

## Solving various NP-hard problems using exponentially fewer qubits on a quantum computer

Yagnik Chatterjee,<sup>1,2,\*</sup> Eric Bourreau,<sup>2,†</sup> and Marko J. Rančić<sup>1,‡</sup><sup>1</sup>*TotalEnergies, Tour Coupole - 2 place Jean Millier, 92078 Paris la Défense cedex, France*<sup>2</sup>*LIRMM, Université de Montpellier, CNRS, 161 rue Ada, 34392 Montpellier Cedex 5, France*

(Received 5 January 2024; accepted 13 May 2024; published 30 May 2024)

NP-hard problems are not believed to be exactly solvable through general polynomial time algorithms. Hybrid quantum-classical algorithms to address such combinatorial problems have been of great interest in the past few years. Such algorithms are heuristic in nature and aim to obtain an approximate solution. Significant improvements in computational time and/or the ability to treat large problems are some of the principal promises of quantum computing in this regard. The hardware, however, is still in its infancy and the current noisy intermediate-scale quantum (NISQ) computers are not able to optimize industrially relevant problems. Moreover, the storage of qubits and introduction of entanglement require extreme physical conditions. An issue with quantum optimization algorithms such as the quantum approximate optimization algorithm is that they scale linearly with problem size. In this paper, we build upon a proprietary methodology which scales logarithmically with problem size—opening an avenue for treating optimization problems of unprecedented scale on gate-based quantum computers. To test the performance of the algorithm, we first find a way to apply it to a handful of NP-hard problems: *Maximum Cut*, *Minimum Partition*, *Maximum Clique*, *Maximum Weighted Independent Set*. Subsequently, these algorithms are tested on a quantum simulator with graph sizes of over a hundred nodes and on a real quantum computer up to graph sizes of 256. To our knowledge, these constitute the largest realistic combinatorial optimization problems ever run on a NISQ device, overcoming previous problem sizes by almost tenfold.

DOI: [10.1103/PhysRevA.109.052441](https://doi.org/10.1103/PhysRevA.109.052441)

## I. INTRODUCTION

NP-hard problems are problems that do not have algorithms that can give an exact solution in polynomial time, whereas it is “easy” to verify the solution if it is known [1–3]. While finding exact solutions to large problems is difficult, there exist many algorithms that find approximate solutions to these problems [4–7]. In the scope of quantum computing, a huge amount of research has been carried out on hybrid quantum-classical algorithms [8–20]. In such algorithms, quantum circuit measurements are used in tandem with a classical optimization loop to obtain an approximate solution.

One of the most commonly used hybrid algorithms is the quantum approximate optimization algorithm (QAOA) [8,21–24]. One of the main drawbacks of the QAOA is that the number of qubits required scales linearly with problem size [25]. This means that a graph of  $n$  nodes would require an  $n$ -qubit quantum computer to be solved. At the moment, the largest available universal gate-based quantum computer is IBM’s Osprey device, containing 433 qubits. All the qubits are not of the same quality and the larger the problem, the

more likely it is to obtain noisier results due to the presence of qubits with higher error rates. Moreover, these qubits are not all-in-all connected, meaning that in case of large sized problem, numerous SWAP gates would have to be used in order to run the circuit, leading to more noise.

It is therefore not surprising that a smaller scale quantum computer is likely to provide much better results than a larger one. In light of this, an algorithm to encode the *Maximum Cut* problem on a quantum computer using logarithmically fewer qubits was developed [26]. This encoding allows us to represent much larger problems using a fairly small number of qubits. Therefore a *Maximum Cut* problem with a graph of  $n$  nodes can be represented using only  $\lceil \log_2 n \rceil$  qubits.

Since the developed algorithm deals specifically with solving the *Maximum Cut* problem, a logical extension of this algorithm would be to expand the applicability of the algorithm to other problems. This can be approached in two ways, as shall be demonstrated in the following sections.

The paper is structured as follows: In Sec. II B, we describe in detail the logarithmic encoding of the *Maximum Cut* problem on a quantum computer. In Sec. II C, we show how this algorithm can be applied on a variety of NP-hard problems by converting them, directly or indirectly, to the *Maximum Cut* problem. In Sec. II D, we show how any quadratic unconstrained binary optimization problem (QUBO) problem can be treated using the logarithmic encoding. In Sec. III, experimental results of all the methods described in the previous sections are shown. Notably, we show quantum simulator results with instances of sizes of over a hundred nodes or

\*yagnik.chatterjee@totalenergies.com

†eric.bourreau@lirmm.fr

‡Present address: University of Heidelberg, Institute of Computer Engineering, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany; marko.rancic@ziti-uni-heidelberg.de

objects, as well as quantum hardware (QPU) results for problem sizes up to 256.

## II. METHODS

### A. An introduction to the quantum model of computation

Quantum computing [27–30] presents a new way of doing computations by making use of fundamental properties of quantum mechanics such as superposition and entanglement. A classical bit consists of two possible states, 0 and 1. In quantum computing the first building block is the quantum bit or qubit. Much like a classical bit, a qubit has two basis states  $|0\rangle$  and  $|1\rangle$ .

However unlike a classical bit, a qubit can exist in any linear combination of  $|0\rangle$  and  $|1\rangle$ . Let  $|\psi\rangle$  define the state of a qubit, then it can be mathematically defined as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad (1)$$

where  $\alpha$  and  $\beta$  are complex coefficients such that  $|\alpha|^2 + |\beta|^2 = 1$ . The coefficients  $\alpha$  and  $\beta$  are such that  $|\alpha|^2$  and  $|\beta|^2$  represent the probability of a qubit being in the state  $|0\rangle$  and  $|1\rangle$ , respectively. The existence of qubits in all the intermediate states between 0 and 1 is called superposition.

Another important concept is that of entanglement. Two qubits are said to be entangled if they cannot be represented as a product state. This means that they can only be written in a combined superposition of all possible two-qubit basis states. Let  $|\Psi\rangle$  be an  $N$ -qubit state. It can be represented mathematically as follows:

$$|\Psi\rangle = \sum_{i \in \{0,1\}^N} c_i |i\rangle. \quad (2)$$

To completely describe the  $N$ -qubit state, we require  $2^N$  coefficients. If there is no entanglement then each qubit can be expressed separately. From Eq. (1) it can be seen that we need two coefficients to encode a single qubit and therefore we would need only  $2N$  coefficients to represent  $N$  nonentangled qubits. Entanglement is therefore of paramount importance in quantum computing because it helps encode much more information.

The specific model of quantum computation used in this paper is called gate-based quantum computing. There exist other types of quantum computing such as quantum annealing [31], topological quantum computing [32], and measurement-based quantum computing [33]. These are, however, beyond the scope of this paper. In the gate-based model we create a quantum circuit which is a combination of qubits and operations on qubits.

Operations on qubits are known as quantum gates. Mathematically, these can be represented by square matrices. A important property of gates is that they must be Hermitian. There are two basic types of gates:

(1) *Single qubit gates*. They act on a single qubit and alter the state of the qubit.

(2) *Multiple qubit gates*. These gates act on multiple qubits. These are also referred to as entangling gates. The simplest entangling gate is the CNOT gate.

All quantum operations can be represented using single qubit gates and the CNOT gate. The gate-based model is therefore a universal model of quantum computation.

Once we have created a circuit out of qubits and gates we finally need to “measure” our qubits, which is equivalent to calculating the expectation value the system. Let all the gates in the circuit be represented by unitary matrix  $U$ . If it is applied to the initial state  $|\Psi\rangle$ , we have the following final state:

$$|\Psi'\rangle = U|\Psi\rangle. \quad (3)$$

The expectation value of a measurable  $H$  is given by the following expression:

$$\langle H \rangle = \langle \Psi' | H | \Psi' \rangle. \quad (4)$$

A measurable can be mathematically represented by a Hermitian matrix.

Since quantum computers were first theorized in the 1980s, the theoretical advantages of quantum algorithms compared with their classical counterparts have been proven for several problems [34–36]. However, implementing them requires a significant amount of high-quality quantum resources. Specifically, they need quantum computers with many qubits as well as the ability to handle a lot of quantum operations (gates) without generating much noise. In other words, they should be able to run deep quantum circuits on several qubits. Such quantum computers might become available in the medium term but at present we have noisy intermediate-scale quantum (NISQ) computers, which cannot handle these algorithms. This has led to a significant amount of research in the development of hybrid algorithms that can run on currently available hardware.

### B. A qubit-efficient *Maximum Cut* algorithm

Contemporary quantum optimization algorithms in general scale linearly with problem size. This means that if the problem consists of an  $n$  node graph, the algorithm will require  $n$  qubits to solve the problem. Note that to solve a problem here implies to obtain an approximate solution. Following Ref. [26], we present an algorithm that scales logarithmically with the problem size. For a problem of size  $n$ , the number of qubits required is  $\lceil \log_2 n \rceil$ .

#### 1. Description of the algorithm

Recall first the definition of *Maximum Cut*:

*Maximum Cut*

**Input:** A weighted graph  $G(V, E, w)$ .

**Task:** Find  $x \in \{1, -1\}^{|V|}$  that maximizes  $\sum_{ij} w_{ij} \frac{(1-x_i)(1+x_j)}{4} \forall \{(i, j) \in E\}$ , where  $w_{ij}$  are the weights on the edges.

Given a graph  $G(V, E)$ , the *Maximum Cut* can be represented using the graph Laplacian matrix. The graph Laplacian is defined as follows:

$$L_{ij} = \begin{cases} \text{degree}(i) & \text{if } i = j \\ -w_{ij} & \text{if } i \neq j \text{ and } (i, j) \in E \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Note that the degree here is a weighted degree.

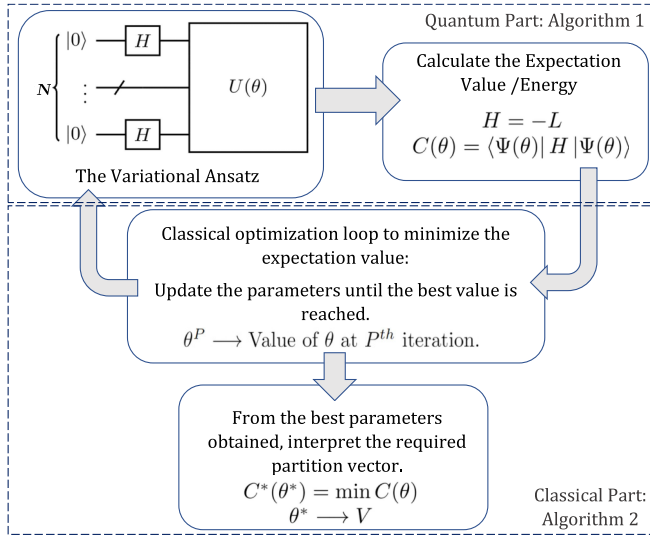


FIG. 1. Diagrammatic representation of the hybrid quantum-classical algorithm.

The *Maximum Cut* value is given by the following equation [37]:

$$\text{Maximum Cut} = \frac{1}{4} x^T L x, \quad (6)$$

where  $L$  is the Laplacian matrix and  $x \in \{1, -1\}^{|V|}$  is the bi-partition vector.

Due to fact that the Laplacian is a Hermitian matrix, it resembles a Hamiltonian of an actual physical system. The quantum analog of Eq. (6) is

$$C(\theta_1 \cdots \theta_n) = 2^{n-2} \langle \Psi(\theta_1 \cdots \theta_n) | L | \Psi(\theta_1 \cdots \theta_n) \rangle, \quad (7)$$

where  $L$  is the Laplacian matrix of the graph,  $|\Psi\rangle$  is the parametrized ansatz,  $n$  is the size of the graph, and  $\theta = \{\theta_1 \cdots \theta_n\}$  are the variables to be optimized.  $2^{n-2}$  is the normalization constant.

As described in Fig. 1, we have designed a variational algorithm that finds a good approximation to the best *Maximum Cut*. Starting from the initial values of  $\theta$  parameters, we call a quantum circuit to evaluate the objective function (Algorithm 1) and run a classical black box optimization loop over the  $\theta$  parameters (Algorithm 2). As a result, we obtain  $\theta^*$  to evaluate the best solution.

To evaluate the expectation value  $C$  on a quantum computer, first we need to create the ansatz  $|\Psi(\theta_1 \cdots \theta_n)\rangle$ . To do this the following steps are required:

- (1) We define a function  $R(\theta_k)$  as follows:

$$R(\theta_k) = \begin{cases} 0 & \text{if } 0 \leq \theta_k < \pi \\ 1 & \text{if } \pi \leq \theta_k < 2\pi. \end{cases} \quad (8)$$

Therefore,

$$\exp[i\pi R(\theta_k)] = \begin{cases} 1 & \text{if } 0 \leq \theta_k < \pi \\ -1 & \text{if } \pi \leq \theta_k < 2\pi. \end{cases} \quad (9)$$

The point of doing this is that not all classical optimizers accept binary variables. The function  $R$  converts a continuous variable into a binary one, which is what we need.

(2) Given a graph  $G(V, E)$  such that  $|V| = n$  and  $|E| = m$ , to create the ansatz we first define the number of qubits

**ALGORITHM 1.** Log encoding of *Maximum Cut*: Building the objective function.

---

**Input:** Laplacian matrix of a graph  $G(V, E)$

- 1  $L \leftarrow$  Graph Laplacian of size  $|V| \times |V|$
- 2  $N \leftarrow \lceil \log_2 |V| \rceil$
- 3  $L^* \leftarrow \begin{bmatrix} L & \mathbb{O}_{2^{N-|V|}} \\ \mathbb{O}_{2^{N-|V|}} & \mathbb{O}_{2^{N-|V|}} \end{bmatrix}$
- 4  $H \leftarrow \frac{1}{n} \sum_{i=1}^n \text{Tr}(J_i \cdot L^*) J_i$  where  $J = \{\prod_{k=1}^N S^{\otimes k}\}$
- 5  $\theta \leftarrow$  List of  $|V|$  parameters
- 6 **Function** EvalCost( $\theta$ ):
- 7    $Q \leftarrow$  Quantum Circuit of  $N$  qubits
- 8   Add Hadamard gate to each Qubit
- 9    $U \leftarrow$  diagonal gate  $\text{diag}(\theta, R)$
- 10   Apply  $U$  to  $Q$
- 11    $F \leftarrow \text{ExpectationValue}(Q, H)$
- 12   **return**  $2^{|V|-2} F$
- 13

---

required as follows:

$$N = \lceil \log_2 n \rceil. \quad (10)$$

When the number of nodes are not an exact power of two, we can adjust  $L$  to be of size  $2^N$  by adding null matrices of size  $2^N - |V|$ ,  $\mathbb{O}_{2^N - |V|}$ , as shown in line 3, Algorithm 1.

(3) Create a quantum circuit and apply a Hadamard gate to all the qubits to achieve an equal superposition of the states (lines 7 and 8, Algorithm 1).

(4) To the circuit, apply a diagonal gate  $U$  (line 9, Algorithm 1) of the following form:

$$U(\theta) = \begin{bmatrix} e^{i\pi R(\theta_1)} & 0 & 0 & \cdots \\ 0 & e^{i\pi R(\theta_2)} & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & e^{i\pi R(\theta_n)} \end{bmatrix}. \quad (11)$$

Therefore the final ansatz is

$$|\Psi(\theta)\rangle = U(\theta) H^{\otimes N} |0\rangle^{\otimes N}. \quad (12)$$

**ALGORITHM 2.** Log encoding of *Maximum Cut*: Minimizing the objective function.

---

**Input:** EvalCost( $\theta$ )

- 1 **Function** Optimizer(EvalCost( $\theta$ ),  $\theta^{\text{initial}}$ ):
- 2   **repeat**
- 3      $\theta^p \leftarrow \theta$  at  $p^{\text{th}}$  iteration
- 4      $C \leftarrow \text{EvalCost}(\theta^p)$
- 5     **if**  $C$  is sufficiently good **then**
- 6        $C^* \leftarrow C$
- 7       **break**
- 8     **else**
- 9       Update  $\theta^p \rightarrow \theta^{p+1}$
- 10      **continue**
- 11    **end**
- 12    **return**  $C^*$
- 13

---

The state in the above equation is obtained in line 10 of Algorithm 1.

Having an ansatz, we can now define the Laplacian as an observable and evaluate the measurement [as in Eq. (7)] which is the energy of the system. Since the classical optimizer minimizes the cost function we take the negative of the Laplacian matrix. Thus the final cost function is

$$C(\theta) = -2^{n-2} \langle \Psi(\theta) | L | \Psi(\theta) \rangle. \quad (13)$$

To evaluate this expectation value, the Laplacian matrix needs to be converted into a sum of tensor products of Pauli matrices (line 4 Algorithm 1, see Appendix A).

Using classical black-box meta optimizers such as COBYLA, Nelder-Mead or a genetic algorithm (as detailed in Algorithm 2), we then obtain

$$C^*(\theta^*) = \min C(\theta). \quad (14)$$

The final parameters obtained  $\theta^*$  gives the bipartition vector, using Eq. (9).

## 2. Complexity analysis of algorithm

The algorithm helps us represent large problems (by current standards of quantum computing) on a quantum computer. Algorithms like the QAOA, for example, require 128 qubits to represent a 128-node *Maximum Cut* problem. The same problem can be solved by the proposed algorithm using only seven qubits. It therefore has the promise of being able to be applied to interesting and even industrially relevant sizes using the currently available sizes of NISQ QCs.

Here we try to analyze the time complexity of our algorithm with respect to depth and density of the problem.

The number of CNOT gates required for the QAOA ansatz is  $p|E|$ , where  $p$  is the depth of the algorithm and  $|E|$  is the number of edges in the graph. In the worst-case scenario, or when the graph is a clique,  $|E| = |V|^2$ .

The time taken to do a complete measurement will depend the number of Pauli terms in the decomposition of the Hamiltonian matrix. This is because to evaluate every Pauli term we require a *separate* circuit [see Eq. (A4)]. For a graph of size  $|V|$ , we need  $N = \lceil \log_2 |V| \rceil$  qubits. In the worst-case scenario we have  $4^N$  Pauli terms to measure. Therefore, the complexity for a measurement will be  $O(4^N)$ . An increase in density of the graph will increase the number of Pauli terms. Since  $N = \lceil \log_2 |V| \rceil$ , the complexity is therefore  $O(4^{\lceil \log_2 |V| \rceil}) = O(|V|^2)$ .

Note that this is exactly the same complexity as if the expectation were calculated classically. For the vertex whose corresponding  $\theta$  value is between  $[0, \pi]$ , it is put into the first partition. For vertex whose corresponding  $\theta$  values are between  $[\pi, 2\pi]$ , it is put into the second partition. Then the expectation can be calculated if one considers whether all the edges are cut. The process can be finished within  $O(V^2)$ .

However, for sparse matrices, much more efficient schemes to decompose the Hamiltonian into  $m$  summands and calculate the expectation values with less than  $m$  queries to the quantum computer exist (where  $m \ll |V|^2$ ). This is discussed in detail in Appendix E.

In order to calculate one expectation value, we need to evaluate  $O(|V|^2)$  circuits. In a single circuit the number of CNOT

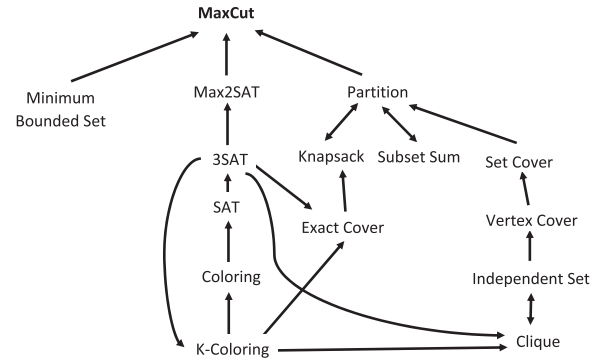


FIG. 2. Graph of *Maximum Cut* transformation family for NP-complete problems.

gates is equal to  $|V| - 1$ ,  $|V|$  being the number of vertices. Hence we have a total of  $|V|^3$  CNOT gates.

While QAOA has  $p|E| \leq p(|V|^2 - |V|)/2$  number of CNOT gates, all our circuits are of depth  $|V|$  which makes QAOA dramatically more sensitive to errors (because it has quadratically more CNOT gates). Moreover, for all practical purposes,  $p \gg 1$  [38], and hence  $\propto p|V|^2$  CNOT gates could be comparable or even greater than our total of  $|V|^3$  CNOT gates.

Finally, in our study to propose a new encoding compatible with NISQ, the search space remains the same as that of the classical search space. A procedure to reduce the number of variables has been presented in Ref. [26]. However, this method is beyond the scope of the current work. Readers should also refer to follow-up studies [39] where the algorithm was evaluated on the *Maximum Cut* problem by using the alternating optimization procedure [40] which scales polynomially in problem size.

## C. Applying the algorithm to other NP-hard problems

A logical next step is to attempt to solve a variety of combinatorial optimization problems using the algorithm. In Karp's paper from 1972 [41], he outlined how we can convert one NP-complete problem into another. A more recent paper [42] lists numerous more such reductions. Figure 2 shows a subset (a transformation family) of these reductions directly or indirectly relating to *Maximum Cut*. Here, we follow a similar logic to convert various NP-hard problems to *Maximum Cut*.

Note, however, that these conversions might not have a one-to-one scaling. For example, an  $n$  variable *Maximum 2-Sat* problem requires us to solve a  $2n$  node *Maximum Cut* problem.

In Karp's paper all the transitions are from one decision problem to another. Usually in classical computing it would be considered trivial to convert a decision problem into an optimization problem. However, our algorithm is inherently an optimization algorithm and moreover will give various results for a various runs. The point being it will not respond well to yes-no decision problems. Therefore, it is important to make reductions between the optimization versions. Moreover it is important to make sure that these conversions support a wide definition of the problems (for example *Maximum 3-Sat* instead of *3-Sat*).



Following are some such polynomial-time reductions of NP-hard problems:

### 1. Minimum Partition to Maximum Cut

#### Minimum Partition

**Input:** A set  $S = \{w : w \in \mathbb{Z}^+\}$ .

**Task:** Find  $A \subseteq S$  that minimizes  $|\sum_{w_k \in A} w_k - \sum_{w_l \notin A} w_l|$ .

This can be converted to the maximum cut problem in the following manner:

- (1) Create a graph such that there is a node for every number.
- (2) For every pair of nodes  $(i, j)$ , connect them using an edge of weight  $w_i * w_j$ .
- (3) The maximum cut value of this graph gives a bi-partition that is equivalent to the minimum partition.

### 2. Maximum 2-Sat to Maximum Cut

#### Maximum 2-Sat

**Input:** A set of  $m$  clauses  $C = \{w_{pq}(x_p + x_q) : x_p, x_q \in X \cup X'\}$  where  $X = \{x : x \in \{0, 1\}\}$ ,  $X' = \{\bar{x} : x \in X\}$ , and  $w_{pq}$  are the clause weights.

**Task:** Find the variable assignment  $X$  that maximizes the combined weight of the satisfied clauses.

The problem is said to be satisfiable if all the clauses are satisfied.

We can convert this problem into the maximum cut problem in the following manner:

- (1) In a graph, assign two nodes for every variable, one for the variable and another for the complement of the variable. Hence there are  $2|X|$  nodes in the graph.
- (2) Draw an edge between the nodes representing the variables and their complements. For example connect  $x_1$  and  $\bar{x}_1$ ,  $x_2$  and  $\bar{x}_2$  and so on. Add a large edge weight (about  $10^m$ ). This is to make sure that the variables and their complements do not fall in the same partition.
- (3) For every clause, add an edge between the respective nodes with edge weight as  $w_{pq}$ .
- (4) The maximum cut of this graph is equivalent to the *Maximum 2-Sat* solution.

### 3. Maximum Clique to Maximum 2-Sat

#### Maximum Clique

**Input:** A graph  $G(V, E)$ .

**Task:** Maximize  $|V'|$  in the graph  $\{G'(V', E') : V' \subseteq V, E' \subseteq E, |E'| = |V'|(|V'| - 1)/2\}$ .

It can be converted to the *Maximum 2-Sat* problem in the following manner:

- (1) Consider a graph  $G(V, E)$  having vertices  $v_i \in V$ . For each vertex  $v_i$  add a variable  $x_i$  and an auxiliary variable  $z$ . We therefore have  $|V| + 1$  variables.
- (2) For every variable add the following two clauses:  $(x_i + z)$  and  $(x_i + \bar{z})$ . Let us refer to these clauses as type A clauses.
- (3) Add the following clauses:  $(\bar{x}_i + \bar{x}_j) \forall (i, j) \notin E$ . We refer to these clauses as clauses of type B.

(4) The clauses of type A ensure that the maximum number of nodes are selected and the clauses of type B make sure that the selected subgraph is a clique.

(5) To the type B clauses, add a large weight. Due to the nature of the algorithm and its susceptibility to errors, we may get solutions that are not cliques at all. Moreover finding a clique and maximizing it are two different problems and by adding weights we make sure that they are not affected by one another.

(6) The *Maximum 2-Sat* problem is solved for this set of clauses. The partition of selected variables form the *Maximum Clique*.

### D. Generalizing the algorithm

The algorithm described above solves, originally, the *Maximum Cut* problem. Various conversions are then used in order to solve other problems. Here, a second, more general approach, shall be described, where any problem which can be written in the form of a quadratic unconstrained binary optimization (QUBO) problem [43] can be solved. Instead of taking the Laplacian matrix as the input, this algorithm takes as input the QUBO matrix of the problem.

First, we define the QUBO matrix. To describe a problem as a QUBO, all the terms in the objective function should be either linear or quadratic. Since the variables in the objective function are binary, a linear term can be easily converted to a quadratic one, since  $x_i^2 = x_i \forall x_i \in \{0, 1\}$ .

Consider the objective function of the following form:

$$P = \sum_{ij} a_{ij} x_i x_j. \quad (15)$$

It can be rewritten as

$$P = (x_1 \ x_2 \ \dots \ x_n) \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}, \quad (16)$$

$$P = x^T Q x, \quad (17)$$

$$Q = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}. \quad (18)$$

$Q$  is the required QUBO matrix.

This matrix cannot be directly used in the algorithm because, in the original *Maximum Cut* algorithm, the variable used belongs to the set  $\{1, -1\}$  and not  $\{0, 1\}$ . To make the equation mathematically consistent, we need to reformulate the QUBO matrix.

Let  $z \in \{1, -1\}$  and  $x \in \{0, 1\}$ , then  $x = (1 - z)/2$ . Equation (15) therefore becomes

$$P = \sum_i a_{ii} \frac{1 - z_i}{2} + \sum_{\substack{ij \\ i \neq j}} a_{ij} \frac{1 - z_i}{2} \frac{1 - z_j}{2}. \quad (19)$$

Note that in the first term  $(1 - z_i)/2$  has been used instead of  $[(1 - z_i)/2]^2$  since  $x_i^2 = x_i$ .

In the search for optimal values of parameters  $z$  we can eliminate the constant terms in  $P$  because they only add a constant shift to the cost function. We can therefore simplify (19) as follows:

$$P = \sum_i a_{ii} \frac{1 - z_i}{2} + \sum_{\substack{ij \\ i \neq j}} a_{ij} \frac{1 - z_i}{2} \frac{1 - z_j}{2} \quad (20)$$

$$= \frac{1}{2} \sum_i a_{ii} (1 - z_i) + \frac{1}{4} \sum_{\substack{ij \\ i \neq j}} a_{ij} (1 - z_i - z_j + z_i z_j) \quad (21)$$

$$= -\frac{1}{2} \sum_i a_{ii} z_i + \frac{1}{4} \sum_{\substack{ij \\ i \neq j}} a_{ij} (-z_i - z_j + z_i z_j). \quad (22)$$

The above *cannot* be represented in a matrix form similar to Eq. (18) since it has linear terms that cannot be quadratized since  $z_i^2 \neq z_i$ .

We therefore need to reformulate the problem. In this reformulation the linear terms are represented in the off-diagonal terms instead of the diagonals. Let us have  $2n$  variables  $\{z_1, z_2, \dots, z_{2n}\}$  where  $z_1, \dots, z_n \in \{1, -1\}$  and  $z_{n+1}, \dots, z_{2n} \in \{1\}$ . Then to represent a linear variable  $z_i$ , we can have the term  $z_i z_{i+n}$  where  $z_{i+n} = 1$ . The Eq. (22) can be rewritten as follows:

$$P = -\frac{1}{2} \sum_i a_{ii} z_i z_{i+n} + \frac{1}{4} \sum_{\substack{ij \\ i \neq j}} a_{ij} (-z_i z_{i+n} - z_j z_{j+n} + z_i z_j). \quad (23)$$

This is our reformulated QUBO, which we call the *spin-QUBO* (sQUBO). This is a matrix of size  $2n \times 2n$ . It requires  $\lceil \log_2 2n \rceil = (1 + \lceil \log_2 n \rceil)$  qubits, or, in other words, one more qubit than the original algorithm. Note that this is still an optimization problem of  $n$  variables since the variables  $z_{n+1}, \dots, z_{2n}$  are fixed.

Using our formulation, we propose that any problem that can be represented in a QUBO format can be solved using the algorithm described in Algorithm 3.

### 1. Maximum Weighted Independent Set using quadratic unconstrained binary optimization

#### Maximum Weighted Independent Set

**Input:** A graph  $G(V, E)$  with node weights  $w_i$ .

**Task:** Find  $x \in \{0, 1\}^{|V|}$  that maximizes  $\sum_i w_i x_i$  such that  $x_i + x_j \leq 1 \forall (i, j) \in E$ .

The *Maximum Weighted Independent Set* problem consists of an objective function and constraints. We can however incorporate the constraints in the objective function as penalty terms. Let  $p$  be the magnitude of the penalty.

$$W = \max \left( -\sum_i w_i x_i + p \left( \sum_{(i,j) \in E} x_i x_j \right) \right). \quad (24)$$

Since  $x_i$  is binary

$$W = \max \left( -\sum_i w_i x_i^2 + p \left( \sum_{(i,j) \in E} x_i x_j \right) \right). \quad (25)$$

**ALGORITHM 3.** Log encoding of a QUBO problem: Building the objective function.

---

**Input:** QUBO Matrix

- 1 Convert *QUBO* to *sQUBO*
- 2  $Q \leftarrow \text{sQUBO}$
- 3  $N \leftarrow \lceil \log_2 2n \rceil$
- 4  $Q^* \leftarrow \begin{bmatrix} Q & \mathbb{O}_{2^{N-n}} \\ \mathbb{O}_{2^{N-n}} & \mathbb{O}_{2^{N-n}} \end{bmatrix}$
- 5  $H \leftarrow \frac{1}{n} \sum_{i=1}^{4^N} \text{Tr}(J_i \cdot Q^*) J_i$  where  $J = \{\prod_{k=1}^N S^{\otimes k}\}$
- 6  $\theta \leftarrow$  List of  $n$  parameters
- 7 **Function** EvalCost( $\theta$ ):
- 8     $QC \leftarrow$  Quantum Circuit of  $N$  qubits
- 9    Add Hadamard gate to each Qubit
- 10     $U \leftarrow$  diagonal gate  $\text{diag}(\theta, R)$
- 11    Apply  $U$  to  $QC$
- 12     $F \leftarrow \text{ExpectationValue}(Q, H)$
- 13    **return**  $F$
- 14

---

Hence we have a QUBO matrix of the following form:

$$Q_{ij} = \begin{cases} -w_i & \text{if } (i, j) \in E \text{ and } i = j \\ \frac{p}{2} & \text{if } (i, j) \in E \text{ and } i \neq j \\ 0 & \text{if } (i, j) \notin E. \end{cases} \quad (26)$$

$Q_{ij}$  can now be used as input in Algorithm 3 to solve the problem.

## III. RESULTS AND DISCUSSIONS

In this section we first show the performance of the algorithm for the *Maximum Cut* problem. We compare the results from our algorithm with the optimal solution achieved using an integer linear program. We test our algorithm on both a quantum simulator and real hardware. Then, the effect of increasing graph density on performance is tested to surpass the sparse examples found in the literature. Finally for *Maximum Cut*, quantum simulator runs of up to 256 nodes are shown. Then we display the results of the *Minimum Partition* problem, which has been solved by converting it to the *Maximum Cut* problem.

Next, the results from the QUBO method are shown. The *Minimum Partition* problem is solved, this time using the QUBO method, and the results are compared with the conversion method.

### A. Maximum Cut

We start by benchmarking the *Maximum Cut* algorithm against classical methods such as 0–1 integer linear programming and Goemans-Williamson method. All graph instances in this section are generated using the *fast\_gnp\_random\_graph()* function of the NETWORKX Python package, with *seed* = 0 for all cases.

Figure 3(a) shows the performance of the algorithm versus the optimal solution obtained using an integer linear program (ILP, see Appendix B). Two different classical optimizers have been used for the runs on the quantum simulator. We

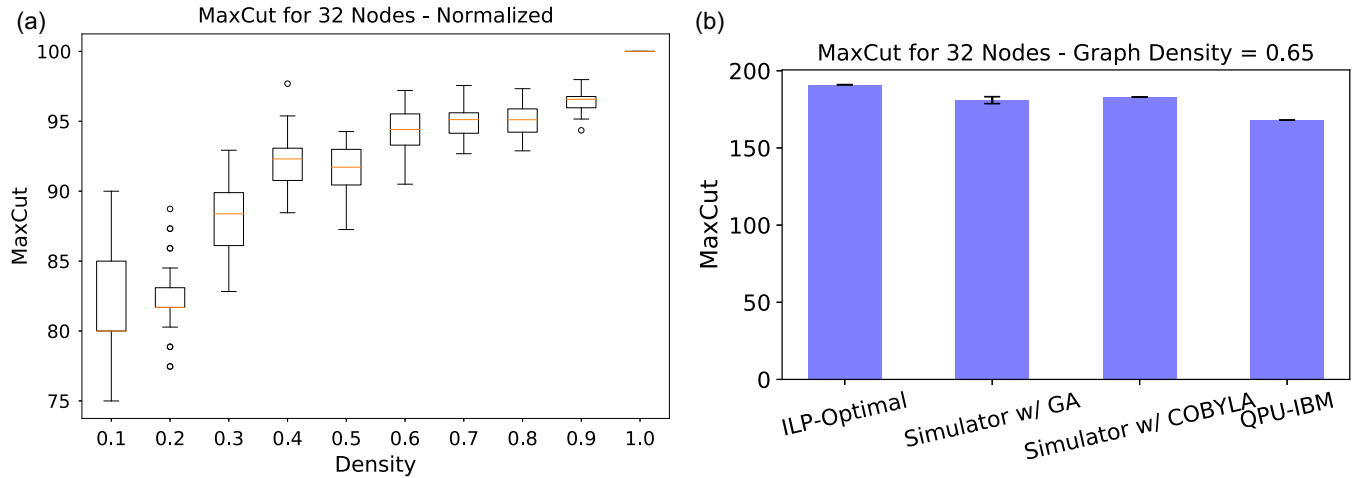


FIG. 3. (a) Performance of the algorithm on a 32-Node graph instance. The QPU result is based on a single run while the simulator results are based on 50 runs. (b) The *Maximum Cut* of 32-node graphs of varying densities. Optimizer used is the genetic algorithm (GA). Data is based on 50 runs for each instance and is normalized using the optimal result obtained using ILP.

can see that both classical optimizers give fairly similar results, 90.04% of the optimal for the genetic algorithm and 91.04% of the optimal for COBYLA. The result from the QPU is slightly worse (83.58% of the optimal), as expected due to the noise present in the current devices. Note that only a single instance has been considered here as opposed to multiple. This is because running algorithms on real hardware is extremely time consuming due to queue times (wait times).

In Fig. 3(b), ten randomly generated 32-node instances are tested with increasing graph density. Here graph density implies the fraction of the total possible edges present in the graph. For each instance, data were collected for 50 runs, using two different types of classical optimizers, COBYLA and the genetic algorithm. In addition, a 0–1 integer linear program (ILP) [44] was used to obtain the optimal result of each of the instances. The ILP data are then used to normalize the simulator data. Hence, the data are in the form of percentage of optimal value. In most instances, the genetic algorithm (GA) performed better than COBYLA. This, however, might be due to the fact that the genetic algorithm is simply able to cover a larger search space. Larger instances might require a larger number of iterations with high computational cost. In these cases, using COBYLA could be more practical. While the GA results vary with each run, the results from COBYLA are the same in each run. This can be seen from the fact that the COBYLA plot has a flat error bar. It is seen that the performance improves with an increase in the density of the

problem. This might be since in graphs of lower density, the choice including or not including one or two nodes can have a relatively significant impact on the final cost function. For example, the minimum, maximum and average cost function for density 0.1 is (30, 36, 32) and for 0.9 is (234, 243, 239). This shows that the variation of cost function is similar for both densities. This higher instability is why we see a higher dispersion on the results in the lower densities. The important point here is that the performance does not degrade with increasing density, a property which will be useful in the sections to follow.

To demonstrate the scalability of the algorithm, we further test the algorithm on problem instances of 64, 128, and 256 nodes (6, 7, and 8 qubits, respectively). For the case of 64 node graphs, as shown in Table I, each instance is run 10 times on the quantum simulator and their mean and standard deviation are shown. The genetic optimizer (GA) is used for all obtained data. Table II shows the parameters of GA used in the algorithm runs. The ILP model solved the problem up to a specified integrality gap of 4%. Also shown are the results of the Goemans-Williamson (GW) method [45]. Since this is an approximate method, a solution range is shown over 50 runs. We can see that, for all cases, the quantum simulator results are nearly or over 90% of the ILP optimal cut. It is seen again that increase in graph density does not degrade the performance.

For 128 and 256 nodes (Tables III and IV), only the GW method is used for benchmarking. This is because the ILP

TABLE I. 64-Node *Maximum Cut* benchmarks.

Graph density	ILP			GW		Quantum simulator		QPU	
	Solution	Time(s)	Gap(%)	Solution	Time(s)	Solution	Time(s)	Solution	Time(s)
0.30	383	7.6	3.92	371.4	0.94	343.9	280	282	383
0.35	443	69.7	3.84	428.7	1.00	400.8	272	365	439
0.40	497	1077.7	3.82	486.2	1.01	454.3	270	380	393
0.45	553	1375.2	3.98	539.6	1.14	512.8	272	446	518

TABLE II. Parameters of the genetic algorithm used in the *Maximum Cut*, *Minimum Partition* and *Maximum Clique* experiments.

GA parameter	Value
max_num_iteration	20
population_size	20
mutation_probability	0.1
elit_ratio	0.05
crossover_probability	0.5
parents_portion	0.3
crossover_type	Uniform
max_iteration_without_improv	None

took longer than two days without converging (Gurobi optimizer) on a PC. The GW mean is based on 50 runs. The quantum solution is compared with the GW upper bound. The GW upper bound is calculated by dividing the worst solution of the GW with 0.878. Table III shows results using a quantum simulator while Table IV shows results obtained using an IBM quantum computer. The *ibmq\_mumbai* backend was used for the instances of size 128 while the *ibmq\_guadalupe* was used for the instance of size 256. The results demonstrate the stability of the results (around 84% of the GW upper bound) as the size increases.

It is highly time consuming to carry out each run on an IBM QPU due to the significant wait times in addition to the inability to run multiple instances or runs simultaneously. This is the reason we decided to demonstrate only one run per instance where possible. The aim of the QPU runs is not to present a performance analysis of the QPU but merely indicative of the current gap between the quantum simulator and the QPU.

It is important to note that the number of GA iterations has a significant impact on the quality of the solution. This is demonstrated in Table V with the instances of size 128. The solutions improve with an increase in GA iterations.

### B. Minimum Partition as a conversion from Maximum Cut

As described in Sec. II C 1, the number partitioning problem can be directly converted into the *Maximum Cut* problem. The graphs hence formed are weighted fully dense graphs.

For the instances, all the numbers used were random integers between 1 and 100. Tests were carried out on the quantum simulator as well as on real hardware from IBM. The classical

TABLE IV. 128- and 256-node *Maximum Cut* results using QPU with GA.

Instance	Q. sol.	GW range	% diff.
Size = 128, density = 0.4	1538	1796–1864	82.5–85.6
Size = 128, density = 0.5	2022	2186–2271	89.0–92.5
Size = 256, density = 0.5	8079	8701–8880	90.9–92.8

optimizer used is GA. Details of the GA parameters are given in Table II.

The results of partition differences have been normalized in the following manner: For a problem with  $N$  numbers, if the partition difference is  $p$ , then the normalized difference is  $p_{\text{norm}} = (50N - p)/50N$ . All our numbers are random integers between 1 and 100, hence 50.5 on an average. For simplicity we use 50 in  $p_{\text{norm}}$ .

The optimal value for each instance is obtained using the integer quadratic model described in Appendix C.

Figure 4 displays the performance of 32, 64, and 128-number *Minimum Partition* converted to *Maximum Cut*. For 32 integers, we have a mean value of 98% and a very small dispersion using the quantum simulator. For 64-number *Minimum Partition*, the mean values are better than 85% for all instances. Moreover, for both problems sizes, despite the fact that the *Minimum Partition* problem leads to a complete graph *Maximum Cut* problem, actual QPUs are able to demonstrate an approximate solution. The problem of size 128 has mean values of about 97% on a quantum simulator.

All QPU runs in this section are done on *ibmq\_mumbai*.

### C. Maximum Clique as a conversion from Maximum Cut

The *Maximum Clique* problem can be converted to the *Maximum Cut* problem by first converting it to the *Maximum 2-Sat* (II C 2) and then from the *Maximum 2-Sat* to *Maximum Cut* (II C 3).

After the conversion, a  $n$ -node *Maximum Clique* problem requires the solution of a  $2(n + 1)$ -node *Maximum Cut*. Table VI shows results for various instances run on a quantum simulator. The classical optimizer used is GA; the details of the GA parameters are given in Table II. It is seen that in half of the instances, the best solution is the optimal solution as obtained using numpy. It should be noted that our approach finds a dense subgraph and then removes the nodes with lowest degree iteratively.

TABLE III. 128- and 256-node *Maximum Cut* benchmarks.

128 nodes					256 nodes			
Graph Density	GW mean	GW upper bound	Quantum solution	Ratio with GW bound (%)	GW mean	GW upper bound	Quantum solution	Ratio with GW bound (%)
0.3	1405.6	1567	1305	83.2	5543.5	6159	5066	82.2
0.4	1837.5	2045	1691	82.7	7264.1	8071	6736	83.5
0.5	2249.6	2489	2103	84.5	8120.3	9910	8367	84.4
0.6	2654.1	2981	2546	85.4	10615.1	11794	9967	84.5



TABLE V. Evolution of *Maximum Cut* solutions with increasing GA iterations on 128-node instances.

GA iterations	Density			
	0.3	0.4	0.5	0.6
10	1286	1704	2090	2523
20	1305	1730	2119	2546
30	1322	1745	2121	2553
40	1322	1747	2145	2577

#### D. *Maximum Weighted Independent Set* using quadratic unconstrained binary optimization method

In this section, results of the *Maximum Weighted Independent Set* problem solved using the QUBO method (Sec. II D 1) is presented.

For each figure the performance of the algorithm is shown. The data are normalized using the optimal solution found using the commercial CPLEX solver.

Figure 5 shows the data for graphs of size 32, 64, and 128. The data for 32, 64, and 128 nodes are based on 50, 50, and 10 runs, respectively. The data represented only take into account the feasible solutions produced. For graphs of size 32, the mean values for all instances are above 80% and the best obtained result is optimal for every instance. For graphs of size 64, the mean values for all instances are above 60% and the best obtained result is on an average over 80%. For 128-node graphs, the solutions are slightly degraded in comparison. Note, however, that the data for 128-node instances are based only on a few runs. Moreover, the performance also depends on the number of GA iterations used in the algorithm run. The details of the GA parameters used are given in Table VII.

Table VIII shows how the performance varies depending upon the number of GA iterations used. For this table, the *Maximum Weighted Independent Set* Instance 4 of size 64 has been taken. An increase in the number of GA iterations improves not only the performance but also the percentage of feasible results.

## IV. CONCLUSION

In this paper, we investigated and further developed methods to logarithmically encode combinatorial optimization problems on a quantum computer. We begin with expanding the work done in Ref. [26], which describes a way to logarithmically encode the *Maximum Cut* problem. We performed several runs of this algorithm with various instances, on the quantum simulator as well as real hardware, using different classical optimizers like COBYLA and the genetic algorithm.

We then reformulate a number of NP-hard combinatorial optimization problems into the *Maximum Cut* problem, either directly or indirectly and solve it on a real quantum computer. We take the *Minimum Partition* problem as an example and solve it by using a reduction as mentioned in Sec. II C 1. This is possible since the algorithm is largely unaffected by increasing the density of the *Maximum Cut* graph in question, since the *Minimum Partition* problem converts into a weighted fully dense graph. Some performance benchmarks of the partition problem have been presented.

We then proceed to present a more general formulation inspired from the structure of the *Maximum Cut* algorithm. We see that instead of using the Laplacian, we can use the QUBO matrix of a problem in order to solve it. We introduce the sQUBO representation of the QUBO matrix for it to be compatible with the algorithm. This therefore opens up the applicability of the algorithm to a wide range of algorithms. The *Maximum Weighted Independent Set* problem is solved using its sQUBO matrix.

The authors will make all data available upon reasonable request.

## ACKNOWLEDGMENTS

Y.C. and M.R. acknowledge funding from European Union's Horizon 2020 research and innovation program, more specifically the NEASQC project under Grant Agreement No. 951821.

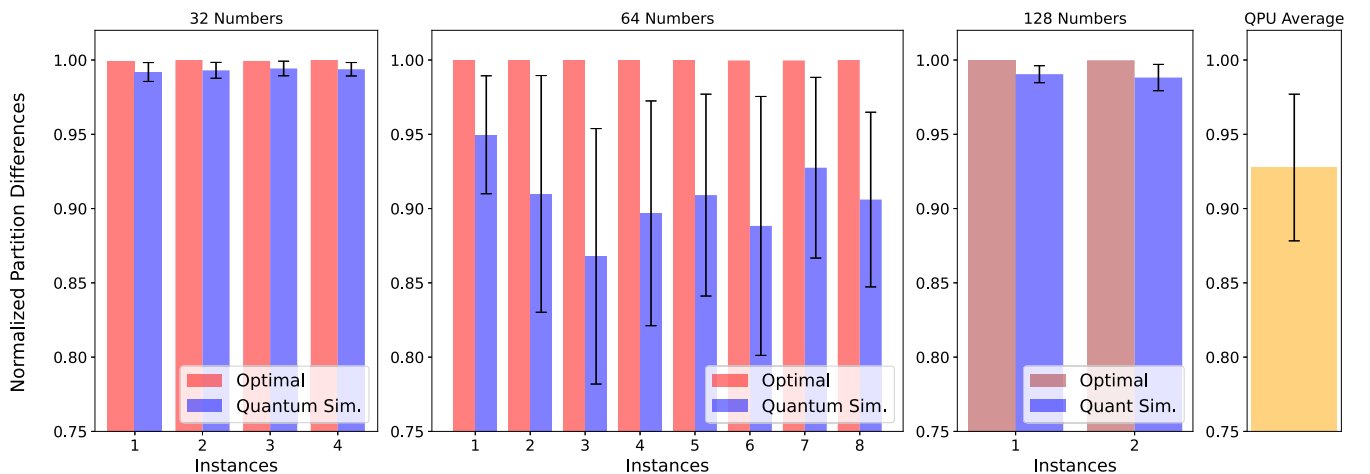


FIG. 4. The difference between partition sets for 32, 64, and 128 numbers. Each instance was run on the quantum simulator with GA, 100 times for 32 numbers, 10 times for 64 numbers, and 4 times for 128 numbers. The QPU data are the aggregate of single runs for every 32- and 64-number instance.

TABLE VI. *Maximum Clique* results using quantum simulator with GA.

Instance	No. of qubits	No. of runs	Best solution	Worst solution	Ave. solution	Opt. solution
Size = 31, density = 0.3	6	50	<b>4</b>	3	3.38	4
Size = 31, density = 0.4	6	50	<b>5</b>	3	3.92	5
Size = 31, density = 0.5	6	50	<b>6</b>	3	4.46	6
Size = 31, density = 0.6	6	50	7	4	5.34	8
Size = 31, density = 0.7	6	50	8	5	6.26	9
Size = 63, density = 0.5	7	10	<b>8</b>	6	6.5	8
Size = 63, density = 0.6	7	10	8	6	7.2	10
Size = 63, density = 0.7	7	10	11	8	9.3	12

A part of the methodology presented in the paper is protected by a provisional patent claim “Method for optimizing a functioning relative to a set of elements and associated computer program product” submission number EP21306155.9 submitted on August 8, 2021.

#### APPENDIX A: CALCULATING THE EXPECTATION VALUE OF AN OBSERVABLE

Given a Hamiltonian matrix  $H$ , we first need to convert into a sum of tensor products of Pauli strings.

Let  $H$  be a  $n \times n$  Hamiltonian matrix and  $S = \{I, X, Y, Z\}^n = \{S_1, S_2, S_3, S_4\}^n$  be the set of Pauli matrices. We can consider  $n$  to be a power of two without any loss of generality. If the size of the Hamiltonian matrix is  $n'$  which is not a power of two, we can easily convert it to a size of  $n = 2^{\log_2(n')}$ , which is a power of two. The extra space in the matrix is filled with zeros.

This Hamiltonian can now be represented on  $N = \log_2(n)$  qubits. Consider the set  $J = \{S_{i_1} \otimes S_{i_2} \cdots \otimes S_{i_N} | i_1, i_2, \dots, i_N \in \{0, 1, 2, 3\}\}$  which consists of all tensor product combinations of the Pauli matrices.

Then the Hamiltonian can be decomposed as

$$H = \sum_{i=1}^{4^N} c_i J_i, \quad (\text{A1})$$

where the coefficients are

$$c_i = \frac{1}{n} \text{Tr}(J_i \cdot H). \quad (\text{A2})$$

The Hamiltonian therefore becomes

$$H = \frac{1}{n} \sum_{i=1}^{4^N} \text{Tr}(J_i \cdot H) J_i. \quad (\text{A3})$$

The expectation value becomes a sum of the expectation values of all the terms:

$$\langle \Psi | H | \Psi \rangle = \frac{1}{n} \sum_{i=1}^{4^N} \text{Tr}(J_i \cdot H) \langle \Psi | J_i | \Psi \rangle. \quad (\text{A4})$$

#### APPENDIX B: INTEGER LINEAR PROGRAM FOR MAXIMUM CUT PROBLEM

Given a graph  $G(V, E)$  such that  $n = |V|$ , and  $A_{ij}$  being the corresponding adjacency matrix terms, we have

**Objective:**  $\max \sum_{1 \leq i \leq j \leq n} x_{ij} A_{ij}$

**Constraints:**

- (1)  $x_{ij} \leq x_{ik} + x_{kj}$ ;
- (2)  $x_{ij} + x_{ik} + x_{kj} \leq 2$ ;
- (3)  $x_{ij} \in \{0, 1\}$ .

#### APPENDIX C: INTEGER QUADRATIC PROGRAM FOR MINIMUM PARTITION PROBLEM

Given a set  $S = \{w : w \in \mathbb{Z}^+\}$  and  $A \subseteq S$ , we have

**Variables:**  $x_i = \begin{cases} 1 & \text{if } w_i \in A \\ 0 & \text{if } w_i \notin A. \end{cases}$

**Objective:**  $\min(\sum_{i=1}^n w_i x_i - \sum_{i=1}^n w_i (1 - x_i))^2$ .

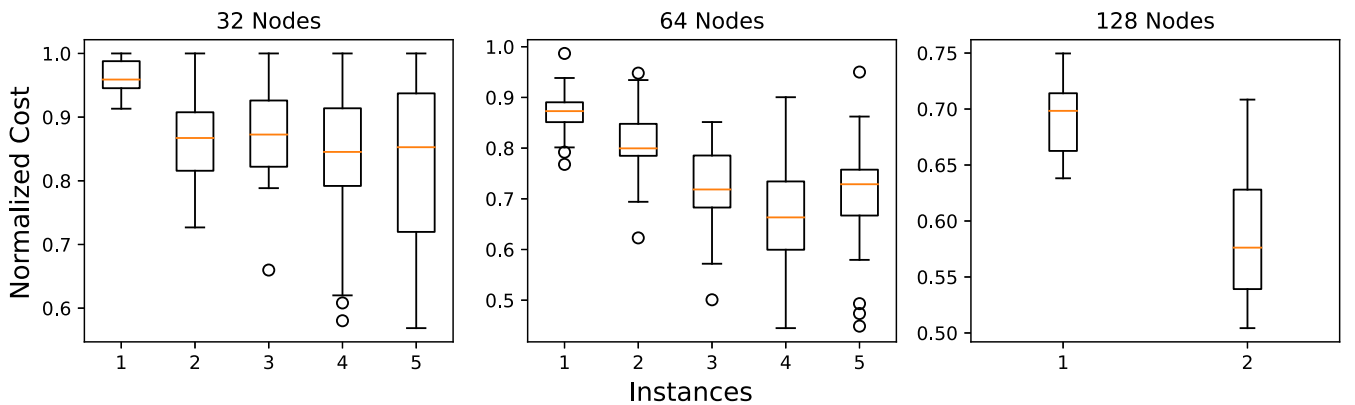


FIG. 5. *Maximum Weighted Independent Set* problem for 32-, 64-, and 128-node graphs using a quantum simulator. Each instance was run on a quantum simulator with GA 50 times for graph sizes of 32 and 64 and 10 times for a graph size of 128.

TABLE VII. Parameters of the genetic algorithm used in the *Maximum Weighted Independent Set* experiments.

Genetic algorithm parameter	Instance size		
	32	64	128
max_num_iteration	50	100	200
population_size		20	
mutation_probability		0.1	
elit_ratio		0.05	
crossover_probability		0.5	
parents_portion		0.3	
crossover_type		Uniform	
max_iteration_without_improv		None	

#### APPENDIX D: A COMPARATIVE STUDY OF TIME TAKEN BY THE SIMULATOR AND THE QPU

In Table IX and Fig. 6, the time taken to run the algorithm for different *Maximum Cut* instance sizes is compared. While the quantum computer still takes a significant amount of time to solve the problem, the time taken does not increase exponentially as in the case of the simulator. As we move towards larger instances, we reach a point where it is quicker to run a problem on a QPU than using a simulator.

Note that the QPU time here does not take into account the queue time or waiting time for the QPU runs. The real-world time was several hours or even several days for the largest run instance. This prevented us from running larger instances on the quantum computer.

The utility of comparing the time consumption of the simulator and QPU is to underline the reason why we could not go further than 256 nodes using the simulator. It is to show that, even though it would take a very long time to simulate the algorithm at 512 or more variables, the increase in cost should not be as much to run it on a quantum system. Therefore, provided we have the required level of access to a quantum computer, experiments of over 256 nodes could potentially be run on a quantum computer.

#### APPENDIX E: EFFICIENT DECOMPOSITION OF $d$ -SPARSE HAMILTONIANS

An efficient way to decompose sparse Hermitian matrices to sums of unitaries is shown in Ref. [46]. Let our  $d$ -sparse Hamiltonian  $H$  be of size  $n \times n$  and let  $N = \log_2 n$ .

TABLE VIII. *Maximum Weighted Independent Set* results demonstrating the relationship between GA iterations and performance. The instance 4 of size 64 is used for this table. The performances are an average over 10 runs.

Size	GA iterations	Solution as percent of optimum	Percent of feasible solutions
64	50	54.8	60
64	100	66.6	96
64	200	77.8	100

TABLE IX. Data for time taken for various instance sizes in the QPU and in the quantum simulator.

$N$	QPU (minutes)	Quantum simulator (minutes)
32	1.7	3
64	9	52
128	45	222
256	112	3202

The quantum operators  $U_x$ ,  $U_y$ , and  $S$  can then be defined, which act as follows:

$$U_x|j\rangle|0\rangle|0\rangle|0\rangle = \frac{1}{\sqrt{d}} \sum_{i \in F_j} |j\rangle|i\rangle|0\rangle(\sqrt{H_{ji}^*}|0\rangle + \sqrt{1 - |H_{ji}|}|1\rangle), \quad (\text{E1})$$

$$U_y|0\rangle|j'\rangle|0\rangle|0\rangle = \frac{1}{\sqrt{d}} \sum_{i' \in F_{j'}} |i'\rangle|j'\rangle(\sqrt{H_{i'j'}^*}|0\rangle + \sqrt{1 - |H_{i'j'}|}|1\rangle)|0\rangle, \quad (\text{E2})$$

$$S|j\rangle|j'\rangle|\cdot\rangle|\cdot\rangle = |j'\rangle|j\rangle|\cdot\rangle|\cdot\rangle. \quad (\text{E3})$$

In all the above equations there are four quantum registers. The first two registers contain  $N$  qubits each while the last two contain one qubit each.  $H_{ij}$  refers to an element of the Hamiltonian matrix with indices  $i$  and  $j$ .  $F_j$  is the set of indices  $i$  for which  $H_{ij} \neq 0$ .

Let  $|\tilde{0}\rangle = |0\rangle|0\rangle|0\rangle$  represent the last three quantum registers. Then, using Eqs. (E1)–(E3), it can be shown that

$$H \otimes |\tilde{0}\rangle\langle\tilde{0}| = d|\tilde{0}\rangle\langle\tilde{0}|U_x^\dagger U_y S|\tilde{0}\rangle\langle\tilde{0}| \quad (\text{E4})$$

$$= \frac{d}{4}(\mathbb{I} - e^{i\pi|\tilde{0}\rangle\langle\tilde{0}|})U_x^\dagger U_y S(\mathbb{I} - e^{i\pi|\tilde{0}\rangle\langle\tilde{0}|}). \quad (\text{E5})$$

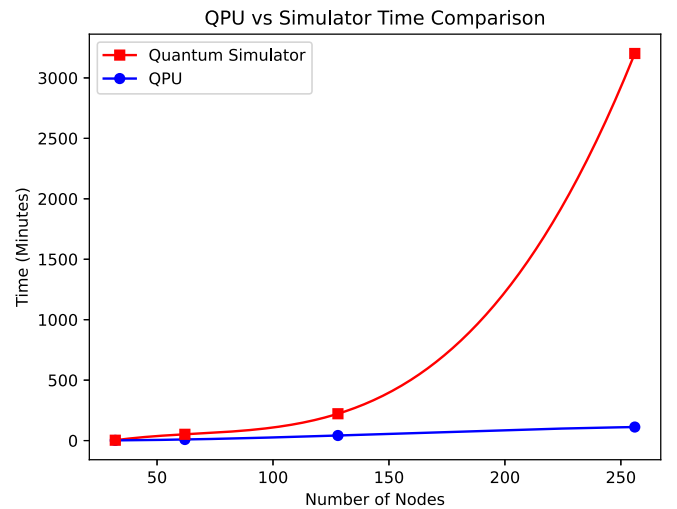


FIG. 6. Plot demonstrating the time taken by the quantum simulator versus the time taken to solve the same instance on real hardware.

Equation (E5) is the decomposition of the Hamiltonian  $H$  into four unitaries. To generate the operations of  $U_x$  and  $U_y$ , we need access to the oracle  $\mathcal{P}_A$  as shown in Sec. 1.1 of Ref. [47]. We can generate  $U_x$  and  $U_y$  using three queries to this oracle as shown in Ref. [46]. Since both  $U_x$  and  $U_y$  require three queries, each of the four unitaries in (E5) require a total of six queries to  $\mathcal{P}_A$ . Besides the queries, the gate complexity of

$U_x$  and  $U_y$  depend on how accurately and precisely the matrix elements of  $H$  are defined. The gate complexity of  $e^{i\pi|\vec{0}\rangle\langle\vec{0}|}$  is  $O(\log_2 n)$ .

In addition to this, an expectation value of a sum of  $m$  summands can be obtained in less than  $m$  queries to the QPU, again for sparse operators. More details can be seen in Ref. [48].

- [1] C. J. Hillar and L.-H. Lim, Most tensor problems are NP-hard, *J. Assoc. Comput. Mach.* **60**, 1 (2013).
- [2] G. J. Woeginger, Exact algorithms for NP-hard problems: A survey, in *Combinatorial Optimization—Eureka, You Shrink!* (Springer, Berlin, 2003), pp. 185–207.
- [3] A. Sánchez-Arroyo, Determining the total colouring number is NP-hard, *Discrete Math.* **78**, 315 (1989).
- [4] P. N. Klein and N. E. Young, Approximation algorithms for NP-hard optimization problems, in *Algorithms and Theory of Computation Handbook: General Concepts and Techniques* (Chapman Hall/CRC, 2010), Chap 34.
- [5] D. S. Hochba, Approximation algorithms for NP-hard problems, *SIGACT News* **28**, 40 (1997).
- [6] T. N. Bui and C. Jones, Finding good approximate vertex and edge partitions is NP-hard, *Inf. Process. Lett.* **42**, 153 (1992).
- [7] J. M. Hendrickx and A. Olshevsky, Matrix p-norms are NP-hard to approximate if  $p \neq 1, 2, \infty$ , *SIAM J. Matrix Anal. Appl.* **31**, 2802 (2010).
- [8] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm, [arXiv:1411.4028](https://arxiv.org/abs/1411.4028).
- [9] S. Hadfield, Z. Wang, B. O’Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, From the quantum approximate optimization algorithm to a quantum alternating operator ansatz, *Algorithms* **12**, 34 (2019).
- [10] A. Callison and N. Chancellor, Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond, *Phys. Rev. A* **106**, 010101 (2022).
- [11] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’Brien, A variational eigenvalue solver on a photonic quantum processor, *Nat. Commun.* **5**, 4213 (2014).
- [12] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn *et al.*, Quantum optimization using variational algorithms on near-term quantum devices, *Quantum Sci. Technol.* **3**, 030503 (2018).
- [13] J. Stokes, J. Izaac, N. Killoran, and G. Carleo, Quantum natural gradient, *Quantum* **4**, 269 (2020).
- [14] K. M. Nakanishi, K. Fujii, and S. Todo, Sequential minimal optimization for quantum-classical hybrid algorithms, *Phys. Rev. Res.* **2**, 043158 (2020).
- [15] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio *et al.*, Variational quantum algorithms, *Nat. Rev. Phys.* **3**, 625 (2021).
- [16] L. Bittel and M. Kliesch, Training variational quantum algorithms is NP-hard, *Phys. Rev. Lett.* **127**, 120502 (2021).
- [17] M. Lubasch, J. Joo, P. Moinier, M. Kiffner, and D. Jaksch, Variational quantum algorithms for nonlinear problems, *Phys. Rev. A* **101**, 010301(R) (2020).
- [18] N. Mariella and A. Simonetto, A quantum algorithm for the subgraph isomorphism problem, *ACM Trans. Quantum Comput.* **4**, 1 (2023).
- [19] P. K. Barkoutsos, G. Nannicini, A. Robert, I. Tavernelli, and S. Woerner, Improving variational quantum optimization using CVar, *Quantum* **4**, 256 (2020).
- [20] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth *et al.*, The variational quantum eigensolver: A review of methods and best practices, *Phys. Rep.* **986**, 1 (2022).
- [21] F. G. Fuchs, H. Ø. Kolden, N. H. Aase, and G. Sartor, Efficient encoding of the weighted MAX  $k$ -cut on a quantum computer using QAOA, *SN Comput. Sci.* **2**, 89 (2021).
- [22] J. Larkin, M. Jonsson, D. Justice, and G. G. Guerreschi, Evaluation of QAOA based on the approximation ratio of individual samples, *Quantum Sci. Technol.* **7**, 045014 (2022).
- [23] R. Herrman, L. Treffert, J. Ostrowski, P. C. Lotshaw, T. S. Humble, and G. Siopsis, Globally optimizing QAOA circuit depth for constrained optimization problems, *Algorithms* **14**, 294 (2021).
- [24] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices, *Phys. Rev. X* **10**, 021067 (2020).
- [25] G. G. Guerreschi and A. Y. Matsuura, QAOA for max-cut requires hundreds of qubits for quantum speed-up, *Sci. Rep.* **9**, 6903 (2019).
- [26] M. J. Rančić, Noisy intermediate-scale quantum computing algorithm for solving an  $n$ -vertex maxcut problem with  $\log(n)$  qubits, *Phys. Rev. Res.* **5**, L012021 (2023).
- [27] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, Cambridge, 2011).
- [28] J. Preskill, Quantum computing in the NISQ era and beyond, *Quantum* **2**, 79 (2018).
- [29] J. Preskill, Quantum Computing 40 Years Later, in *Feynman Lectures on Quantum Computation* (CRC Press, Boca Raton, 2023).
- [30] R. P. Feynman, Simulating physics with computers, *Int. J. Theor. Phys.* **21**, 467 (1982).
- [31] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, Perspectives of quantum annealing: Methods and implementations, *Rep. Prog. Phys.* **83**, 054401 (2020).
- [32] A. Stern and N. H. Lindner, Topological quantum computation—from basic concepts to first experiments, *Science* **339**, 1179 (2013).
- [33] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest, Measurement-based quantum computation, *Nat. Phys.* **5**, 19 (2009).



- [34] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Rev.* **41**, 303 (1999).
- [35] L. K. Grover, A fast quantum mechanical algorithm for database search, in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY, 1996), p. 212.
- [36] I. Kerenidis and A. Prakash, A quantum interior point method for LPs and SDPs, *ACM Trans. Quantum Comput.* **1**, 1 (2020).
- [37] A. Pothen, H. D. Simon, and K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs, *SIAM J. Matrix Anal. Appl.* **11**, (1990).
- [38] M. P. Harrigan, K. J. Sung, M. Neeley, K. J. Satzinger, F. Arute, K. Arya, J. Atalaya, J. C. Bardin, R. Barends, S. Boixo *et al.*, Quantum approximate optimization of non-planar graph problems on a planar superconducting processor, *Nat. Phys.* **17**, 332 (2021).
- [39] D. Winderl, N. Franco, and J. M. Lorenz, A comparative study on solving optimization problems with exponentially fewer qubits, [arXiv:2210.11823](https://arxiv.org/abs/2210.11823).
- [40] J. C. Bezdek and R. J. Hathaway, Some notes on alternating optimization, in *AFSS International Conference on Fuzzy Systems* (Springer, Berlin, 2002), pp. 288–300.
- [41] R. M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations*, edited by R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, The IBM Research Symposia Series (Springer, Boston, 1972).
- [42] J. A. Ruiz-Vanoye *et al.*, Survey of polynomial transformations between NP-complete problems, *J. Comput. Appl. Math.* **235**, 4851 (2011).
- [43] F. Glover, G. Kochenberger, and Y. Du, Quantum bridge analytics I: A tutorial on formulating and using QUBO models, *Ann. Oper. Res.* **17**, 335 (2019).
- [44] W. F. de la Vega and C. Kenyon-Mathieu, Linear programming relaxations of maxcut, in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms SODA '07* (ACM Press, New York, NY, 2007), pp. 53–61.
- [45] M. X. Goemans and D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *J. Assoc. Comput. Mach.* **42**, 1115 (1995).
- [46] C. Bravo-Prieto, R. LaRose, M. Cerezo, Y. Subasi, L. Cincio, and P. J. Coles, Variational quantum linear solver, *Quantum* **7**, 1188 (2023).
- [47] A. M. Childs, R. Kothari, and R. D. Somma, Quantum algorithm for systems of linear equations with exponentially improved dependence on precision, *SIAM J. Comput.* **46**, 1920 (2017).
- [48] D. W. Berry, G. Ahokas, R. Cleve, and B. C. Sanders, Efficient quantum algorithms for simulating sparse Hamiltonians, *Commun. Math. Phys.* **270**, 359 (2007).