Finding Small and Large k-Clique Instances on a Quantum Computer

Sara Ayman Metwalli
Keio Quantum Computing Center

François Le Gall
Graduate School of Mathematics

Rodney Van Meter
Keio Quantum Computing Center

Keio University
Fujisawa, Japan
sara@sfc.wide.ad.jp

Nagoya University
Nagoya, Japan
legall@math.nagoya-u.ac.jp

Keio University
Fujisawa, Japan
rdv@sfc.wide.ad.jp

Abstract—Algorithms for triangle-finding, the smallest non-trivial instance of the k-clique problem, have been proposed for quantum computers. Still, those algorithms assume the use of fixed access time quantum RAM (QRAM). We present a practical gate-based approach to both the triangle-finding problem and its NP-hard k-clique generalization. We examine both constant factors for near-term implementation on a Noisy Intermediate Scale Quantum computer (NISQ) device, and the scaling of the problem to evaluate long-term use of quantum computers. We compare the time complexity and circuit practicality of the theoretical approach and actual implementation. We propose and apply two different strategies to the k-clique problem, examining the circuit size of Qiskit implementations. We analyze our implementations by simulating triangle finding with various error models, observing the effect on damping the amplitude of the correct answer, and compare to execution on six real IBMQ machines. Finally, we estimate the date when the methods proposed can run effectively on an actual device based on IBM's quantum volume exponential growth forecast and the results of our error analysis.

Index Terms—Quantum computing, clique, graph, Grover's algorithm

I. INTRODUCTION

A *clique* is defined as a complete subgraph over a subset of vertices in an undirected graph. Several computational problems address finding cliques in a given graph. These problems vary based on what information about the clique needs to be found. One such is the k-clique problem, which answers the question, "Given an undirected graph and a positive integer k, does a clique with size k exist?" The k-clique problem is NP-Complete for large values of k, as shown by Karp [1] and Cook [2]. Probably one of the most studied version of the k-clique problem is the triangle finding problem (the 3-clique problem), which has been addressed both classically [3], [4] and quantumly [5], [6]. The best known classical algorithm has time complexity O($n^{2.38}$) while the best known quantum algorithm has time complexity O($n^{1.5}$), where n is the number of nodes in the graph and N is the sizeof the search space (n^2). Several quantum algorithms have also been proposed for the k-clique problem with $k > 3$ [6], [7], [8].

In this paper, we present several implementations based on Grover's algorithm [9]. The asymptotic behavior of the algorithm tells us that quantum computers will offer better scaling than classical computers for a broad range of problems

in the long run. However, we must also assess the constant factors, especially when considering near-term implementation on a NISQ device (section III). We use the algorithm to solve the k-clique problem using Dicke or W states to limit the search space, studying the trade-off against circuit size. We address the theoretical complexity of the above algorithms, which assume the existence of constant access time QRAMs, whereas the best proposed approach would be $O(\log N)$ access time [10]. Moreover, implementation of even logarithmic access time memory is not yet possible. Instead, gate-based representations of graphs are necessary, with cost that exceeds the polynomial gains promised by using Grover for triangle finding (section IV). Our work aims to decrease the gap between theory and implementation by presenting a robust implementation of the k-clique problem in general, regaining the quantum advantage for larger problems.

We implement our proposed scheme using Python and the Quantum Science Kit (Qiskit)[1] developed by IBM [11]. Then based on our data, we predict when the smallest instance of the k-clique problem (triangle finding) can be executed with minimal error on a real quantum computer (section V). Finally, we conclude the paper with some discussions and future directions (section VI).

II. BACKGROUND

In this section, we will lay out some background knowledge on Grover's Search algorithm, the k-clique problem, and the Dicke states.

A. Grover's Search Algorithm

Grover's Search Algorithm answers the question "Given a function $f(x)$, what values of x cause $f(x)$ to evaluate to **True**?". The algorithm presents a framework for tackling the search problem in an unsorted database with complexity O($\sqrt{N}$). It mainly consists of three sections, state preparation, the oracle, and the diffusion operator, which can be seen in Fig. 1.[2]

The algorithmic steps of Grover's search are:

1) Prepare the input in a symmetric-superposition state.

---

[1]The Qiskit version used in this work is '0.15.0'
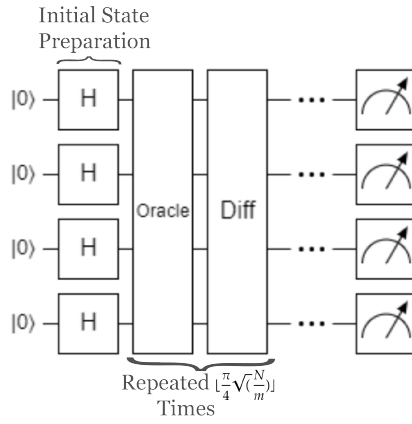[2]Circuit illustration is created using Quirk [12]

Fig. 1. An Overview of Grover's algorithm's Steps

2) Apply Grover's Oracle to the prepared state.
3) Apply the diffusion operator to the oracle's results.
4) Iterate over step 2 and step 3 until the answer is reached.

The first step of Grover's algorithm is preparing the initial state. In the simplest version, the initial state is prepared in an equal superposition over the entire Hilbert space. That is done by applying the Hadamard gate to all input qubits. In this paper, however, we use another approach to create entangled symmetric states as an input to the oracle (Dicke states) to decrease the size of the search space.

After the state preparation comes the oracle. The oracle is a black box function that inverts the answer by flipping its sign. Following that the diffusion operator will magnify the amplitude of the correct state while damping the amplitude of other states.

The diffusion operator is formed by: the inverse of state preparation, $C^{\otimes n}Z$ gate, state preparation. The $C^{\otimes n}Z$ gate cost is $2n-3$ gates, divided into 1 CZ gate and $2n-4$ CCX gates.

The answer's amplitude will grow to a maximum and then decline after the optimal number of iterations `opt_iter` cyclically. The optimal number of times steps 2 and 3 are repeated depends on two factors, the size of the search space N and the number of answers for our search query m (how many cliques in the graph) following Eq. 1 [13]. For example, in Fig. 1, the search space is the entire Hilbert space; in this case (4 qubits), it is 2^4 cases 0000, 0001,....., 1111. Hence, assuming we have a question with only one answer, we will have `opt_iter` of 3.

$$opt\_iter = \left\lfloor \frac{\pi}{4} \sqrt{\frac{N}{m}} \right\rfloor \qquad (1)$$

B. Dicke States

A Dicke state $|D_k^n\rangle$ [14] is a fully symmetric entangled state over the n-qubit Hilbert space with Hamming weight k. For example, given a Hilbert space of 4 qubits, the Dicke state $|D_3^4\rangle$ will be the superposition of $\frac{1}{2}(|1110\rangle + |1101\rangle + |1011\rangle + |0111\rangle)$ as defined in II.1.

The number of basis states with k Hamming weight in a Hilbert space of n qubits is $\binom{n}{k}$.

Definition II.1. Dicke state $|D_k^n\rangle$ is an entangled superposition of all n-states $|s\rangle$ with Hamming weight (hw) k:

$$|D_k^n\rangle = \binom{n}{k}^{-\frac{1}{2}} \sum_{s \in \{0,1\}^n \, s.t. \, \text{hw}(s)=k} |s\rangle. \qquad (2)$$

Dicke states can be implemented in several different ways; we followed the approach proposed in [18] to prepare our Dicke states deterministically. The proposed method computes the Dicke state for any Hamming weight k and n qubits with $O(kn)$ gates and $O(n)$ depth.

1) W States: W state is a Dicke state with Hamming weight 1, as shown by formula 3. The implementation of the W-state preparation we used in this work is the algorithm proposed in [19]. Since the W state is a special case of the Dicke state, the method we chose to implement the Dicke state can be used to implement a W state. However, the approach we used to implement the W state is more efficient in terms of circuit size and depth than the general Dicke state method, but it can't be extended to implement an arbitrary k Hamming weight Dicke state.

$$|W\rangle = \frac{1}{\sqrt{n}}(|100...0\rangle + ... + |01...0\rangle + |00...01\rangle) \qquad (3)$$

We must mention that using W states as our state preparation approach works only for clique size $k = n-1$; otherwise, W states cannot be used.

C. The k-clique Problem

Given an undirected graph (G), if there exists a subset of k vertices that are connected to form a complete graph, then it is said that G contains a k-clique — for example, Fig. 2 represents a graph of 4 nodes which includes a 3-clique between nodes 0,1, and 2.

The k-clique problem asks us to determine if the input graph G contains a k-clique, and if it does, output the vertices forming the clique [20]. A popular variant of this problem only asks us to determine if G contains a k-clique [21]. Classically, several algorithms can find a clique of size k in any graph with efficient complexity $O(n^k)$ [23], [24], [25]. Nevertheless, these problems become NP-complete when k is large [1], [2], [26].

Clique-finding algorithms have many practical applications. One of the main fields they can be used in is chemistry, to find chemicals matching a specific structure [27], to model molecular docking, and to find the binding sites of chemical reactions [28]. They can also be applied to find similar structures within different molecules [29].

III. IMPLEMENTATION

Efficient execution of Grover is a two-fold problem: reducing the number of iterations , and finding a practical implementation of each iteration. In this section, we present

two approaches to implementing the oracle circuit; we will call them the checking-oracle and the incremental-oracle, respectively. The remainder of this section will discuss both implementations in detail. Although either implementation can be used to find any k-clique in any given undirected graph, while explaining how both implementations work, we will consider the simplest case possible, which is a 3-clique problem. In all explanations, the graph in Fig. 2 will be used.
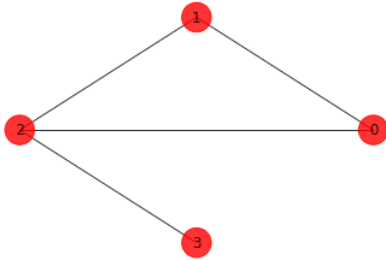

Fig. 2. 4-node graph containing a triangle (3-clique) on nodes 0, 1 and 2. This graph is used in Tables I,III

A. State Preparation

State preparation is the first step of the implementation. Usually, when implementing Grover's algorithm, the states are prepared in an equal superposition of the whole Hilbert space using the Hadamard Gate. Initializing into full superposition needs only n H gates and time complexity $O(1)$

However, let's consider the case represented in Fig. 2. If we wish to search for a 3-clique, then it makes no sense to look for a subgraph with one, two, or even four nodes. Instead, we should consider only subgraphs with k nodes, and then assess whether the induced subgraph contains $\binom{k}{2}$ edges. Searching over a limited space should be faster. However, it will cause a significant increase in the state preparation gate count.
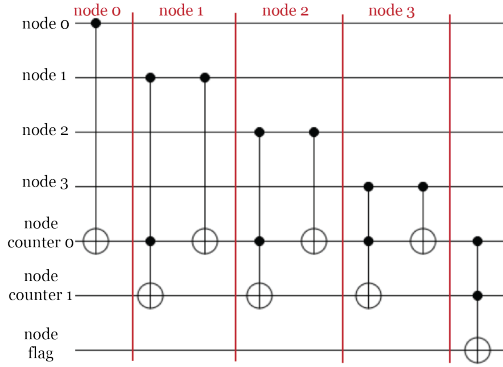
B. Checking-based Oracle

To determine that a triangle exists, we need to confirm that 3 nodes are connected with 3 edges. This counting of nodes and edges is exactly what the oracle circuit should do. In the checking-based oracle, each node in the graph is represented as a qubit, and the edges between them are expressed using one or more multiple-Toffoli $C^{\otimes n}X$ gates connecting specific qubits. After all edges have been counted, the results are checked. The sequence of $C^{\otimes n}X$ gates forms a simple adder that adds one every time an edge is encountered. In the case of a triangle, after the $C^{\otimes n}X$ gates, we need to check that we have precisely three edges (11_2). To check for 11_2, we need two qubits that we will call edges_counter.

In general, we need $\lceil \log \binom{k}{2} \rceil$ qubits to represent the edges_counter. For example, for 4-clique, the edges_counter will be a 3-qubit counter than can count up to 7 (111_2), and for a 5-clique which can count up to 15 (1111_2), the edges_counter will need four qubits and so on.
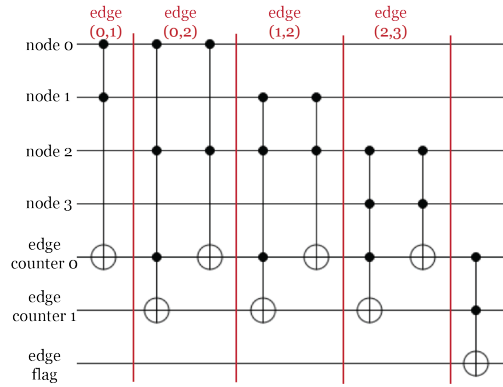
a- Checking-based Oracle's Edges Counter

b- Checking-based Oracle's Nodes Counter

Fig. 3. Checking-based Oracle for the graph in Fig. 2.

Finally, to check if the edges_counter contains the correct value, another $C^{\otimes n}X$ gate needs to be applied, the result of which will be saved in another qubit, edge_flag (Fig. 3-a). A similar circuit is then applied to count nodes; a k-clique should have k nodes. The node_counter needs $\lceil \log k \rceil$ qubits with $C^{\otimes n}X$ between them. If the node_counter contains the correct number of nodes (k), the qubit node_flag will become 1. Fig. 3-b shows the node counting section of the oracle. Finally, after checking for both edges and nodes, a CCX is applied to edge_flag and node_flag and stored in another qubit clique_exists. If we have the correct number of both the edges and the nodes, then a clique of size k exists; otherwise, no clique exists.

C. Incremental -based Oracle

For incremental-based oracle, each node in the graph is represented with a qubit, and the edges are expressed using $C^{\otimes n}X$ gates. The difference between this and the checking-based oracle is in the edges_counter and clique_flag. In this implementation, the edges_counter is replaced with a one qubit edge_flag, the edge_flag becomes 1 if and only if an edge exists between two nodes. That flag is then used as a control qubit controlling an increment circuit that adds one every time it encounters an edge (Fig. 4). In order for the edge_flag to function correctly, we need to uncompute it (reset to $|0\rangle$ state) after each increment.
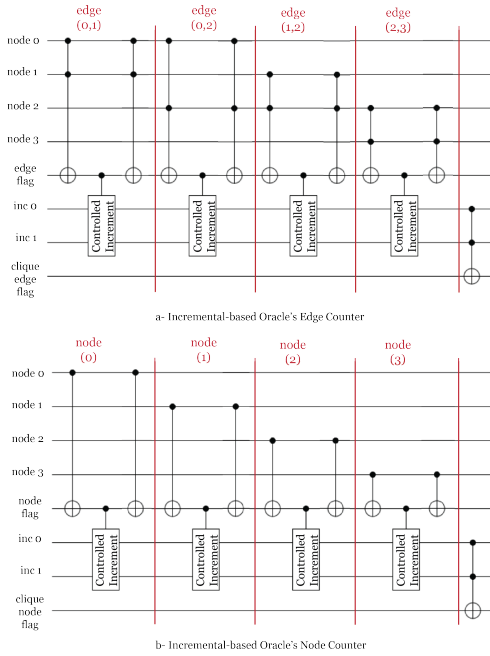
Fig. 4. Incremental-based Oracle for the graph in Fig. 2.

The increment circuit size depends on the clique size; it will need $\lceil \log \binom{k}{2} \rceil$ qubits. For example, when applying the oracle for a triangle ($k = 3$), we will need a 2-qubit increment circuit to count up to 3 or 11_2. Fig 5 shows different sizes of the increment circuit. The circuit for finding the triangle in Fig. 2 needs two qubits for the increment circuit in addition to some ancillary qubits to implement the control functionality. The 2-qubit increment circuit, 3-qubit increment circuit, and 4-qubits increment circuits can be seen in Fig. 4-a. After finishing all the edges in the graph, the qubit `clique_edge_flag` will be 1 only if the number of edges is correct $\binom{k}{2}$. When applying this oracle on the entire Hilbert Space, another circuit to count nodes needs to be added to the oracle 4-b. The size of the controlled increment circuits in the case of counting nodes is $\lceil \log k \rceil$. The number of nodes will be stored in qubit `clique_node_flag`. Once both the edge counter and the node counter sections of the oracle are executed, the `clique_edge_flag` and `clique_node_flag` are used in a CCX to generate the `clique_flag` which will indicate if a clique of size k exists in the graph or not.
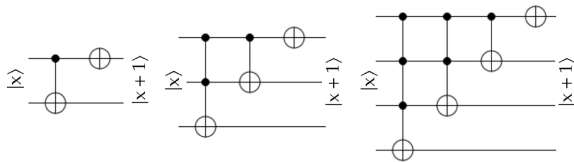

Fig. 5. Different Size Increment Circuits. From right to left, 2-qubit increment, 3-qubit increment and 4-qubit increment circuits.

TABLE I
CIRCUIT SIZE, DEPTH (LENGTH OF CRITICAL PATH) AND NUMBER OF QUBITS NEEDED FOR ALL APPROACHES OF CHECKING-BASED ORACLE AND INCREMENTAL-BASED ORACLE FOR THE OPTIMAL NUMBER OF ITERATIONS FOR THE TRIANGLE FINDING PROBLEM IN FIG. 2

	Checking-based Oracle		
	Full Search Space	W state Prep	Dicke state Prep
Size	214	97	259
Depth	165	79	237
# of Qubits	13	9	9
	Incremental-based Oracle		
	Full Search Space	W state Prep	Dicke state Prep
Size	1000	215	281
Depth	837	131	200
# of Qubits	15	10	10

IV. ANALYSIS

In order to test the efficiency of our implementation, we compared various combinations of the problem variables. To be consistent, the comparison is based on the smallest instance of the problem, i.e. the triangle finding problem. The combinations in the comparison are:

- Grover's algorithm with checking-based oracle over the entire Hilbert space.
- Grover's algorithm with checking-based oracle over limited search space using W-state preparation (W state followed by n NOT gates).
- Grover's algorithm with checking-based over limited search space using Dicke state preparation.
- Grover's algorithm with incremental-based oracle over the entire Hilbert space.
- Grover's algorithm with incremental-based oracle over limited search space using W state preparation (W state followed by n NOT gates).
- Grover's algorithm with incremental-based over limited search space using Dicke state preparation.

We will address the analysis from two perspectives, complexity and practicality, comparing the type of gates and depth of the resultant circuit. In addition, we will also discuss how different state preparation affects the amplitude of the correct answer, using both the ideal-case simulation and gate-error simulation.

A. Gate Count Analysis

First, we will discuss the different circuit sizes for different implementations of the oracle and various state preparations. Again, as a base case, we will compare the different approaches in the case of finding a 3-clique (triangle) in a 4-node graph. Table I shows the different operation counts from both checking-based and incremental-based oracle for the optimal oracle iteration count.

To understand better the numbers in Table I, we need to consider how many times the oracle is repeated. Since Grover's Algorithm is periodic, the optimal number of repetitions of the oracle and diffusion is calculated based on the number of input qubits (number of nodes in the graph in our case) and the number of solutions we want the algorithm to find as shown in Eq. 1. If we are using the entire search space, then

$N = 2^n$ while $m = 1$, and so the optimal number of iterations here will be three iterations. However, if we are using Dicke/ W states to limit our search space, $N = \binom{n}{k}$, $m = 1$ giving an optimal iteration number of one. Although the number of iterations is smaller with state preparation (Dicke/ W state), the circuit may increase in size, based on the state preparation approach followed.Detailed layout of the gates used in every state preparation is found in Table II.

TABLE II
GATE TYPE AND COUNT FOR EACH STATE PREPARATION APPROACH

State Preparation Method	Gate Count	Gate Type
Full search space	4	(Hadamard, 4)
W State	17	(U3, 6), (CNOT, 6), (NOT, 5)
Dicke State	39	(CNOT, 18), (U3, 12), (CCNOT, 6), (NOT, 3)

Circuit size by itself is crucial, but it is more important to check the full list of gates used. More particularly, NOT, CNOT, CCNOT, $C^{\otimes N}$NOT gate counts play an essential factor in whether the circuit can be applied to a real hardware device or not. Table III lists the number of NOT, CNOT, $C^{\otimes N}NOT$ gates in every approach proposed for the optimal number of iterations for each.

B. Simulation Results Analysis

This subsection discusses how the change in state preparation affects the amplitude of the correct answer. To observe this change, we will simulate the circuit twice, once using the ideal-case simulator (QASM Simulator) and another simulation with added gate error. QASM simulates any given circuit assuming ideal qubits and gates with no errors. The results of using the QASM simulation are not realistic for current hardware, and represent the goal of future advancements in quantum computers. For more realistic results, Aer also provides a way to add noise to the gates while assuming perfect qubits.

1) Thermal-relaxation Error: There are several types of errors that can be applied to the QASM simulator. Qiskit Aer offers ten standard error models, including Depolarization Error, Reset Error, and Thermal Error with an option to create user-customized error models [11]. In addition, the user can choose whether to apply the error to all qubits or a specific set of qubits. In our gate-error simulation, we decided on a realistic thermal-error model and applied it to all qubits in the

TABLE III
THE NUMBER OF NOT, CNOT AND CCNOT GATES IN CHECKING-BASED AND INCREMENTAL-BASED APPROACHES FOR THE TRIANGLE FINDING PROBLEM IN FIG. 2

Checking-based Oracle			
	Full Search Space	W state Prep	Dicke state Prep
NOT	25	24	18
CNOT	24	18	90
CCNOT	123	31	37
Incremental-based Oracle			
	Full Search Space	W state Prep	Dicke state Prep
NOT	25	24	18
CNOT	312	42	78
CCNOT	99	17	35

TABLE IV
AVERAGE VALUES OF T_1 AND T_2 IN MICROSECONDS FOR SIX DIFFERENT IBMQ DEVICES

Device Name	T_1 (in μs)	T_2 (in μs)
ibmq_melbourne	55	59
imbq_poughkeepsie	64	65
ibmq_singapore	83	89
ibmq_paris	76	67
ibmq_cambridge	81	39
ibmq_rochester	55	59

algorithm.

Thermal relaxation needs two main parameters defined, T_1 and T_2, together called decoherence times. T_1 is known as the relaxation time constant; it is defined as the time needed for the system to go from state $|1\rangle$ to $|0\rangle$ with probability $\frac{1}{e}$. T_2 is defined as the dephasing time constant, expressing how long the phase of certain qubit stays intact, in other words, it is the time from state $|-\rangle$ to state $|+\rangle$ with probability $\frac{1}{e}$ [39].

Since the value of T_1, T_2 depends on the specific qubits; we took the average T_1, T_2 of the devices when we applied our different circuits. Table IV shows the average values of T_1, T_2, and the names of the six devices used. We should point out that the error rates are determined by gate execution times and the qubit T_1 and T_2 values. The values chosen for the gate execution times are averages based on actual devices as follows, U2 gates take 50 nanoseconds, U3 gates take 100 nanoseconds, CNOT gates take 300 nanoseconds, and finally, the readout will take 1000 nanoseconds[3]. Fig. 6 shows the results of all proposed approaches on each of the six devices. Various observations can be made by looking at the bar chart. Mainly, it can be seen that the W-state preparation approach retains the correct answer better than other methods, followed by the incremental-based Dicke state preparation approach. It can also be seen that the `ibmq_singapore` device has the lowest error among this set of devices, followed by `ibmq_paris`, which is due to these devices having the highest T_1, T_2 among the devices used. In addition, we added another simulation where T_1, T_2 = 200 μs, and 500 μs.Increasing T_1, T_2 by 60% reduced the error rate and the damping in the amplitude of the correct answer by nearly 42%.

2) Device-specific Error: The above case incorporates only memory errors; gates are assumed perfect. Hence, to provide a more realistic effect of noise models in NISQ devices, we applied the device-specific noise models to three of our implementations. The three implementations we chose to apply device-specific models are Checking-based Oracle with W-state Preparation, Incremental-based Oracle with W-state Preparation, Incremental-based Oracle with Dicke state Preparation. We chose these three approaches because they have the highest error tolerance among the six strategies. All three implementations have nine qubits circuits and an ideal (QASM simulator) amplitude of 1.

---

[3]U2, and U3 are basic single-qubit unitary gates presented by Qiskit [40]
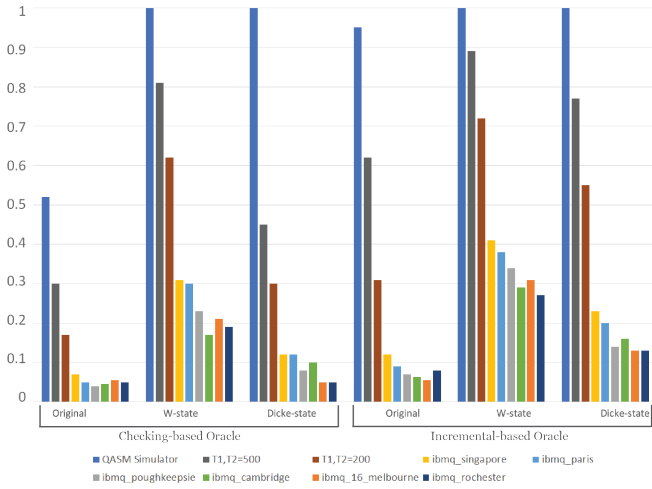
Fig. 6. The amplitude damping effect of memory decoherence, assuming perfect gates. The bars are the probability of finding the correct answer after simulating a perfect machine (leftmost bar in each group), $T_1 = T_2 = 500$, 200 (next two bars) as well as T_1 and T_2 based on the six different IBMQ devices in table IV (last six bars). The figure is sorted based on average error rate from lowest to highest.
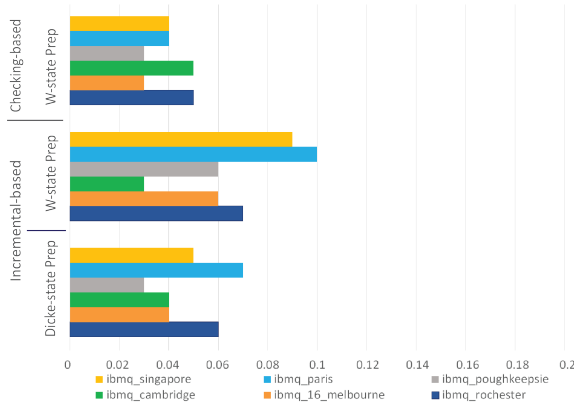

Fig. 7. Probability of finding the correct answer using the Checking-based Oracle with W-state Preparation, Incremental-based Oracle with W-state Preparation, and Incremental-based Oracle with Dicke state Preparation. Data taken on the six different IBMQ devices in table IV.

Considering Fig. 7, we can observe that when executing the Checking-based Oracle with W-state Preparation, Incremental-based Oracle with W-state Preparation, and Incremental-based Oracle with Dicke state Preparation on real IBMQ devices, the error rate increases sharply. Even the implementations with high error tolerance for changes in T_1, T_2, show a significant drop in the amplitude of the correct answer, with error rate ranging from 93% to 96%.

C. Time Complexity Analysis

For our proposed approaches, we can split the time complexity analysis into four main parts: analyzing the number of iterations in Grover's algorithm, the initial state preparation (in case of limited Hilbert space search) complexity, the different oracles and diffusion operators complexities, and finally analyzing the total complexity of the algorithm.

1) Number of Iterations in Grover's Algorithm: The oracle and the diffusion operator are repeated $\lfloor \frac{\pi}{4}\sqrt{\frac{N}{m}} \rfloor = O(\sqrt{\frac{N}{m}})$ times, which depends on the size of the search space and the expected number of answers. Assuming the simplest case, where $m = 1$, such as the case in Fig. 2, the complexity then becomes $O(\sqrt{N})$. Notice that applies to the case when the entire Hilbert space is used, however, if we limit the search space using initial state preparation, the number of iterations then depends also on the size of clique k and becomes $O(\sqrt{\binom{n}{k}})$.

2) State Preparation Complexity: In section III, we proposed the usage of two different state preparation techniques to limit the search space. Using either W-state preparation in case $k = n-1$ or using Dicke state preparation otherwise. As mentioned in section II, we followed the algorithm in [19] to prepare the nodes qubits in a W-state superposition; the algorithm produces a circuit with depth $O(\log n)$ and complexity of $O(n)$. Here n represents the number of qubits involved in the W-state preparation, which is, in our case, the number of nodes $|V|$. Hence the cost of preparing W-states becomes $O(|V|)$. On the other hand, when using the Dicke state preparation proposed in [18], we get a circuit with complexity $O(kn)$ and depth $O(n)$, where k is the clique size, and n is the number of qubits. Therefore, the cost of preparing the Dicke state becomes $O(k|V|)$.

3) Oracle and Diffusion Operator Complexities: First, we will discuss the complexity of the diffusion operator. The diffusion operator consists of the adjoint of the state preparation, a $C^{\otimes n}Z$ gate, and a state preparation, respectively. Hence, we can generalize the complexity of the diffusion operator as $O(state\_prep) + O(C^{\otimes n}Z) + O(state\_prep)$. The cost of the state preparation depends on which approach is used; hence, it will be $O(|V|)$ in case of W-state preparation or $O(k|V|)$ in case of Dicke state preparation. However, the complexity of the $C^{\otimes n}Z$ gate depends on the number of nodes $|V|$, therefore the complexity of the gate will be $O(|V|)$. Consequently, the total complexity of the diffusion operator will become $O(state\_prep) + O(|V|)$.

The complexity of the oracle, however, depends on whether an initial state preparation is used. Regardless of the oracle implementation (checking-based or incremental-based), the primary function of the oracle counts the number of edges and nodes needed to compose a clique of size k, in addition to k itself. So, the complexity of the oracle for the entire Hilbert space is $O(k + |V| + |E|)$. If we use state preparation, we are eliminating the need to count nodes; that is because we only allow states with the specific k nodes activated at any time to be included in the search space, thus eliminating the need to count the nodes in the clique. Hence, the complexity of the oracle when using initial state preparation to limit the search space is $O(k + |E|)$.

4) Algorithm Total Complexity: The total complexity of Grover's algorithm can be expressed as the number of iterations times the cost of one iteration. The number of iterations,

as discussed in previous subsections can be presented as $O(\sqrt{\frac{\binom{n}{k}}{m}})$. Each iteration's cost can be divided into two parts, the oracle's cost, and the diffusion operator's cost. Hence the total complexity becomes $O(\sqrt{\frac{\binom{n}{k}}{m}}) \times (O(\text{oracle}) + O(\text{diffusion operator}))$. This complexity, however, assumes the initial state preparation of states in the entire Hilbert space. That would not be correct if we used W-state or Dicke-state as initial state preparation. In that case, the complexity becomes $O(\text{state\_prep}) + O(\sqrt{\frac{\binom{n}{k}}{m}}) \times (O(\text{oracle}) + O(\text{diffusion operator}))$.

V. DISCUSSION

To estimate the time when our proposed schemes of Grover's algorithm to solve the clique finding problem can be implemented on a real device with minimal error, we need to address two factors, the quantum volume, and the device performance.

A. Quantum Volume

IBM has proposed a single number indicator to describe the quantum processing capabilities of any NISQ device. IBM not only introduced the concept of Quantum Volume (QV) [41]; they also laid out a prediction for the future of their quantum devices, Fig. 8. Their proposed roadmap for the advancement of quantum processor power aims to double the performance every year in order to achieve Quantum Advantage in the near future [42].
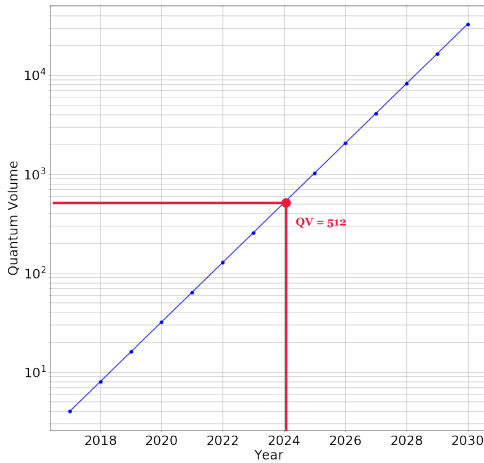

Fig. 8. IBM's Quantum Volume Growth Chart with a highlight on 2024, when QV 512 is hoped to be achieved.

QV represents the ability to run a circuit on an IBM quantum device with at least 2/3 probability of measuring an answer that passes some statistical test [41].

In order to run the smallest instance of our proposed scheme, which is the checking-based oracle with W-state preparation, on a real device with a reasonable probability of success, we need QV of at least 512. QV is calculated as described in Eq. 4 introduced in [41], where d represents the depth of the circuit and n is the number of qubits.

$$QuantumVolume(QV) \approx 2^{min(d,n)} \qquad (4)$$

QV of 2^9 is needed to execute that circuit correctly. Hence, based on that and the growth chart proposed by IBM for future growth, it is expected to reach QV of 512 in early 2024.

B. Device Performance

Even among machines with similar QV, their performance depends on more than just the number of qubits in the machine or the depth of the circuit that can be implemented on it. It also depends on the coupling map. Each machine has a different amount of these errors per qubit, which makes it challenging to estimate the ability to implement any algorithm on a real device based solely on its QV. We analyzed the performance of our top three error-resistant approaches (Checking-based Oracle with W-state Preparation, Incremental-based Oracle with W-state Preparation, Incremental-based Oracle with Dicke state Preparation) on the two machines with the overall best performance `ibmq_singapore` and `ibmq_paris`. We obtained the error_model of both these devices and modified it in three ways in order to understand which factor affects the overall error most. We changed the thermal relaxation error by modifying T_1, T_2 while keeping all other errors untouched, we then did the same but with gate error, and finally, we edited both the thermal relaxation error and the gate error together.

The difference in error rate due to modifications (changing T_1, T_2, and gate error) can increase the device error up to 7.5% and decrease down to 20.5%. It can also be seen that the incremental-based approach with W-state preparation has the most decrease in error rate, especially when modifying both T_1, T_2, and the gate error. Finally, we can see that changing T_1, T_2, only leads to better results than modifying the gate error only. The reason for modifying the execution parameters caused the error rate to increase in some cases is the fact that the values of T_1, T_2, or those of gate errors are not constant. They depend on many factors, such as machine maintenance, and the date on which experiments had been conducted.[4]

VI. CONCLUSION

In this paper, we proposed two approaches to utilizing Grover's Algorithm to solve the k-clique problem on a NISQ device, with theoretical asymptotic performance for long-term use. We analyzed the performance of the proposed approaches from different perspectives, such as gate count, gate type, and time complexity, we also analyzed the performance of our method via simulation of six different IBMQ devices. In addition, we showed how theory and implementation could be far apart when it comes to quantum algorithm complexity, due to assumptions about hardware capabilities. Finally, we estimated the closest time our proposed application can be executable with minimal error on a real NISQ device based on the growth chart of quantum processing power introduced by IBM and the current performance of NISQ devices.

[4] Experiments on the real machines to extract these results were conducted from April 15 to April 25, 2020.

REFERENCES

[1] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[2] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.

[3] J. L. Castellanos, S. Gómez, and V. Guerra, "The triangle method for finding the corner of the l-curve," *Applied Numerical Mathematics*, vol. 43, no. 4, pp. 359–373, 2002.

[4] V. Vassilevska Williams, J. R. Wang, R. Williams, and H. Yu, "Finding four-node subgraphs in triangle time," in *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2014, pp. 1671–1680.

[5] F. Le Gall, "Improved quantum algorithm for triangle finding via combinatorial arguments," in *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE, 2014, pp. 216–225.

[6] F. Magniez, M. Santha, and M. Szegedy, "Quantum algorithms for the triangle problem," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 413–424, 2007.

[7] A. M. Childs, E. Farhi, J. Goldstone, and S. Gutmann, "Finding cliques by quantum adiabatic evolution," *arXiv preprint quant-ph/0012104*, 2000.

[8] A. M. Childs and J. M. Eisenberg, "Quantum algorithms for subset finding," *arXiv preprint quant-ph/0311038*, 2003.

[9] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[10] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum random access memory," *Physical review letters*, vol. 100, no. 16, p. 160501, 2008.

[11] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C. Chen *et al.*, "Qiskit: An open-source framework for quantum computing," *Accessed on: Mar*, vol. 16, 2019.

[12] [Online]. Available: https://algassert.com/quirk

[13] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, "Tight bounds on quantum searching," *Fortschritte der Physik: Progress of Physics*, vol. 46, no. 4-5, pp. 493–505, 1998.

[14] R. H. Dicke, "Coherence in spontaneous radiation processes," *Phys. Rev.*, vol. 93, pp. 99–110, Jan 1954. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.93.99

[15] S. K. Özdemir, J. Shimamura, and N. Imoto, "A necessary and sufficient condition to play games in quantum mechanical settings," *New Journal of Physics*, vol. 9, no. 2, p. 43–43, Feb 2007. [Online]. Available: http://dx.doi.org/10.1088/1367-2630/9/2/043

[16] R. Prevedel, G. Cronenberg, M. S. Tame, M. Paternostro, P. Walther, M. S. Kim, and A. Zeilinger, "Experimental realization of dicke states of up to six qubits for multiparty quantum networking," *Physical Review Letters*, vol. 103, no. 2, Jul 2009. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.103.020503

[17] G. Tóth, "Multipartite entanglement and high-precision metrology," *Physical Review A*, vol. 85, no. 2, Feb 2012. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.85.022322

[18] A. Bärtschi and S. Eidenbenz, "Deterministic preparation of dicke states," in *International Symposium on Fundamentals of Computation Theory*. Springer, 2019, pp. 126–139.

[19] D. Cruz, R. Fournier, F. Gremion, A. Jeannerot, K. Komagata, T. Tosic, J. Thiesbrummel, C. L. Chan, N. Macris, M.-A. Dupertuis *et al.*, "Efficient quantum algorithms for ghz and w states, and implementation on the ibm quantum computer," *Advanced Quantum Technologies*, vol. 2, no. 5-6, p. 1900015, 2019.

[20] G. Valiente, *Algorithms on trees and graphs*. Springer Science & Business Media, 2013.

[21] M. Pelillo, "Encyclopedia of optimization, chapter heuristics for maximum clique and independent set," 2001.

[22] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM Journal on computing*, vol. 14, no. 1, pp. 210–223, 1985.

[23] V. Vassilevska, "Efficient algorithms for clique problems," *Information Processing Letters*, vol. 109, no. 4, pp. 254–257, 2009.

[24] M. Regneri, "Finding all cliques of an undirected graph," in *Seminar current trends in IE WS jun*, 2007.

[25] J.-M. Bourjolly, G. Laporte, and G. Pesant, "Heuristics for finding k-clubs in an undirected graph," *Computers & Operations Research*, vol. 27, no. 6, pp. 559–569, 2000.

[26] B. Rossman, "The monotone complexity of k-clique on random graphs," *SIAM Journal on Computing*, vol. 43, no. 1, pp. 256–279, 2014.

[27] N. Rhodes, P. Willett, A. Calvet, J. B. Dunbar, and C. Humblet, "Clip: similarity searching of 3d databases using clique detection," *Journal of chemical information and computer sciences*, vol. 43, no. 2, pp. 443–448, 2003.

[28] F. S. Kuhl, G. M. Crippen, and D. K. Friesen, "A combinatorial algorithm for calculating ligand binding," *Journal of Computational Chemistry*, vol. 5, no. 1, pp. 24–34, 1984.

[29] N. R. Council *et al.*, *Mathematical challenges from theoretical/computational chemistry*. National Academies Press, 1995.

[30] W. H. Day and D. Sankoff, "Computational complexity of inferring phylogenies by compatibility," *Systematic Biology*, vol. 35, no. 2, pp. 224–229, 1986.

[31] D. Lin, X. Wang, and M. Yung, *Information security and cryptology: 11th International Conference, Inscrypt 2015, Beijing, China, November 1-3, 2015, revised selected papers*. Springer, 2016.

[32] R. Samudrala and J. Moult, "A graph-theoretic algorithm for comparative modeling of protein structure," *Journal of molecular biology*, vol. 279, no. 1, pp. 287–302, 1998.

[33] V. Spirin and L. A. Mirny, "Protein complexes and functional modules in molecular networks," *Proceedings of the national Academy of sciences*, vol. 100, no. 21, pp. 12 123–12 128, 2003.

[34] A. M. Childs, E. Schoute, and C. M. Unsal, "Circuit transformations for quantum architectures," *arXiv preprint arXiv:1902.09102*, 2019.

[35] P. Andrés-Martínez and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Physical Review A*, vol. 100, no. 3, p. 032308, 2019.

[36] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, ser. CGO 2018. New York, NY, USA: ACM, 2018, pp. 113–125. [Online]. Available: http://doi.acm.org/10.1145/3168822

[37] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. V. Meter, "Extracting Success from IBM's 20-Qubit Machines Using Error-Aware Compilation," 2020, to appear; preprint available as arXiv:1903.10963.

[38] B. Nash, V. Gheorghiu, and M. Mosca, "Quantum circuit optimizations for NISQ architectures," *Quantum Science and Technology*, vol. 5, no. 2, p. 025010, mar 2020. [Online]. Available: https://doi.org/10.1088/2F2058-9565/2Fab79b1

[39] X. Hu, R. de Sousa, and S. D. Sarma, "Decoherence and dephasing in spin-based solid state quantum computers," in *Foundations Of Quantum Mechanics In The Light Of New Technology: ISQM—Tokyo'01*. World Scientific, 2002, pp. 3–11.

[40] "Elementary operations qiskit 0.10.4 documentation." [Online]. Available: https://bit.ly/367Dpcw

[41] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, no. 3, p. 032328, 2019.

[42] D. Gil and W. M. Green, "The future of computing: Bits+ neurons+ qubits," *arXiv preprint arXiv:1911.08446*, 2019.