum domain, which are used in the rest of the paper.

2.5.1 | Clique, k-clique and maximum clique

A clique is a complete subgraph of a graph. Particularly, if there is a subset of k vertices that are connected to each

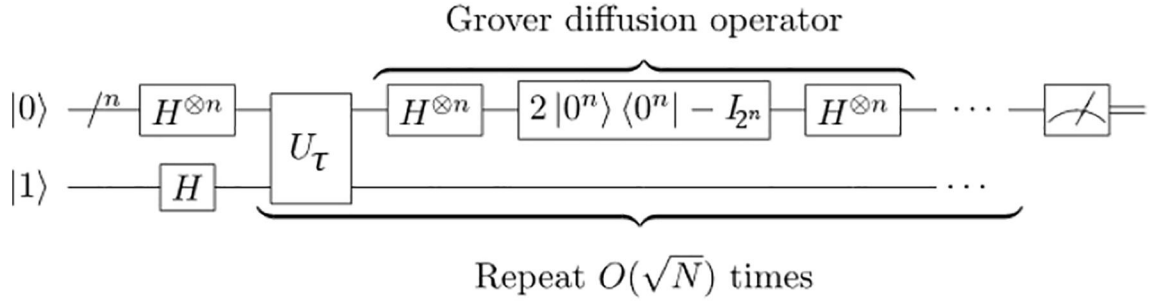

FIGURE 4 Generalised circuit for Grover's algorithm [4]

TABLE 2 Diffusion operator

Six-qubit diffusion circuit	Generalised matrix representation
d1	$\begin{pmatrix} -1+\frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & -1+\frac{2}{N} & \cdots & \frac{2}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \cdots & -1+\frac{2}{N} \end{pmatrix}$

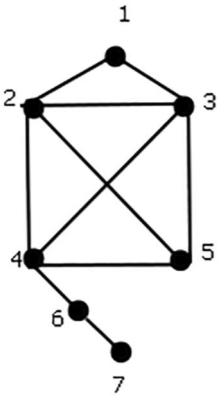

FIGURE 5 An arbitrary graph

other, we state that the graph contains a k-clique. Let $G(V, E)$ be a graph, where V be the set of vertices and E be the set of edges. If $u, v \in E$, then u and v are said to be adjacent. The set of vertices adjacent to a vertex v is called the neighbourhood of v and is denoted by $N(v)$. A clique of a graph G is a set of vertices C in which $u, v \in C \Rightarrow u, v \in E$. We say that the graph contains a k-clique if there is a subset of k vertices that are connected to each other. A maximum clique is a complete subgraph of a graph G, whose size is largest among all other complete subgraphs in G. In the graph of Figure 5, there are six vertices, but available cliques are (234) (352) (543) (425) (2345) and maximum clique is (2345).

2.5.2 | Clique problem in quantum domain

The most commonly studied the clique problem specifically the k-clique problem is the 3-clique problem or triangle finding problem. It is quite trivial to show that the randomised classical query complexity of the 3-clique problem is $O(n^2)$ where n denotes the number of vertices in the graph [29]. Szegedy constructed a quantum algorithm for the 3-clique problem with query complexity $O(n^{\frac{10}{7}})$ [30]. Later, Magniez et al. showed that the 3-clique problem can be solved with improved query complexity $O(n^{\frac{13}{10}})$ using a quantum walk approach [5]. To date, the best known lower bound on the quantum query complexity of the 3-clique problem is $\Omega(n)$ and the best-known algorithm has a complexity of $O(n^{\frac{5}{4}})$ [31, 32]. Further, using quantum algorithms, some research works have also been done for the k-clique problem, when $k > 3$ [5, 14]. Among all these works on the k-clique problem, the state-of-the-art work [14] showed that to implement the k-clique problem, at least n number of input qubits are required, when the number of vertices of the given graph is n. In addition, total $\log_2 k + 3$ ancilla qubits are required to perform Grover's algorithm for solving the k-clique problem. In this work, we compare our proposed work with this state-of-the-art work on the k-clique problem.

We also propose a generalised approach to solving MCP with the help of the proposed approach of the k-clique problem. There are some existing works on MCP in quantum computing, which are needed to be discussed. In 2015, Pronaya Prosun Das et al. [33] used the concept of a quantum-inspired evolutionary algorithm proposed by Kuk-Hyun Han et al [34] for solving the MCP. A quantum-inspired evaluation algorithm is a combination of quantum computing and evolutionary algorithm. Elijah Pelofske et al. proposed quantum annealing for solving the MCP [35]. With this background, we are inclined to contribute some research advancement on the clique problem in the quantum domain.

3 | GENERAL FLOW OF PROPOSED AUTOMATED FRAMEWORK FOR MAPPING CLIQUE PROBLEM TO QUANTUM COMPUTERS

Figure 6 depicts the entire flow of the suggested design for transferring the clique problem from a graph to a quantum computer. **Oracle_clique_problem** is a suggested algorithm that accepts an adjacency matrix as input and outputs a QASM gate list. This QASM gate list can be used as an input to a quantum computer to acquire the implementation result of the clique problem after converting the logical gates into quantum computer-supported one-qubit and two-qubit gates and using a qubit mapping algorithm based on qubit topology. In the literature, these MCT realisations [36] and qubit mapping algorithms [37–39] are well defined. We have just included them in our framework as a source of help.

The major contribution of this study is to use Grover's technique to automatically construct the circuit for the clique problem when the adjacency matrix of the graph and the clique size k are known. The diffusion operator's circuit design is generalised, as mentioned in paragraph 2.5, regardless of the computational task. Furthermore, because the oracle is problem-specific, our technique is entirely centred on the construction of oracle circuits for clique problems.

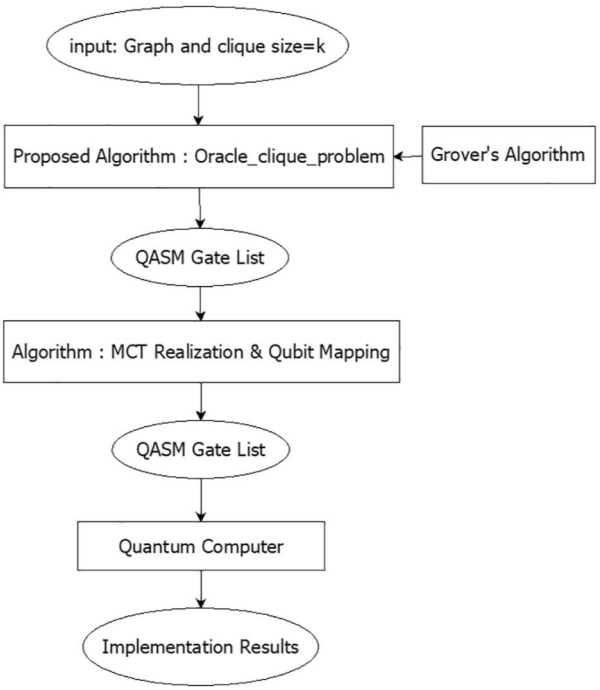

FIGURE 6 The complete flow of the proposed work

4 | PROPOSED METHODOLOGY OF CIRCUIT DESIGN FOR CLIQUE PROBLEM

The proposed methodology for the circuit synthesis of the k-clique problem utilising Grover's search algorithm is outlined in this section. The k-clique problem takes a graph's adjacency matrix as input and finds cliques of size k within the graph. Grover's search algorithm is divided into two components, as indicated in the previous section. The diffusion operator is the same for all issue cases, but the oracle is unique to the search problem at hand. As a result, this work suggests the design of an oracle circuit that accepts a graph as input and checks if it contains a clique of size k. Before applying the diffusion operator to those states that constitute a clique, the oracle marks them. Later in this part, the proposed technique to the k-clique problem is applied to solve MCP. Let us start with the oracle for the k-clique problem that has been proposed.

4.1 | Proposed oracle for k-clique problem

The paper aims to construct the quantum circuit block for the k-clique problem. Theoretically, the oracle is only a function that checks whether a specific item is a target or not. However, due to the linearity of quantum mechanics, when the oracle is applied to the superposition state $|\psi_0\rangle$, all possible items are examined against the criteria. In this generalised circuit, there are seven main steps, which are Initialisation, Hadamard transformation, Qubit activation, Edge detection, Clique detection, Qubit deactivation, and phase flip. For finding k-clique in a graph, all the steps have been described in the following subsections.

4.1.1 | Initialisation

If there are n vertices in the input graph, then the number of qubits required to represent each vertex is $\lceil \log_2 n \rceil$. The oracle checks a combination of k vertices from a combination of all k vertices without duplicate vertices set at a time to determine if a clique is formed by them. Hence, a total of $m = k * \lceil \log_2 n \rceil$ input qubit lines are required to represent a combination of k vertices for input. Total number of ancilla qubits required to verify whether the subgraph formed by k number of vertices is complete or not is $\binom{k}{2}+1$. The initial input qubits include m qubits prepared in the ground state $|\psi\rangle = |0\rangle^{\otimes m}$, $(\binom{k}{2}+1)$ ancilla qubits in the ground state $|\theta\rangle = |0\rangle^{\otimes \binom{k}{2}+1}$ (These $(\binom{k}{2}+1)$ ancilla qubits are required to prepare clique detector block, which is described in the next subsection thoroughly) and one output qubit in the excited state $|\phi\rangle = |1\rangle$, which is required to perform the CNOT operation of the oracle. This entire initialisation can be mathematically written as:

$$|\psi\rangle \otimes |\theta\rangle \otimes |\phi\rangle = |0\rangle^{\otimes m} \otimes |0\rangle^{\otimes \binom{k}{2}+1} \otimes |1\rangle$$

4.1.2 | Hadamard transformation

After the initialisation, the Hadamard transform $H^{\otimes m}$ on input qubits and H on output qubit is performed; therefore, all possible states are superposed as $|\psi_0\rangle \otimes |\theta_0\rangle \otimes |\phi_0\rangle$, where

$$|\psi_0\rangle = \frac{1}{\sqrt{2^m}} \sum_{i=0}^{2^m-1} |i\rangle$$

$$|\theta_0\rangle = |0000\ldots0\rangle$$

$$|\phi_0\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle - |1\rangle\Big)$$

4.1.3 | Qubit activation

For a given graph, vertices need to be numbered as {0, 1, 2… $n-1$}. The input qubit lines act as the binary representation of the combination of k vertices. After Hadamard transformation on input qubit lines, $|\psi_0\rangle$ represents all possible combinations of k vertices of a given graph. But, the oracle checks only the combination of k vertices without any duplicate entry of vertices to determine if a clique exists or not. After Hadamard transformation, we get a superposition of all the states that includes several invalid states. So, we have reduced the search space by excluding all the invalid vertex combinations that do not contribute to a clique. Only valid vertex combinations that may

form a clique are considered using a technique named Qubit Activation. To make sure that the oracle is checking only all possible combinations of k vertices without duplicate entry, all the input qubit lines are needed to be in the excited state $|1\rangle$ for those particular combinations of k vertices to make the input qubit lines suitable as control lines for Multi Control Toffoli (MCT) operation. Some NOT gates have to be imposed on the input qubit lines, which are in the ground state $|0\rangle$ for every possible combination of k vertices without duplicate entry. The qubit activation block activates a qubit by applying the NOT gate if it is 0 to make the desired inputs a string of 1's.

4.1.4 | Clique detector block

The proposed Clique Detector Block is shown in Figure 7. This block is defined as follows:

$$CliqueDetector(v_1, v_2, v_3, \ldots, v_k, f) = \begin{cases} f = 1 & \text{if } v_1, v_2, v_3 \ldots v_k \text{ form a clique} \\ f = 0 & \text{otherwise} \end{cases}$$

where v_1, v_2, v_3, … , v_k are the combination of k vertices of the input graph, which are activated input qubit lines and the f is the circuit output of the Clique Detector Block, which is represented by the $\left(\binom{k}{2}+1\right)^{st}$ ancilla qubit line. Then, the Edge Detector Block imposes if a pair of vertices of input graph is adjacent. This edge detecting sub-circuit is defined next.

4.1.5 | Edge detector block

The function of the Edge Detector Block is to detect edges between pair of vertices. A Multi-controlled Toffoli gate is applied for checking the connectivity between a pair of vertices. One MCT gate indicates the presence of an edge between a pair of vertices. So for detecting multiple edges, multiple MCT gates are needed.

$$EdgeDetector(v_1, v_2, f) = \begin{cases} f = 1 & \text{if } v_1 \text{ and } v_2 \text{ are adjacent} \\ f = 0 & \text{otherwise} \end{cases}$$

This sub-circuit Edge Detector Block checks if two vertices are adjacent and change the state of the output line to $|1\rangle$. First, $\binom{k}{2}$ ancilla lines are the representation of these output lines. If this is true to all $\binom{k}{2}$ ancilla lines, the Clique Detector confirms the presence of a clique in the form of $|1\rangle$ state on the $\left(\binom{k}{2}+1\right)^{st}$ ancilla line, otherwise the state of $\left(\binom{k}{2}+1\right)^{st}$ ancilla line remain same as its initial state. Therefore, if there exists a clique, the ancilla qubit state becomes $|\theta_1\rangle$ for respective input vertices.

$$|\theta_1\rangle = |1111\ldots1\rangle$$

Clique detection block detects a clique by applying the Toffoli gate between the outputs of edge detection blocks.

4.1.6 | Qubit deactivation

A number of NOT gates have to be imposed on the input qubit lines in the reverse order to deactivate the qubits again or to reset the input qubits to their initial value.

4.1.7 | CNOT operation

The output qubit state $|\phi_0\rangle$ is initially set as $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Applying an CNOT gate on output line considering $\left(\binom{k}{2}+1\right)^{st}$ ancilla qubit as control results in an eigenvalue

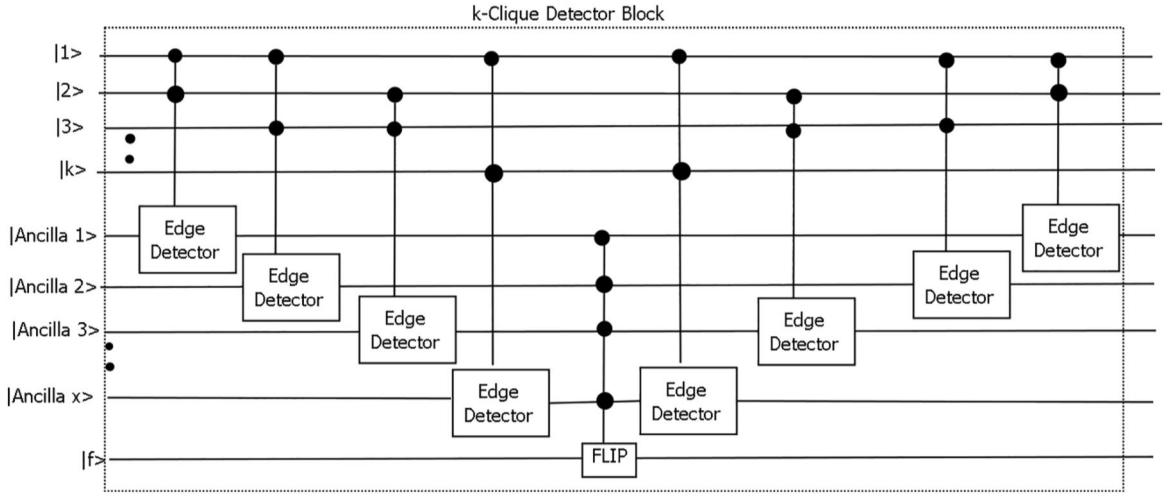

FIGURE 7 Circuit representation of the k-clique detector block

kickback −1, which causes a phase shift for the respective input state, which makes a clique. Figure 8 shows the generalised view of the oracular circuit for the *k*-clique problem.

4.2 | Generalised algorithm for solving *k*-clique problem

The proposed algorithm for designing the *k*-clique problem takes the adjacency matrix as input and gives QASM gate list as output. This QASM gate list can further be used as an input to the quantum simulator for verification or to map into quantum technologies. The algorithm for the synthesis of the *k*-clique problem has a classical control that consists of classical components, which helps to build the quantum part of the algorithm. This algorithm can be illustrated as follows:

- **Classical input**:
  - Adj [][]: Adjacency matrix of the input graph G (V, E).
  - arr []: Array that holds the vertices of the graph.
  - $comb_{arr}$ []: Array that holds all combinations of the vertices for a clique size.
  - active []: Array that holds the binary equivalent of all combinations of vertices.
- **Classical part of the algorithm**:
  - Classical part of the algorithm prepares all possible input combinations of the vertices for a clique size. It creates $\binom{n}{k}$ combination of vertices if n is the total number of vertices and k is the clique size.
- **Quantum input**:
  - I []: This is a quantum register that holds the input qubits.
  - A []: This is a quantum register that holds the ancilla qubits.
  - T []: This is a quantum register that holds the target qubits.
  - O []: This is a quantum register that holds the output qubit.
- **Quantum part of the algorithm**
  - Quantum portions of the algorithm are executed on quantum computers, where unitary quantum gates (Hadamard, NOT, CNOT) are applied on the input qubits.

The execution of the algorithm is as follows:

- **Step 1**: Initialise Adjacency Matrix for the input graph. Calculate the total number of combinations of vertices using a classical algorithm.
- **Step 2**: Initialise quantum register I [] that holds the input qubits with $|0\rangle$.
- **Step 3**: Apply Hadamard gates on all the input qubits.
- **Step 4**: Apply Hadamard gates on the output qubit.
- **Step 5**: Execute the proposed unitary blocks, *i.e.* Qubit Activation, Clique Detector, and Qubit Deactivation for finding the marked states. This checks full connectivity in a set of vertices by applying the unitary transformation of quantum gates.
- **Step 6**: Apply Grover's operator for maximising the amplitude of the marked states.
- **Step 7**: Measure the output of quantum register I [] using a classical register.

4.2.1 | Automated and generalised algorithm for circuit synthesis of proposed oracle for *k*-clique problem

The proposed algorithm of oracular circuit synthesis for the *k*-clique problem is illustrated in this subsection. Algorithm 1 describes the algorithm for oracle circuit synthesis of the *k*-clique problem. The algorithm takes the adjacency matrix of the given graph as input and the *k*-sized cliques to be searched. The output of the algorithm is the oracle circuit in the form of a circuit netlist.

Algorithm 1 AutoGenerateOraclek-Clique

Input: Adjacency matrix $adj(n, n)$ of graph $G(V, E)$. Size of the clique to be searched that is k. I_m are input qubit lines where $1 \leq m \leq k * \lceil \log_2 n \rceil$. A_x and T are ancilla lines where $1 \leq x \leq \binom{k}{2}$. O is output line.
Output: Circuit Netlist(QASM)
Initialize I_m input lines with $|0\rangle$ followed by Hadamard gate, A_x ancilla lines with $|0\rangle$, T ancilla line with $|0\rangle$, and output line O with $|1\rangle$ followed by a Hadamard gate. Make $\binom{n}{k}$ combinations of vertices and store them in an array $comb_{arr}$.
Initialize $comb \leftarrow 0$, $m \leftarrow k * \lceil \log_2 n \rceil$ and $x \leftarrow 1$;
for $i \leftarrow 1$ *to* $\binom{n}{k}$ **do**
  insert Qubit Activation Block;
  insert k-clique Detector Block;
  insert Qubit Deactivation Block;
  $comb \leftarrow comb + k$ (To check next possible combination of vertices for searching k-clique), $x \leftarrow 1$;
Insert CNOT gate T line as control and O output line as target;
For mirror circuit:
for $i \leftarrow 1$ *to* $\binom{n}{k}$ **do**
  insert Qubit Activation Block;
  insert k-clique Detector Block;
  insert Qubit Deactivation Block;
  $comb \leftarrow comb + k, x \leftarrow 1$;

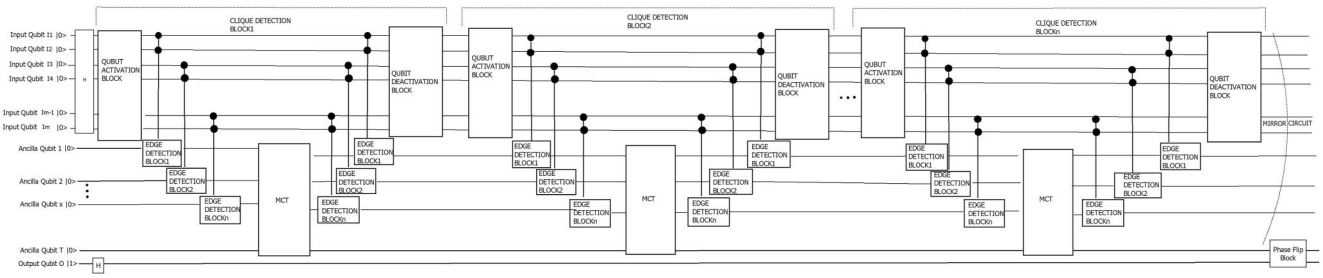

FIGURE 8 Circuit representation of oracle for the *k*-clique problem

Algorithm 2 Qubit Activation Block & Qubit Deactivation Block

Binary representation of $comb_{arr}[comb]$, $comb_{arr}[comb+1] \ldots comb_{arr}[comb+k-1]$ in $\lceil \log_2 n \rceil$ binary digits stores in an array *active*, the current possible combination of vertices for searching k-clique;
Qubit Activation Block:
for $r \leftarrow 0$ *to* $m-1$ **do**
  if $active(r) \leftarrow 0$ **then**
    insert NOT gate to qubit line I_{r+1} to make input qubit line active. $r \leftarrow r+1$;
  else
    $r \leftarrow r+1$
Qubit Deactivation Block:
for $r \leftarrow 0$ *to* $m-1$ **do**
  if $active(r) \leftarrow 0$ **then**
    insert NOT gate to qubit line I_{r+1} to make input qubit line active. $r \leftarrow r+1$;
  else
    $r \leftarrow r+1$

Algorithm 3 *k*-clique Detector Block

for $i \leftarrow comb$ *to* $comb+k-2$ **do**
  for $j \leftarrow i+1$ *to* $comb+k-1$ **do**
    if $adj(comb_{arr}(i), comb_{arr}(j)) \leftarrow 1$ **then**
      if $(j+1)\%k \leftarrow 0$ **then**
        insert a multi-controlled Toffoli gate with $I_{((i\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((i+1)\%k)*\lceil \log_2 n\rceil}$ input qubit lines and $I_{((j\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((j\%k)+1)*\lceil \log_2 n\rceil}$ input qubit lines as control and ancilla A_x as target.
        $x \leftarrow x+1$;
      else
        insert a multi-controlled Toffoli gate with $I_{((i\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((i+1)\%k)*\lceil \log_2 n\rceil}$ input qubit lines and $I_{((j\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((j+1)\%k)*\lceil \log_2 n\rceil}$ input qubit lines as control and ancilla A_x as target.
        $x \leftarrow x+1$;
    else
      $x \leftarrow x+1$
Insert a k-controlled Toffoli gate with each of the k A_x ancilla lines as control and ancilla T as target;
$x \leftarrow k$ $comb \leftarrow 0$;
for $j \leftarrow comb+k-1$ *to* $comb+k-2$ **do**
  for $i \leftarrow j-1$ *to* $comb$ **do**
    if $adj(comb_{arr}(i), comb_{arr}(j)) \leftarrow 1$ **then**
      if $(j+1)\%k \leftarrow 0$ **then**
        insert a multi-controlled Toffoli gate with $I_{((i\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((i+1)\%k)*\lceil \log_2 n\rceil}$ input qubit lines and $I_{((j\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((j\%k)+1)*\lceil \log_2 n\rceil}$ input qubit lines as control and ancilla A_x as target.
        $x \leftarrow x-1$;
      else
        insert a multi-controlled Toffoli gate with $I_{((i\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((i+1)\%k)*\lceil \log_2 n\rceil}$ input qubit lines and $I_{((j\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((j+1)\%k)*\lceil \log_2 n\rceil}$ input qubit lines as control and ancilla A_k as target.
        $x \leftarrow x-1$;
    else
      $x \leftarrow x-1$

The steps of the algorithms are described as follows: flushleft

Step 1: From the adjacency matrix, the number of vertices in the graph are obtained, which are further used to calculate the total number of qubit lines required for the oracle circuit. If there are n vertices in the input graph, at first vertices are indexed as $\{0, 1, 2, \ldots n-1\}$. The algorithm stores $\binom{n}{k}$ combination of vertices without a duplicate entry in an array called $comb_{arr}$. The number of qubits required to represent each vertex is $\lceil \log_2 n \rceil$. Hence, a total of $k * \lceil \log_2 n \rceil$ qubit lines are required for input. There are two types of ancilla qubit lines, A_x and T, which are initialised by $|0\rangle$. A_x ancillas are used for checking whether two vertices are connected by an edgeand T ancilla is used for checking if $\binom{k}{2}$ edges form aclique of size k. There is one output line O, initialised to $|1\rangle$, which indicates if a clique of size k exists in the input graph.

Step 2: The algorithm first applies Hadamard gates on all input qubit lines. For every possible combination from array $comb_{arr}$, binary representation of k vertices in $\lceil \log_2 n \rceil$ binary digits stores in array *active*. This is further required to activate all the input qubit lines having $|0\rangle$ quantum state using appropriate NOT gates to make them suitable as control lines for MCT operation with the help of the Qubit Activation method. This Qubit Activation method is described in Algorithm 2.

Step 3: For every possible combination from array $comb_{arr}$, the algorithm checks every possible vertex pairs ($comb_{arr}(i)$, $comb_{arr}(j)$) that are connected by an edge using an edge detector block and insert a k-clique detector block on appropriate qubit lines for respective vertices. The circuit synthesis of the k-clique Detector Block is described in Algorithm 3. The algorithm checks for every possible combination of k vertices without duplicate entry, if vertex pairs ($comb_{arr}(i)$, $comb_{arr}(j)$) are connected by an edge, then insert a multi-controlled Toffoli gate with $I_{((i\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{(i\%k)*\lceil \log_2 n\rceil}$ input qubit lines and $I_{((j\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{((j\%k)+1)*\lceil \log_2 n\rceil}$ input qubit lines as control and one of the ancilla A_x as target if $(j+1)\%k$ is zero (where i, j are the index values of array $comb_{arr}$) or else insert a multi-controlled Toffoli gate with $I_{((i\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{(i\%k)*\lceil \log_2 n\rceil}$ input qubit lines and $I_{((j\%k)*\lceil \log_2 n\rceil)+1}$ to $I_{(j+1)*\lceil \log_2 n\rceil}$ input qubit lines as control and one of the ancilla A_x as target. If all vertex pairs of a vertex combination are adjacent, then each of the A_x ancilla qubits are activated, which implies there exists a clique of size k between them. This information is stored in the T ancilla by applying a $\binom{k}{2}$-controlled Toffoli gate with A_x ancillas as control and T ancilla as target. These steps are needed to be repeated followed by Qubit Deactivation Block to generate the mirror of it in order to keep the overall circuit reversible.

Step 4: The algorithm then applies a CNOT gate to the output line with T ancilla as control. As a result, the output line shall indicate the presence of a clique of size k in the graph. On the contrary, if there is no clique of size k in the input graph, T ancilla will be in their initial state $|0\rangle$.

The remaining steps of the algorithm generate the mirror for the original circuit in order to keep the overall circuit netlist reversible. To understand Algorithm 1 more precisely, an example of 4 vertices graph has been considered. The oracle circuit generated for the example graph by the proposed algorithm is illustrated in the next subsection.

4.2.2 | Generation of the proposed oracle circuit for *k*-clique problem for two exemplified graph

The circuit for finding a triangle in the graph of Figure 9 has been shown in Figure 10. For a 4-vertex graph, an adjacency matrix is represented by *adj* (4, 4) and vertices are indexed as 0, 1, 2, 3. Each vertex is represented by $\lceil \log_2 4 \rceil = 2$ qubits. Hence, total six qubit lines are required for inputs. As per Algorithm 1, input qubit lines are represented as I_m , where $1 \leq m \leq 6$. These input qubit lines are initialised with $|0\rangle$ followed by the Hadamard gate. In order to store edge information, three ancilla lines A_x, where $1 \leq x \leq 3$, one ancilla line T initialised to $|0\rangle$ and one output line O initialised to $|1\rangle$

followed by the Hadamard gate are required to store the output of the circuit. Hence, a total of 11 qubit lines are required for the simulation. The algorithm now stores $\binom{4}{3=4}$ possible combination of vertices as ((0, 1, 2), (0, 1, 3), (0, 2, 3), (1, 2, 3)) in an array called $comb_{arr}$. The algorithm now initialises variable *comb* as 0, variable m as 6 and x as 1.

- At first the algorithm considers the first combination of three vertices. Hence, the binary representation of three vertices (0, 1, 2) in $\lceil \log_2 4 \rceil = 2$ binary digits as (000110) stores in an array *active* of length 6.
- For each zero in the *active* array, insert a NOT gate to qubit line I_{r+1} where r is the index value of an *active* array to make these input lines suitable as control for MCT operation or edge detector block.
- Thereafter, for every possible vertex pair (0, 1), (0, 2), and (1, 2), the algorithm checks that they are connected by an edge or not by accessing the information of the adjacency matrix.
- At first for vertex pair ($comb_{arr}$ ($i = 0$), $comb_{arr}$ ($j = 1$)) = (0, 1), adj (0, 1) = 1, insert an MCT gate with I_1 to I_2 input qubit lines and I_3 to I_4 input qubit lines as control and A_2 as target.
- In case of vertex pair ($comb_{arr}$ ($i = 0$), $comb_{arr}$ ($j = 2$)) = (0, 2), adj (0, 2) = 1, then $(j + 1)\%3$ is zero, then insert an MCT gate with I_1 to I_2 input qubit lines and I_5 to I_6 input qubit lines as control and A_1 as target.
- Similarly, for vertex pair ($comb_{arr}$ ($i = 1$), $comb_{arr}$ ($j = 2$)) = (1, 2), the entry for the adjacency matrix adj (1, 2) is 0 as there is no edge between vertex 1 and 2. So, the algorithm does not insert any MCT gate on the A_3 ancilla line.
- Thereafter, insert a three-controlled Toffoli gate with each of the three A_1, A_2, and A_3 ancilla lines as control and T ancilla as a target.
- Again to make three A_1, A_2, and A_3 ancilla lines reusable, In case of vertex pair ($comb_{arr}$ ($i = 0$), $comb_{arr}$ ($j = 1$)) = (0, 1), adj (0, 1) = 1, insert an MCT gate with I_1 to I_2 input qubit lines and I_3 to I_4 input qubit lines as control and A_2 as target.
- For vertex pair (($comb_{arr}$ ($i = 0$), $comb_{arr}$ ($j = 2$)) = (0, 2), adj (0, 2) = 1, then $(j + 1)\%3$ is zero, then insert an MCT gate with I_1 to I_2 input qubit lines and I_5 to I_6 input qubit lines as control and A_1 as target.
- These steps are repeated for the other three combinations of three vertices. The algorithm then applies a CNOT gate to the output line O with T ancilla as control. As a result, the output line shall indicate the presence of a clique in the graph. The remaining steps of the algorithm generate the mirror for the original circuit in order to keep the overall circuit netlist reversible.

The circuit for finding a triangle in the graph of Figure 11, which has been used in the state-of-the-art work [14], is shown in Figure 12. As this is also a 4-vertex graph, the size of the adjacency matrix remains the same as the graph of Figure 9 that is adj (4, 4). The total number of input qubits and ancilla qubits are also same. The difference is, the graph of Figure 9 has two cliques of

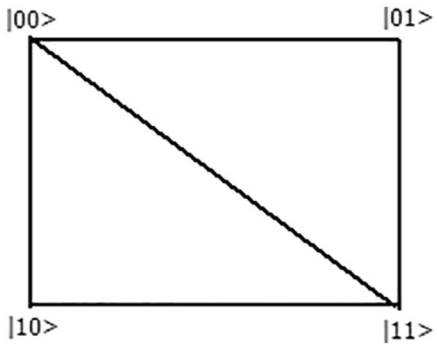

FIGURE 9 Graph with clique size 3

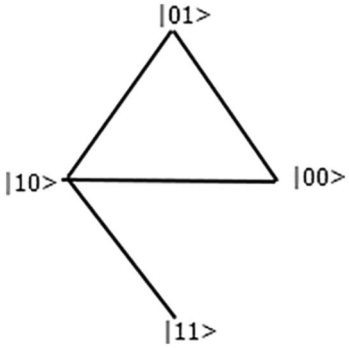

FIGURE 11 Graph with clique size 3

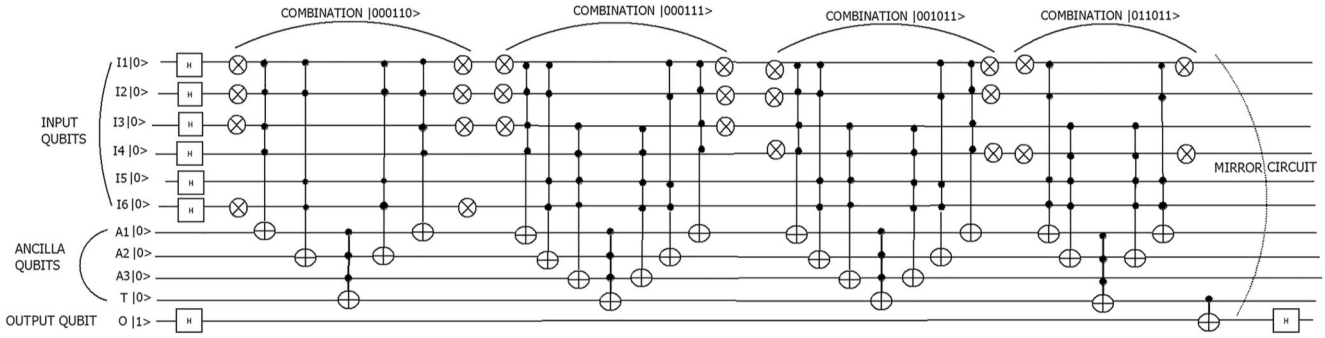

FIGURE 10 Circuit for checking the presence clique of size 3

size 3 and the graph of the Figure 11 has only one clique of size 3. Hence, the circuit designed using our proposed method is shown in the Figure 12, which lists all cliques present in the graph.

4.2.3 | Mathematical formulation of proposed oracle for k-clique problem for an exemplified graph

The generalisation of the oracle for the k-clique problem is already well established in the previous subsections. This section shows how an arbitrary graph formulates the proposed oracle mathematically. We have considered the graph as shown in Figure 13 and the corresponding generated circuit given in Figure 14.

Initialisation: The input qubits are initialised with $|0\rangle$. Ancilla qubits are initialised with $|0\rangle$. Output qubit is initialised with $|1\rangle$. Entire initialisation can be mathematically written as:

$$|\psi\rangle \otimes |\theta\rangle \otimes |\phi\rangle = |0\rangle^{\otimes 6} \otimes |0\rangle^{\otimes 3} \otimes |1\rangle$$

Hadamard Transformation: After the initialisation, the Hadamard transform $H^{\otimes 6}$ on input qubits and H on output qubit are performed; therefore, all possible states are superposed as $|\psi_0\rangle \otimes |\theta_0\rangle \otimes |\phi_0\rangle$, where

$$|\psi_0\rangle = \frac{1}{\sqrt{2^6}} \sum_{i=0}^{2^6-1} |i\rangle$$

$$|\theta_0\rangle = |000\rangle$$

$$|\phi_0\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$|\rho_0\rangle = |\psi_0\rangle \otimes |\theta_0\rangle \otimes |\phi_0\rangle$$

$$= \frac{1}{\sqrt{2^6}} \sum_{i=0}^{2^6-1} |i\rangle \otimes |0\rangle^{\otimes 3} \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$\Rightarrow \frac{1}{\sqrt{2^6}}(|000000\rangle + \cdots + |111111\rangle) \otimes |0\rangle^{\otimes 3} \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$\Rightarrow \frac{1}{\sqrt{2^7}}(|0000000000\rangle + |0000010000\rangle + \cdots + |1111110000\rangle)$$

$$+ \frac{1}{\sqrt{2^7}}(-|0000000001\rangle - |0000010001\rangle - \cdots - |1111110001\rangle)$$

Oracle:

- **combination of vertices $|000110\rangle$**:-
  1. **Search space reduction using qubit activation**:- After applying NOT gates to the first (I_1), second (I_2), third (I_3) and sixth (I_6) qubit of all superposed states, the quantum state evolves as $|\rho_1\rangle = \frac{1}{\sqrt{2^7}}(|1110010000\rangle + |1110010000\rangle + |1110110000\rangle + |1110100000\rangle + |1111010000\rangle + |1111000000\rangle + |1111000000\rangle + |\mathbf{1111110000}\rangle + \cdots + |0001100000\rangle) + \frac{1}{\sqrt{2^7}}(-|1110010001\rangle - |1110000001\rangle - \ldots |\mathbf{1111110001}\rangle \cdots - |0001100001\rangle)$
  2. **Edge detection**:- Three MCT gates are applied for the combination of vertices ($|00\rangle$, $|10\rangle$), ($|01\rangle$, $|10\rangle$) and ($|00\rangle$, $|01\rangle$) for three edges. The quantum state evolves as $|\rho_2\rangle = \frac{1}{\sqrt{2^7}}(|1110010000\rangle + |1110000000\rangle + |1110110000\rangle + |1110100000\rangle + |1111010000\rangle + |1111000000\rangle + |1111000000\rangle + |\mathbf{1111111110}\rangle + \cdots + |1111110000\rangle) + \frac{1}{\sqrt{2^7}}(-|1110010001\rangle - |1110000001\rangle - \ldots |\mathbf{1111111111}\rangle \cdots - |0001100001\rangle)$
  3. **Clique detection**:- An MCT gate flips the fourth ancilla qubit T, if the three ancilla qubits A_1, A_2, A_3 are 1. The quantum state evolves as $|\rho_3\rangle = \frac{1}{\sqrt{2^7}}(|1110010000\rangle +$

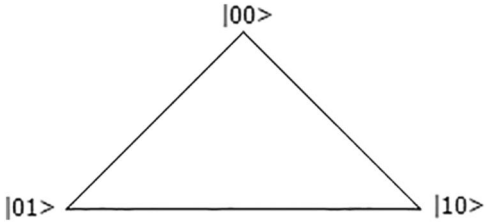

FIGURE 13 Graph with clique size 3

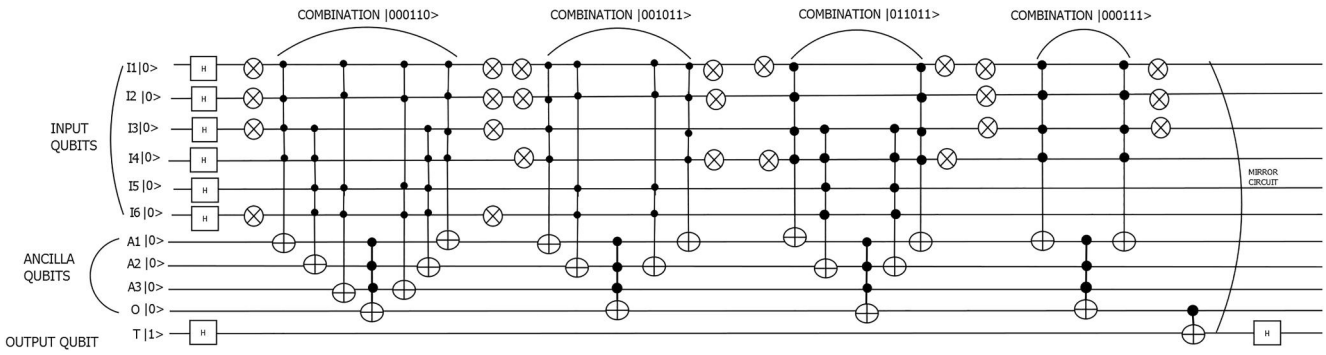

FIGURE 12 Circuit for checking the presence clique of size 3

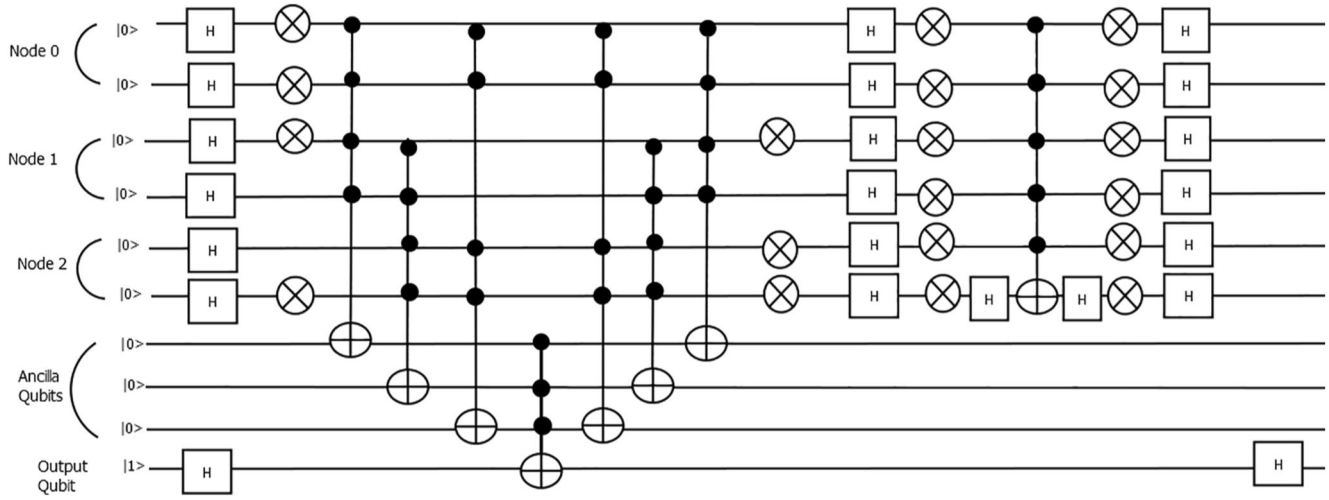

FIGURE 14 Circuit for checking the presence clique of size 3

$|1110000000\rangle + |1110110000\rangle + |1110100000\rangle + |1111010000\rangle + |1111000000\rangle + |1111000000\rangle + |\mathbf{1111111111}\rangle + \cdots + |1111110000\rangle) + \frac{1}{\sqrt{2^7}}(-|1110010001\rangle - |1110000001\rangle - \ldots|\mathbf{11111111110}\rangle \cdots - |00011000001\rangle)$

4. **Mirror circuit**: After applying mirror circuit, the quantum state evolves as $|\rho_4\rangle = \frac{1}{\sqrt{2^7}}(|0000000000\rangle + |0000010000\rangle + |0000100000\rangle + |0000110000\rangle + |0001000000\rangle + |0001010000\rangle + |0001010000\rangle + |\mathbf{0001100001}\rangle + \cdots + |1111110000\rangle) + \frac{1}{\sqrt{2^7}}(-|0000000001\rangle - |0000010001\rangle - \ldots|\mathbf{0001100000}\rangle \cdots - |1111110001\rangle)$

- **Hadamard gate on the output qubit:** After applying Hadamard gate on the output qubit (O), the quantum state evolves as $|\rho_5\rangle = \frac{1}{\sqrt{2^7}}(|0000000001\rangle + |0000010001\rangle + \cdots + |0001100001\rangle \cdots + |0110110001\rangle \cdots + |1111110001\rangle) + \frac{1}{\sqrt{2^7}}(-|\mathbf{0001100001}\rangle)$

We observe that the marked state $|000110\rangle$ is with the negative amplitude only. Now, Grover's diffusion operator performs the inversion about the average of all amplitudes of the quantum state for a certain number of iterations to get the amplitude of the marked state large enough. So, the amplitude of the marked states becomes higher than other states.

4.3 | Automated and generalised algorithm for solving maximum clique problem using the proposed approach of *k*-clique problem in classical-quantum hybrid architecture

The automated and generalised algorithm of quantum circuit synthesis for the maximum clique problem is discussed in this subsection. The maximum clique problem finds clique in a graph with maximum cardinality. The input of the algorithm is the Adjacency Matrix of the graph $G(V, E)$ and the output is the QASM gate list. The algorithm uses the concept of the proposed k-clique algorithm. The algorithm starts with finding clique of size $k = n$ (n is the total number of vertices of the graph) using the algorithm of k-clique. If a clique is found, it is the maximum clique of the graph. Otherwise, a clique is searched for size $n - 1$. The process continues up to $n = 2$. The algorithm is performed in a classical-quantum hybrid architecture. As described in subsection 2.4, hybrid architecture requires a feedback loop from the quantum part of the algorithm to the classical part of the algorithm, which is portrayed in Figure 15 for the MCP problem.

The steps of the circuit synthesis for MCP are shown below:

- **Step 1**: Algorithm for finding k-clique is used for searching clique of size n, where the number vertices of the given graph are k.
- **Step 2**: If a clique is found, then exit from the process.
- **Step 3**: If clique is not found, then the clique size n is reduced to $n - 1$ and repeat the whole process from step 1.
- **Step 4**: At each iteration, the value of n is reduced by 1 (n, $n - 1$, $n - 2$, $n - 3$, ..., 2) until a clique is found.

The algorithm for solving MCP using the proposed approach of the k-clique problem is illustrated in Algorithm 4.

Algorithm 4 AutoGenerateOracleMaximumClique

Input: Adjacency Matrix Adj[][]
Output: A file "OutputMaximumClique.QASM"
keep all vertices of the input graph(n) in an array $arr[]$
for $i \leftarrow n$ *downto* 2 **do**
  Use k**-clique alogorithm for finding clique of size** i
  if *clique of size* i *is found* **then**
    Maximum clique is found of size i
    Exit from the loop

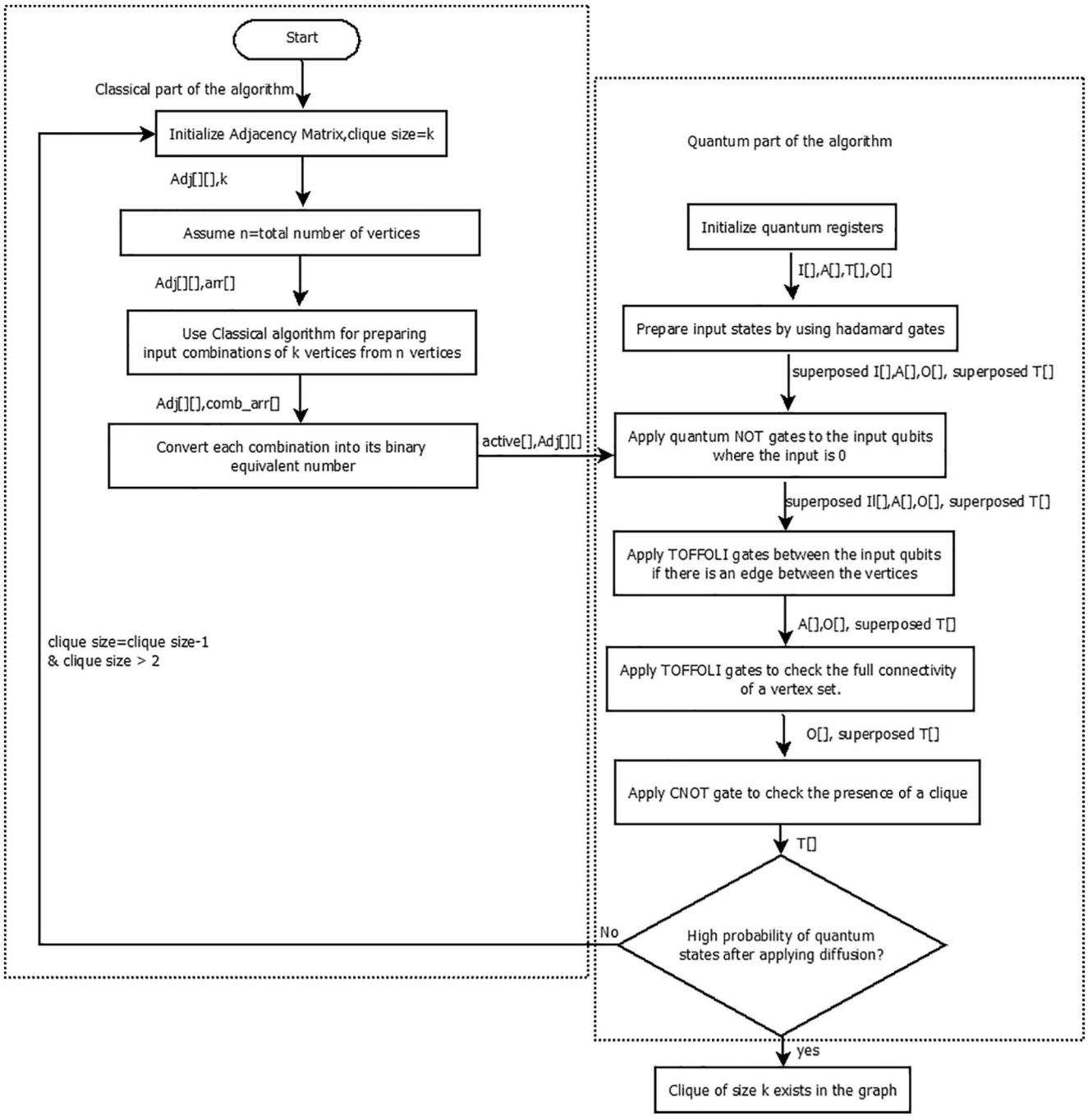

FIGURE 15 The flow of the complete algorithm for the maximum clique problem in a classical-quantum hybrid architecture

5 | COST ANALYSIS OF THE CIRCUIT

The cost of a quantum circuit depends on the number of the input lines, ancilla lines, and the number of gates used. If the circuit is designed for a size k, then the total combination is $\binom{n}{k}$, when the number of vertices of a given graph is n. Therefore, a total number of $O\left(k * \binom{n}{k} + k\right)$ logical gates are required to design the oracle circuit for a variant of the k-clique problem, which lists all the cliques of size k. As this is a first of its kind approach to solving such a k-clique problem, there is no sufficient work to compare the oracular gate cost. In the present scenario, due to the constraint of less number of qubits, a quantum circuit with a large number of gates with multiple qubits cannot be implemented in real hardware due to its depth. As discussed in the previous section that each vertex is represented by $\lceil \log_2 n \rceil$ qubits, so a total number of input qubits $k*\lceil \log_2 n \rceil$ are required along with $\binom{k}{2} + 2$ ancilla qubits as shown in Table 3. Therefore, for detecting an edge between two vertices, we need a multi-controlled Toffoli gate with $2*\lceil \log_2 n \rceil$ control qubits and for detection of the clique

of size k, we need multi-controlled Toffoli gate with $\binom{k}{2}$ control qubits as shown in Table 3. Finally, we can conclude that $k * \lceil \log_2 n \rceil$-qubit MCT gate is required to perform the diffusion part of the proposed approach of Grover's algorithm.

Analysis of the qubit cost and the depth of the complete circuit for the k-clique problem is as follows:

Lemma 1 *Improved qubit cost for the proposed approach of solving the k-clique problem, when $n \gg k$, where n is a large number of nodes of given graph ($G(V = n, E = e)$) and k is, respectively, very small.*

The total number of qubits are required to solve the k-clique problem using proposed approach is $k * \lceil \log_2 n \rceil + \binom{k}{2} + 2$. As discussed in Subsection 2.6.2 in the backgr ound, the state-of-the-art approach requires a total $n + \lceil \log_2 k \rceil + 3$ qubits for solving the k-clique problem, which gives a clique of size k as output. The state-of-the-art problem is the subset of the proposed problem of k-clique as our proposed algorithm gives all the cliques of size k as output. So from this, we can infer that if we solve such a variant of the k-clique problem, then it can also solve the state-of-the-art problem of k-clique. To prove the stated *Lemma*, we have considered three instances of n and k values. Let us start with if $n = 1024$ and, respectively, k is very small, *i.e.* 3, then the qubit cost $3 * \lceil \log_2 1024 \rceil + \binom{3}{2} + 2 = 35$ of the proposed approach is very efficient as compared to the qubit cost $1024 + \lceil \log_2 3 \rceil + 3 = 1029$ of the state-of-the-art approach. Now, if we slightly increase the value of k as 32, but still remains very small as compared to $n = 1024$, then the qubit cost $32 * \lceil \log_2 1024 \rceil + \binom{32}{2} + 2 = 818$ of the proposed approach is still efficient as compared to the qubit cost $1024 + \lceil \log_2 32 \rceil + 3 = 1032$ of state-of-the-art approach. But, if we increase the value of k a little further towards n, our approach becomes costlier with respect to the qubit cost to find a clique size of k in a given graph. For instance, suppose $n = 1024$ and $k = 64$, the qubit cost $64 * \lceil \log_2 1024 \rceil + \binom{64}{2} + 2 = 2658$ of the proposed approach is costlier as compared to the qubit cost $1024 + \lceil \log_2 64 \rceil + 3 = 1033$ of the state-of-the-art approach. Thus, we can conclude that our proposed approach outperforms the state-of-the-art approach of solving the k-clique problem with respect to qubit cost, when k is, respectively, very small with respect to large n. The comparative result of the qubit cost has been shown in Figure 16. The total number of vertices

TABLE 3 Quantum cost analysis of an oracular circuit for the k-clique problem

No of vertices	No of input qubit	No of ancilla qubits	Maximum number of controls in the MCT gate
n	$k * \lceil \log_2 n \rceil$	$\binom{k}{2} + 2$	Edge detection $= 2 * \lceil \log_2 n \rceil$, Clique detection $= \binom{k}{2}$

Abbreviation: MCT, multi-controlled Toffoli gate.

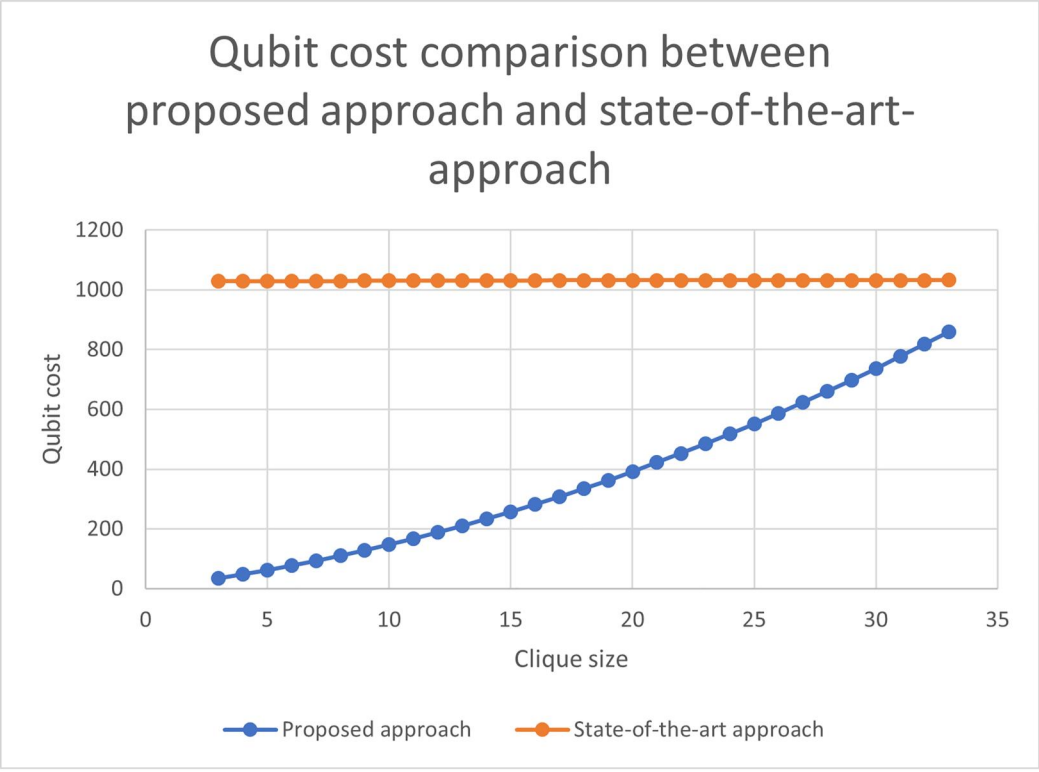

FIGURE 16 Comparison of qubit cost between the proposed approach and the state-of-the-art approach [14] with respect to clique size k varying from 3 to 33 for an exemplified graph of 1024 vertices

has been taken as 1024. The clique size k is varied from 3 to 33. It can be observed from Figure 16 that the qubit cost of the proposed approach is better than the state-of-the-art approach [14] when $n \gg k$. flushleft

Proposition 1 *Improved circuit depth for the proposed approach of solving the k-clique problem, when $n \gg k$, where n is a large number of nodes of given graph ($G(V = n, E = e)$) and k is, respectively, very small.*

From *Lemma* 1, it can be inferred that the input qubit cost $k * \lceil \log_2 n \rceil$ of the proposed approach is also efficient as compared to the input qubit cost n of the state-of-the-art approach, when $n \gg k$ and when $k \equiv n$, it becomes inefficient as $n * \lceil \log_2 n \rceil > n$. It is already defined that the diffusion operator is imposed on all the input qubits. Therefore, a $k * \lceil \log_2 n \rceil$-qubit MCT is required to perform this diffusion for the proposed approach as compared to n-qubit MCT for the state of the art. As $k * \lceil \log_2 n \rceil < n$, when $n \gg k$, we can infer from Figure 2 in Subsection 2.3 that the depth of the circuit is also efficient for the proposed approach as compared to the state-of-the-art method. As Figure 2 suggests that with the increasing number of controls in the MCT gate also increases the depth of the circuit drastically, our goal is to restrict the number of controls in the MCT gate as compared to the work in [14], which is successfully achieved for $n \gg k$. As it can be observed from Table 4, if the clique size k is much less than the number of vertices n, then the number of controls of the MCT gate is much lower in the proposed approach than the state-of-the-art approach [14], which in turn reduces the circuit depth of the proposed approach.

Now, if we design the circuit for the same graph as proposed in the state-of-the-art work [14] and decompose the circuit into only one-qubit and two-qubit gates, we get the following result as shown in Table 5.

As it can be observed from the Table 5, the circuit size as well as the number of one-qubit and two-qubit gates is less in the proposed approach. The proposed method lists all cliques of size 3 as compared to that in [14]; hence, the depth of the oracular circuit is higher than the state-of-the art approach.

6 | IMPLEMENTATION RESULTS

Quantum simulators imitate the behaviour of a quantum algorithm on a classical computer in order to get a sense of how the circuit may be implemented on a real quantum computer. We can also use the IBM cloud to test quantum algorithms on quantum computers. IBM's Quantum Experience platform includes free quantum computers with up to 14 qubits and a 32-qubit quantum simulator. Despite the fact that real quantum computers may be used to conduct experiments, simulators are still used to test quantum. We must consider the fact that on a genuine quantum computer, certain data that is valuable for testing purposes cannot be accessed. Another issue is that we would like to test a quantum algorithm that requires a huge number of qubits, which is still beyond the capabilities of real-world hardware. The circuit depth or length of the circuit's critical route determines how a circuit is implemented in a real quantum device. Regrettably, the circuit depth is strongly reliant on the hardware configuration of qubits and their connections. We were unable to create the circuit for larger graphs in the real quantum device due to this constraint.

6.1 | Ideal simulation on *ibmq_qasm_simulator*

The oracle circuit for the k-clique problem for the graph shown in Figure 11 has been implemented in *ibmq_qasm_simulator*. Here, we have 6 qubit lines for inputs; there can be a total of $2^6 = 64$ different qubit combinations as $|000000\rangle$, $|000001\rangle$, …,$|111111\rangle$ after applying the Hadamard gate to all input qubits. Hence, the database on which Grover's algorithm is applied contains 64 different elements. Figure 12 shows the complete gate-level synthesis of Grover's circuit for the 3-clique problem.

The simulation steps on *IBMQ_Qasm_Simulator* [21] required for the synthesis are as follows:

1. The oracle checks whether two vertices are adjacent or not. It determines whether a clique is formed by a three-vertex combination. Three-vertex combinations are $|000110\rangle$. It

TABLE 4 Circuit depth comparison of an oracular circuit for the proposed approach and state-of-the-art approach [14]

No of vertices	Clique size	Number of controls of the MCT gate in the proposed approach	Number of controls of the MCT gate in the state-of-the-art approach
n	k	$k * \lceil \log_2 n \rceil$	n
1024	16	160	1024
2048	32	640	2048

Abbreviation: MCT, multi-controlled Toffoli gate.

TABLE 5 Circuit size, depth, number of qubits and number of gates of oracular circuit for the proposed approach and state-of-the-art approach [14]

Approach	Circuit size	Circuit depth	One-qubit gates	Two-qubit gates
Proposed	1619	1250	846	773
State-of-the-art [14]	1816	1173	1051	765

then inverts the amplitude of the solution state for which the vertex combination forms a clique. The solution state for this case is $|000110\rangle$.

2. The output of the oracle is acted upon by the Grover's diffusion operator as shown in Figure 14. The diffusion operator amplifies the amplitude of the marked solution state.
3. Steps 1 and 2 constitute the Grover's operator for Grover's search algorithm. Hence, these two steps are repeated $\frac{\pi}{4}\sqrt{64}$ times. (For an N item database with M number of known solutions [40, 41], Grover's iteration must be repeated $\frac{\pi}{4}\sqrt{\frac{N}{M}}$ times in order to obtain a solution. If our number of solutions are not previously known [42–47],

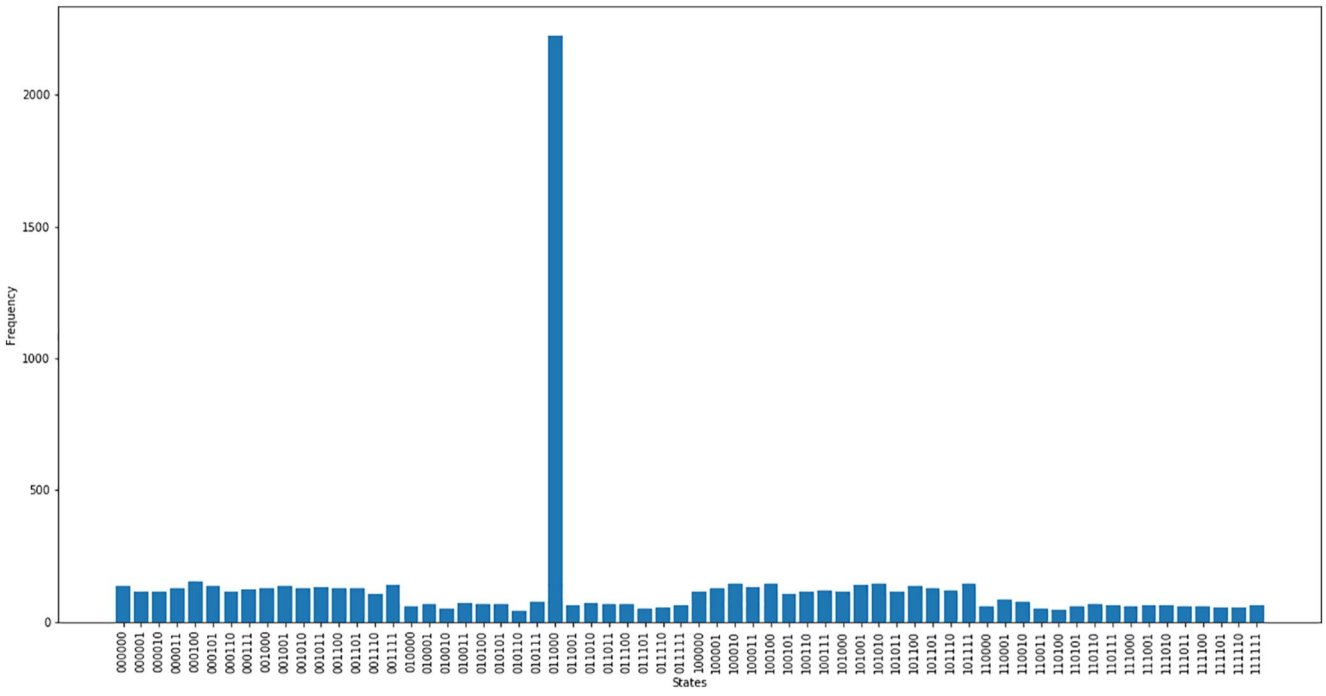

FIGURE 17 Ideal simulation of the algorithm

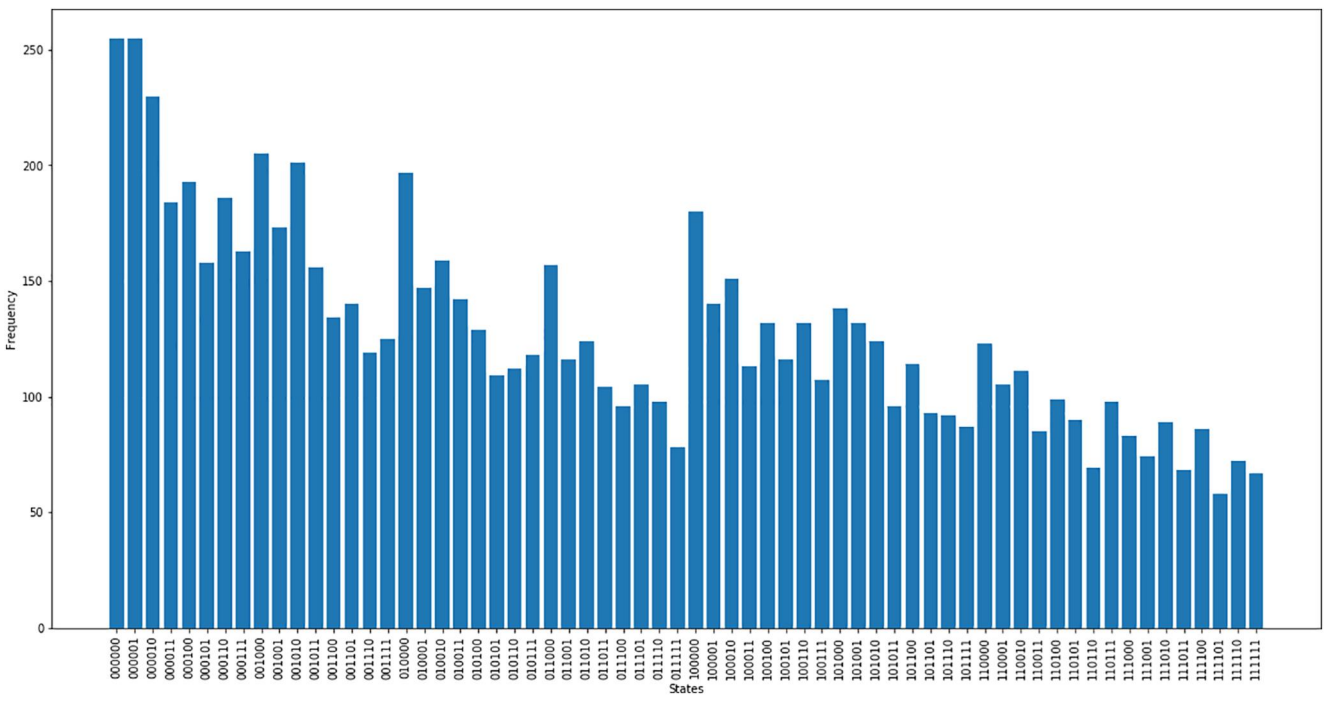

FIGURE 18 Implementation result in *IBMQ_16_melbourne* after one iteration

then we need to consider a tricky method for the iterations without any asymptotic penalty. As per [48], we need to start with a single application of the Grover Iterate. If the marked state is not found, then repeat Grover's algorithm with a number of iterations, which is 6/5 times the previous number. We need to stop the algorithm if the number of iteration is reached to $\frac{\pi}{4}\sqrt{N}$.)

The resultant output after applying Grover's operator $\frac{\pi}{4}\sqrt{1} \sim O(1)$ time is shown in Figure 17, where the amplitude of the solution state has been amplified. The location of the solution state is $|000110\rangle$, which is the vertex combination in an example graph of Figure 13 that forms a clique with high probability. The circuit for clique size three has been simulated in *IBMQ_Qasm_Simulator*. The marked state $|011000\rangle$ is in high amplitude as shown in the output Figure 17, which depicts that $|00\rangle$, $|01\rangle$ and $|10\rangle$ form a triangle.

6.2 | Implementation on *IBMQ_16_melbourne*

The simulation results using IBM's simulator are unrealistic because they display perfect output with no errors or noise. On *IBMQ_16_melbourne*, the circuit of Figure 12 is implemented once more. The histogram in Figure 18 is obtained by repeating the technique in *IBMQ_16_melbourne*. As we can see, the required state $|000110\rangle$ is not adequately identified, and the noise resulted in data loss.

6.3 | Simulation using noise model of *IBMQ_16_melbourne*

As discussed in the background, quantum gates and qubits are noisy and error-prone. So, for a realistic result, the quantum circuit can be implemented with the noise model obtained from a real quantum device. Qiskit Aer offers 10 standard error models, like Pauli Error, phase Damping Error, Mixed Unitary Error, Depolarisation Error, Reset Error, Thermal Error where users can customise the error models. In addition, the user can choose whether to apply the error to all qubits or a specific set of qubits. We have executed the quantum circuit of Figure 14 on the remote *ibmq_qasm_simulator*, applying a noise model obtained from the available data about *ibmq_16_melbourne*. Figure 19 shows the comparison between the two histograms, with ideal computation in blue and noisy computation in orange.

6.4 | Analysis of the result

The findings of the implementation in the real quantum device show that the result has been influenced by noise, as shown in Figure 18. In the following subsections, the success rate and execution time of the circuit in Figure 12 are analysed.

6.4.1 | Success rate on *IBMQ_16_Melbourne*

A total number of gates of the circuit of Figure 12 are 132. From the data available about *IBMQ_16_Melbourne*, we can mention the following two parameters:

- Mean gate error: 2.14×10^{-3}
- Mean measure error: 2.68×10^{-2}

The probability that at least one gate fails is given by the following.

P (at least one gate fails).
$= 1 - P$ (all gate succeed).
$= 1 - P$ (1 gate succeed)^{132}
$= 1 - (1 - 2.14 \times 10^{-3})^{132} = 1 - 0.75368 = 0.24632$

This result infers that without accounting for decoherence errors, the failure rate of our circuit is 25%. This is the first important piece of information that gives us an idea of the problem of noise in quantum computation.

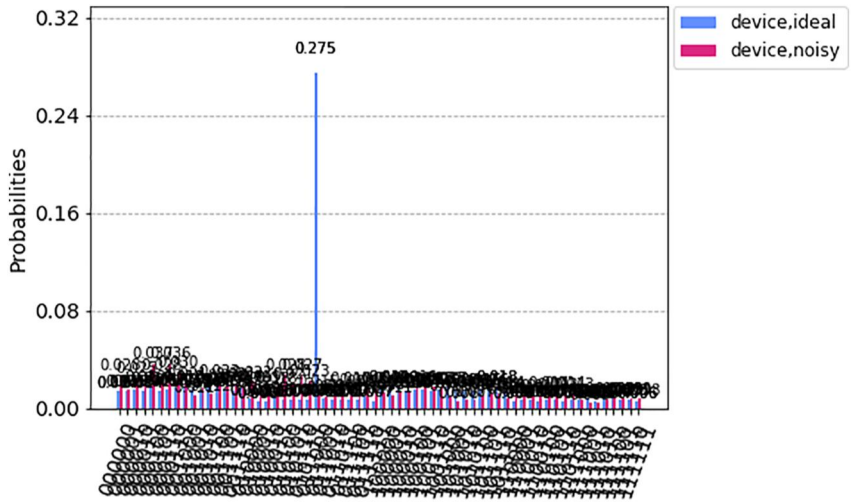

FIGURE 19 Ideal result and noisy result

6.4.2 | Execution time of the circuit in Figure 12 on *IBMQ_16_Melbourne*

Now that we have talked about decoherence faults, we can try to figure out how long it will take the *ibmq_16_melbourne* to complete our circuit's execution. *U*1, *U*2, *U*3 and *CNOT* are the basis gates utilised by the *ibmq_16_melbourne* to create quantum circuits and transform more complicated gates into simpler ones. We can gather the essential data to calculate the following from public data on the *ibmq_16_melbourne* available at [27, 49], we can obtain the required data to calculate the following:

- Time needed for U1: 0 ns;
- Time needed for U2: 100 ns + 20 ns = 120 ns;
- Time needed for U3: 100 ns + 20 ns + 100 ns + 20 ns = 240 ns;
- Roughly average time needed for CNOT: 100 ns + 20 ns + 360 ns + 20 ns + 100 ns + 20 ns + 360 ns + 20 ns = 1000 ns.

Now, if the circuit of the Figure 12 is decomposed into *u*1, *u*2, *u*3 and *CNOT* gates, then we have 699 *u*1 gates, 773 *CNOT* gates, 11 *U*3 gates and 136 *U*2 gates.

- Total time required for the CNOT gate is 1000 × 773 = 773000 ns.
- Total time required for *U*2 gate is 136 × 120 = 16320 ns.
- Total time required for *U*3 gate is 11 × 240 = 2640 ns

Hence, the total time required to execute the circuit is 773000 + 16320 + 2640 = 791960 *ns* = 791.96 μs. Therefore, our circuit will take approximately 791.96 μs. As coherence times for Melbourne are $T1 = 71.5$ μs and $T2 = 21.4$ μs, we can infer that we have a relatively low probability to execute the whole circuit without at least one error due to decoherence. It is worth noticing that for a larger circuit, the total number of gates and circuit depth will be higher.

Now, the circuit in Figure 12 is further implemented on four IBMQ devices: *IBMQ_16_Melbourne*, *IBMQ_Cambridge*, *IBMQ_Rochester*, *IBMQ_Singapore*. If we compare the execution time of the circuit on the four IBMQ devices, then we get the result as shown in Table 6. We could not compare the execution time with the state-of-the-art work [14] as execution time is not included in the research work [14].

TABLE 6 Execution time of the circuit in Figure 12

Device	Runtime
IBMQ_Singapore	1404s
IBMQ_Cambridge	1210s
IBMQ_Rochester	1230s
IBMQ_16_Melbourne	546s

The Qiskit code for generating the simulation graphs is given in the GitHub. The link is https://github.com/abhaduri77/k-clique-code.git.

7 | COMPLEXITY ANALYSIS

The proposed combinatorial approach for the synthesis of the k-clique problem in this paper is based on Grover's algorithm, which gives quadratic speed up than its classical counterpart. As the total number of input vertices are n and clique size is k, the oracle and the diffusion of Grover's algorithm needs to be iterated $O\left(\sqrt{\frac{\binom{n}{k}}{m}}\right)$ to find m possible cliques with high probability. Whereas in [14], the authors try to find out only one clique; hence, the complexity is $O\left(\sqrt{\binom{n}{k}}\right)$.

8 | CONCLUSION

An automated and generalised approach of quantum circuit synthesis for the k-clique problem using Grover's algorithm with $O\left(\sqrt{\frac{\binom{n}{k}}{m}}\right)$ iterations for any given undirected and unweighted graph has been proposed in this paper, which gives all cliques of size k as output. When k is very small in comparison to a big graph, the proposed way to solve the k-clique problem is cost-effective in terms of qubit and circuit depth. When $k = 3$, the triangle finding problem is used as an example of the k-clique problem to demonstrate the efficacy of our proposed strategy to solving the k-clique problem. In a classical-quantum hybrid context, the algorithm for the k-clique problem is also utilised to solve the maximum clique problem. Also, we devised and proposed an automated end-to-end framework for translating the clique issue to any existing quantum computer. NISQ devices and the i*ibmq_qasm_simulator* are used to implement the produced circuits. According to the findings, contemporary practical quantum computers are constrained by a number of significant constraints, including the maximum number of usable qubits, various types of noise, and circuit depth. We will try to apply some error mitigation strategies in the future scope of this paper to reduce the error of the output in the real quantum device so that we can get the intended result. In building the circuit for the clique problem, we will also strive to reduce the number of gates and ancilla qubits. These will aid in the implementation of the clique issue for graphs with big k.

ACKNOWLEDGMENTS

All the authors contributed equally to this work.

CONFLICT OF INTEREST

There is no conflict of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in https://github.com/abhaduri77/k-clique-code.git.

ORCID

Arpita Sanyal Bhaduri https://orcid.org/0000-0002-9707-7172
Amit Saha https://orcid.org/0000-0003-2583-9825

REFERENCES

1. Hao, G., et al.: A quantum algorithm for finding a Hamilton circuit. Commun. Theor. Phys. 35(4), 385 (2001)
2. Hao, G., Long, G.L., Feng, L.: Quantum algorithms for some well-known np problems. Commun. Theor. Phys. 37(4), 424 (2002)
3. Shor, P.W.: Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
4. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, STOC'96. pp. 212–219. New York (1996)
5. Magniez, F., Santha, M., Szegedy, M.: Quantum Algorithms for the Triangle Problem. arXiv e-prints, pages quant–ph/0310134 (2003)
6. Buhrman, H., Špalek, R.: Quantum verification of matrix products. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm. SODA 06. Society for Industrial and Applied Mathematics, pp. 880–889. USA (2006)
7. Karp, R.M.: Reducibility among combinatorial problems, vol. 40, pp. 85–103. Springer, Boston (1972)
8. Douik, A., et al.: A tutorial on clique problems in communications and signal processing. IEEE (2020)
9. Beigi, S., Shor, P.W.: On the complexity of computing zero-error and holevo capacity of quantum channels. arXiv (2008)
10. Pavan, M., Pelillo, M.: A new graph-theoretic approach to clustering and segmentation. In: 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, pp. I (2003)
11. Brouwer, A.E., et al.: A new table of constant weight codes. IEEE Trans. Inf. Theory. 36(6), 1334–1380 (1990)
12. Boginski, V., Butenko, S., Pardalos, P.M.: Statistical analysis of financial networks. Comput. Stat. Data Anal. 42(2), 431–443 (2005)
13. Wang, L., et al.: An order-clique-based approach for mining maximal co-locations. Inf. Sci. 179(19), 3370–3382 (2009)
14. Metwalli, S.A., Le Gall, F., Van Meter, R.: Finding small and large *k*-clique instances on a quantum computer. IEEE Trans. Quantum Eng. 1, 1–11 (2020)
15. Fortunato, S.: Community detection in graphs. Phys. Rep. 486(3), 75–174 (2010)
16. Palla, G., et al.: Uncovering the overlapping community structure of complex networks in nature and society. Nature. 435, 814–818 (2005)
17. Sadi, S., Öğüdücü, Ş., Şima Uyar, A.: An efficient community detection method using parallel clique-finding ants. IEEE Congr. Evol. Comput. 1–7 (2010)
18. Matsunaga, T., Yonemori, C., Tomita, E.: Clique-based data mining for related genes in a biomedical database. BMC Bioinforma. 10(205) (2009)
19. Bonner, R.E.: On some clustering techniques. IBM J. Res. Dev. 8(1), 22–32 (1964)
20. IBM. Ibmresearch.qiskitsdk0.5.3documentation. (2018)
21. IBM. Quantum Computer Composer (2018)
22. IBM. Qiskit - Open Source Quantum Information Software Kit (2021)
23. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. 10th Anniversary Edition. Cambridge University Press (2010)
24. Barenco, A., et al.: Elementary gates for quantum computation. Phys. Rev. A. 52, 3457–3467 (1995)
25. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum. 2, 79 (2018)
26. Sanders, B.C.: How to Build a Quantum Computer. 2399–2891. IOP Publishing (2017)
27. IBM Quantum Experience. Ibmq_16_melbourne
28. Aghaei, M., et al.: A hybrid architecture approach for quantum algorithms. J. Comput. Sci. 5, 725–731 (2009)
29. Cirasella, J.: Classical and Quantum Algorithms for Finding Cycles. MSc Thesis, pp. 1–58 (2006)
30. Szegedy, M.: On the Quantum Query Complexity of Detecting Triangles in Graphs. arXiv: Quantum Physics (2003)
31. Le Gall, F.: Improved quantum algorithm for triangle finding via combinatorial arguments. IEEE 55th Annual Symposium on Foundations of Computer Science, pp. 216–225 (2014)
32. Gall, F.L., Nakajima, S.: Quantum algorithm for triangle finding in sparse graphs. Algorithmica. 79(3), 941–959 (2017)
33. Das, Pronaya Prosun, Mozammel, H., Khan, A.: Solving maximum clique problem using a novel quantum-inspired evolutionary algorithm. In: 2015 International Conference on Electrical Engineering and Information Communication Technology, pp. 1–6 (2015)
34. Han, K.H., Kim, J.H.: Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Trans. Evol. Comput. 6(6), 580–593 (2002)
35. Pelofske, E., Hahn, G., Djidjev, H.N.: Solving Large Maximum Clique Problems on a Quantum Annealer. Springer International Publishing (2019)
36. Acasiete, F., et al.: Implementation of quantum walks on IBM quantum computers. Quantum Inf. Process. 19(12), Nov (2020)
37. Zulehner, A., Paler, A., Wille, R.: Efficient mapping of quantum circuits to the IBM QX architectures. In: 2018 Design Automation Test in Europe Conference Exhibition (DATE), pp. 1135–1138 (2018)
38. Li, G., Ding, Y., Xie, Y.: Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. arXiv (2019)
39. Tan, B., Jason, C.: Optimal Layout Synthesis for Quantum Computing. Association for Computing Machinery (2020)
40. Long, G.L.: Grover algorithm with zero theoretical failure rate. Phys. Rev. A. 64(2) (2001)
41. Toyama, F.M., Dijk, W.V., Nogami, Y.: Quantum search with certainty based on modified grover algorithms: optimum choice of parameters. Quantum Inf. Process. 12, 1897–1914 (2013)
42. Yoder, T.J., Low, G.H., Chuang, I.L.: Fixed-point quantum search with an optimal number of queries. Phys. Rev. Lett. 113(21) (2014)
43. Long, G.L., et al.: Phase matching in quantum searching. Phys. Lett. A. 262(1), 27–34 (1999)
44. Long, G., Li, X., Sun, Y.: Phase matching condition for quantum search with a generalized initial state. Phys. Lett. A. 294, 143–152 (2002)
45. Long, G.L.: General quantum interference principle and duality computer. Commun. Theor. Phys. 45(5), 825–844 (2006)
46. Jin, S., et al.: A query-based quantum eigensolver. Quantum Eng 2(3), e49 (2020)
47. Ding, L., Zhou, T.: Implementation of a fixed-point quantum search by duality computer. EPL Europhys. Lett. 126(2), 20004 (2019)
48. Boyer, M., et al.: Tight bounds on quantum searching. Fortschritte Phys. 46(4-5), 493–505 (1998)
49. IBM Quantum Experience. Overview of Quantum Gates

How to cite this article: Sanyal Bhaduri, A., et al.: Circuit design for clique problem and its implementation on quantum computer. IET Quant. Comm. 3(1), 30–49 (2022). https://doi.org/10.1049/qtc2.12029

APPENDIX A

QUANTUM LOGIC GATES

Quantum gates are the unitary operations, where the number of gate collections to be sequentially executed defines the depth of a quantum circuit. Whereas the number of qubits defines the width of the circuit. The matrix representation of the quantum gates of NOT gate, CNOT gate, Hadamard gate and Toffoli gate that are used in the proposed approach is shown in the following Table A1.

Apart from these four gates described in Table A1, Multi-Controlled Toffoli Gate (MCT) is also used to design our proposed circuit for the clique problem. There are n number of inputs and outputs in an n-qubit MCT. This MCT gate passes the first $n-1$ inputs, which are referred to as control bits to the output unaltered. It inverts the nth input, which is referred to as the target bit if the first $n-1$ inputs are all one. An MCT gate is shown in Figure A1, where black dots • represent the control bits and the target bit is denoted by a $\oplus$.

APPENDIX B

ERRORS IN NISQ DEVICES

Qubits are not stable as even a small perturbation in the environment can change the state of a qubit. The error rate for a qubit can be defined as the probability of undesired change in the qubit state. Errors in quantum computers can be classified into two categories: retention-errors and operational-errors.

Retention Errors or Coherence Errors: A qubit can retain data for only a limited time, and this duration is called Coherence Time. There are two types of retention errors that can occur, and there are two metrics to specify the coherence time of a quantum device. A qubit in a high-energy state (state $|1\rangle$) naturally decays to the low-energy state (state $|0\rangle$), and the time constant associated with this decay is called $T1$ Coherence Time. However, there is also a possibility that the qubit might interact with the environment and encounter a phase error even before relaxing into $|0\rangle$ state, and the time constant associated with this decay is called $T2$ Coherence Time. $T2$ indicates the time for a qubit to get affected by the environment.

Operational Errors or Gate Errors: Quantum operations do not give the perfect result. So operations on qubits can also affect their state inectly due to errors, for example, an instruction that rotates the state by some desired angles can introduce extra erroneous rotation. The operational error rate is defined as the probability of introducing an error while performing the operation. Thus, if the depth of a circuit is high, operational errors become huge.

TABLE A1 Matrix representation of quantum gates

Quantum operator	Quantum gate	Matrix representation
Hadamard		$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$
NOT		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
TOFFFOLI		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$

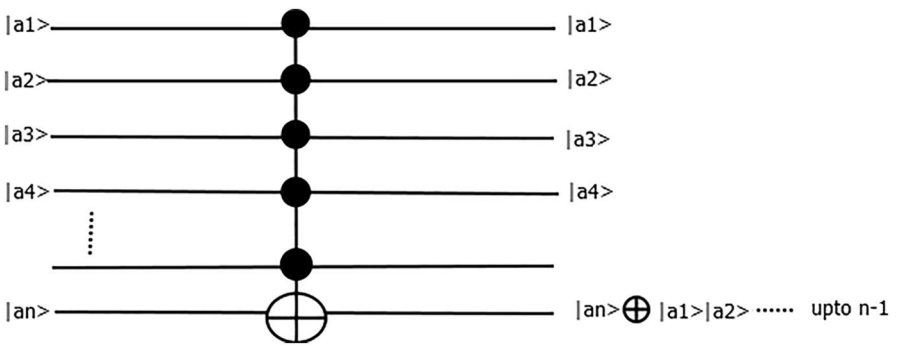

FIGURE A1 Multi-control Toffoli gate