

# 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

(Dated: January 15, 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 QAOA 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. In order 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.

## 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 section II B, we describe in detail the logarithmic encoding of the MAXIMUM CUT problem on a quantum computer. In section 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 section II D, we show how any Quadratic Unconstrained Binary Optimization Problem (QUBO) problem can be treated using the logarithmic encoding. In section 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/objects, as well as quantum hardware (QPU) results for problem sizes up to 256.

\* yagnik.chatterjee@totalenergies.com

† eric.bourreau@lirmm.fr

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

## 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 2 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 2 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 will require  $2^N$  coefficients. If there is no entanglement then each qubit can be expressed separately. From equation (1) it can be seen that we need 2 coefficients to encode a single qubit and therefore we would need only  $2N$  coefficients to represent  $N$  non-entangled qubits. Entanglement is therefore of paramount importance in quantum computing as 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]. They 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 2 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, theoretical advantages of quantum algorithms compared to 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.

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 equation (6) is

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

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

As described in Figure 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  $|\Psi(\theta_1 \dots \theta_n)\rangle$ . In order 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.

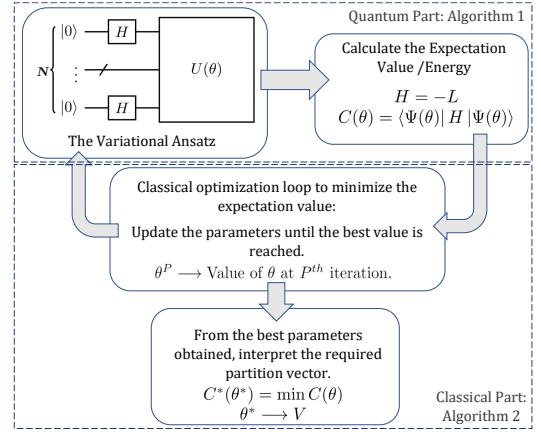


Figure 1. Diagrammatic representation of the hybrid quantum-classical algorithm.

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 required as follows:

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

When the number of nodes are not an exact power of 2, 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:

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

Therefore the final ansatz is:

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

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 equation 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 Genetic Algorithm (as detailed in Algorithm 2), we then obtain

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

The final parameters obtained  $\theta^*$  gives the bi-partition vector, using equation (9).

---

**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  $H \leftarrow \frac{1}{n} \sum_{i=1}^{4^N} \text{Tr}(J_i \cdot L^*) J_i$  where  $J = \{\prod_{k=1}^N S^{\otimes k}\}$
- 5  $\theta \leftarrow$  List of  $|V|$  parameters
- 6 **Function**  $\text{EvalCost}(\theta)$ :
- 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

---

**Algorithm 2:** Log Encoding of MAXIMUM CUT:  
Minimizing the Objective Function

---

**Input:**  $\text{EvalCost}(\theta)$

- 1 **Function**  $\text{Optimizer}(\text{EvalCost}(\theta), \theta^{initial})$ :
- 2   | **repeat**
- 3   |   |  $\theta^p \leftarrow \theta$  at  $p^{th}$  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

---

## 2. Advantages and disadvantages of the 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 7 qubits. A problem with over a million nodes can be represented with just 20 qubits, something that is quite unthinkable using contemporary

algorithms. 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. While the QAOA depends on the availability of a large number of qubits for the algorithm to work on any interesting problem, the above algorithm only depends on an increased accuracy in QCs of current size. The number of CNOT gates required for the QAOA ansatz is  $p|E|$ , where  $p$  is the depth of the algorithm (for all practical purposes,  $p \gg 1$  [38]) and  $|E|$  is the number of edges in the graph. In our algorithm the number of CNOTs is equal to  $|V| - 1$ ,  $|V|$  being the number of vertices. Generally, and especially at higher densities, it is easy to see that  $p|E| \gg |V|$ . This means that our circuit turns out to be much shallower than that of QAOA.

In our study, 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 [26]. However, this method is beyond the scope of the current work. Readers should also refer to followup 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 optimisation 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.

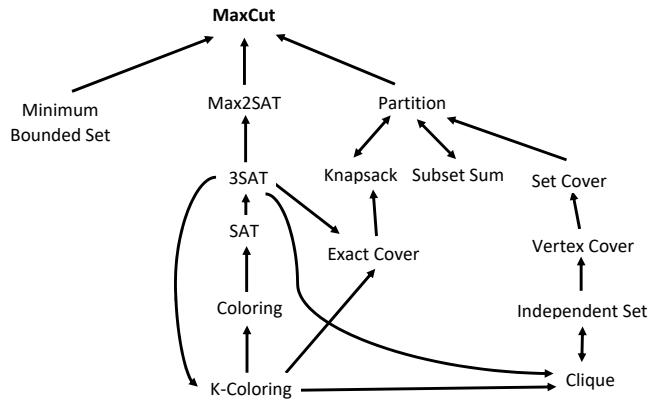


Figure 2. Graph of MAXIMUM CUT transformation family for NP-complete problems.

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 optimisation one. However, our algorithm is inherently an optimisation 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 optimisation 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 bipartition 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 2 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  $\{G'(V', E') : V' \subseteq V, E' \subseteq E, |E'| = \frac{|V'|(|V'| - 1)}{2}\}$ .

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  $x_i$ . Also an auxiliary variable  $z$ . We therefore have  $|V| + 1$  variables.
2. For every variable add the following 2 clauses:  $(x_i + z)$  and  $(x_i + \bar{z})$ . Let us refer to these clauses as Type A clauses.
3. Add the following clauses:  $(\bar{x}_i + \bar{x}_j) \vee (i, j) \notin E$ . We will 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 2 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.

Firstly, 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  $x_i^2 = x_i \forall x \in \{0, 1\}$ .

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. This is 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  $x = \frac{1-z}{2}$ . 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  $\frac{1-z_i}{2}$  has been used instead of  $\left(\frac{1-z_i}{2}\right)^2$  since  $x_i^2 = x_i$ .

In the search for optimal values of parameters  $z$  we can eliminate the constant terms in  $P$  as 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 equation (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 shall call the *spin-QUBO* (sQUBO). This is a matrix of size  $2n \times 2n$ . It will require  $\lceil \log_2 2n \rceil = (1 + \lceil \log_2 n \rceil)$  qubits, or, in other words, 1 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.

---

**Algorithm 3:** Log Encoding of a QUBO problem: Building the Objective Function

---

**Input:** QUBO Matrix

1 Convert QUBO to sQUBO

2  $Q \leftarrow$  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  $H \leftarrow \frac{1}{n} \sum_{i=1}^{2^N} \text{Tr}(J_i \cdot Q^*) J_i$  where  $J = \{\prod_{k=1}^N S^{\otimes k}\}$

6  $\theta \leftarrow$  List of  $n$  parameters

7 **Function**  $\text{EvalCost}(\theta)$ :

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

---

1. MAXIMUM WEIGHTED INDEPENDENT SET using QUBO

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

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

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 3a) shows the performance of the algorithm versus the optimal solution obtained using an Integer Linear Program (see Appendix B). Two different classical optimizers have been used for the runs on the Quantum Simulator. We 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 is 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 Figure 3b), 10 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 was collected for 50 runs, using 2 different types of classical optimizers, COBYLA and 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 is then used to normalize the simulator data. Hence, the data is in the form of percentage of optimal value. In most instances, the genetic algorithm (GA) performed better than COBYLA. This, however, might be down 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. We can see that the increase in the number edges does not heavily impact the accuracy of the algorithm. This is an important factor and is useful for the sections to follow.

In order 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 is used for all obtained data. The data was normalized by using the ILP algorithm as mentioned before. The model solved the problem upto a specified integrality gap of 4%. The data represented in the table is given as a percent of the solution obtained from ILP. We can see that for all cases, the results are nearly or over 90% of the optimal cut. It is seen again that increase in graph density does not affect performance whatsoever. Furthermore, the execution time of ILP increases rapidly with graph density whereas it remains more or less the same for both the simulator and the QPU. For example in the graph with density 0.45, the ILP takes 1375.2 seconds whereas the quantum simulator and QPU take 272 seconds and 518 seconds respectively.

For 128 and 256 nodes (Table II and III), the Goemans Williamson (GW) method [45] is used for benchmarking. This is because the ILP took longer than 2 days without converging (Gurobi optimizer) on a PC. The GW range is based on 50 runs. Table II shows results using a quantum simulator while Table III 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.

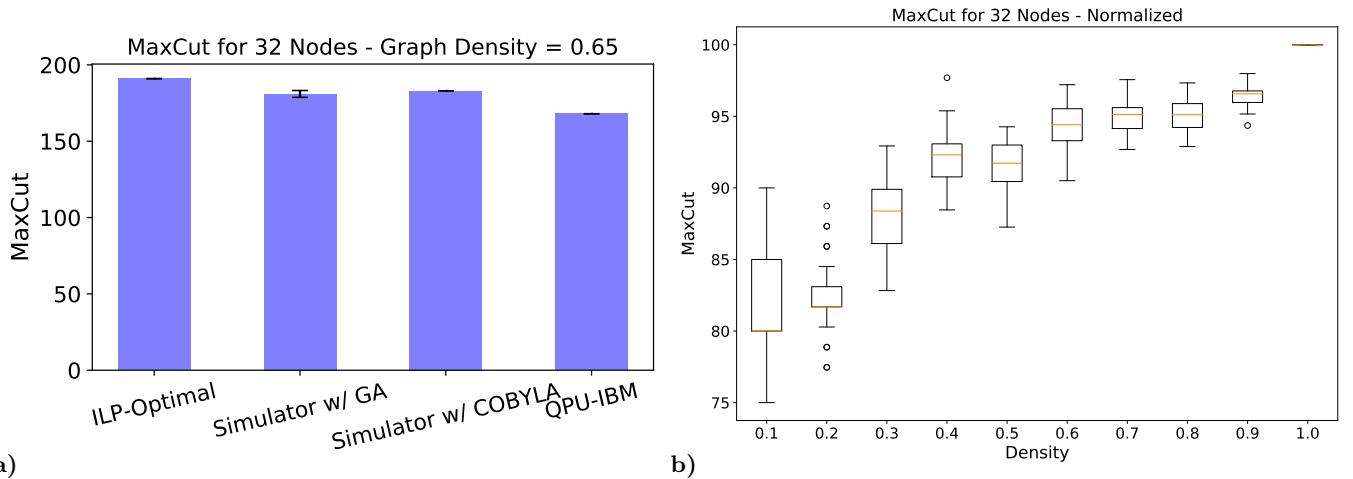


Figure 3. **a)** Performance of the algorithm on a 32-Node graph instance. The QPU result is based on a single run while the simulator results are based on 50 runs. **b)** The MAXIMUM CUT of 32-node graphs of varying densities. Optimizer used is Genetic Algorithm. Data is based on 50 runs for each instance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

Graph Density	ILP			Quantum Simulator		QPU	
	Solution	Time(s)	Gap(%)	Solution	Time(s)	Solution	Time(s)
0.30	383	7.6	3.92	343.9	280	282	383
0.35	443	69.7	3.84	400.8	272	365	439
0.40	497	1077.7	3.82	454.3	270	380	393
0.45	553	1375.2	3.98	512.8	272	446	518

Table I. 64-Node MAXIMUM CUT results

Graph Density	128 Nodes			256 Nodes		
	GW Range	Solution	% Difference	GW Range	Solution	% Difference
0.3	1376 - 1431	1270	88.7 - 92.3	5408 - 5587	5066	90.7 - 93.7
0.4	1796 - 1864	1691	90.7 - 94.1	7087 - 7232	6736	93.1 - 95.0
0.5	2186 - 2271	2103	92.6 - 96.2	8701 - 8880	8367	94.2 - 96.2
0.6	2618 - 2679	2501	93.3 - 95.5	10356 - 10504	9967	94.9 - 96.2

Table II. 128 and 256-Node MAXIMUM CUT results using a quantum simulator.

Instance	Solution	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

Table III. 128 and 256-Node MAXIMUM CUT results using QPU with GA.

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

As described in Section 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 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 normalised difference is  $p_{norm} = \frac{50N - p}{50N}$ . All our numbers are random integers between 1 and 100, hence 50.5 on an average. For simplicity we use 50 in  $p_{norm}$ .

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

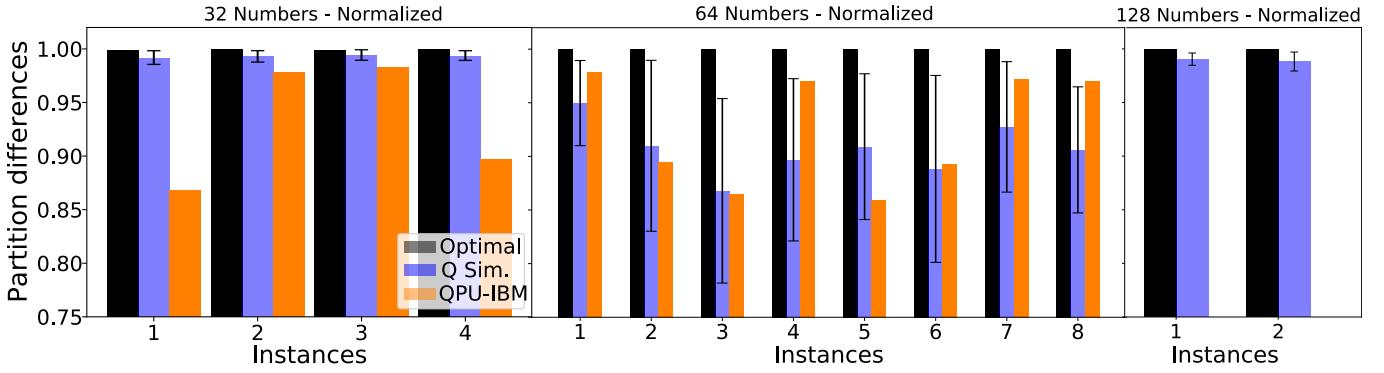


Figure 4. The difference between partition sets for 32, 64 and 128 Numbers. Each instance was run on the quantum simulator 4 times. Each instance was run on the quantum simulator with GA, 100 times for 32 numbers, 10 times for 64 numbers and 4 times for 128 numbers. The QPU data are based on a single run.

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 IV shows results for various instances run on a quantum simulator. 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.

### D. Maximum Weighted Independent Set using QUBO method

In this section, results of the MAXIMUM WEIGHTED INDEPENDENT SET problem solved using the QUBO method (section IID 1) is presented.

For each figure the performance of the algorithm is shown. The data is 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 is based on 50, 50 and 10 runs respectively. The data represented only takes 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 is based only on a few runs. Moreover, the performance also depends on the number of GA iterations used in the algorithm run. The number of GA iterations used for the experiments for sizes 32, 64, and 128 are 50, 100 and 200 respectively.

Table V 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 not only improves the performance but also the percentage of feasible results.

### E. A comparative study of time taken by the simulator and the QPU

In Table VI and Figure 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.

## 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 [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

Instance	No. of Qubits	No. of Runs	Best Solution	Worst Solution	Av. Solution	Opt. Solution
Size=31, Density=0.3	6	50	4	3	3.38	4
Size=31, Density=0.4	6	50	5	3	3.92	5
Size=31, Density=0.5	6	50	6	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	8	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

Table IV. MAXIMUM CLIQUE results using quantum simulator with GA.

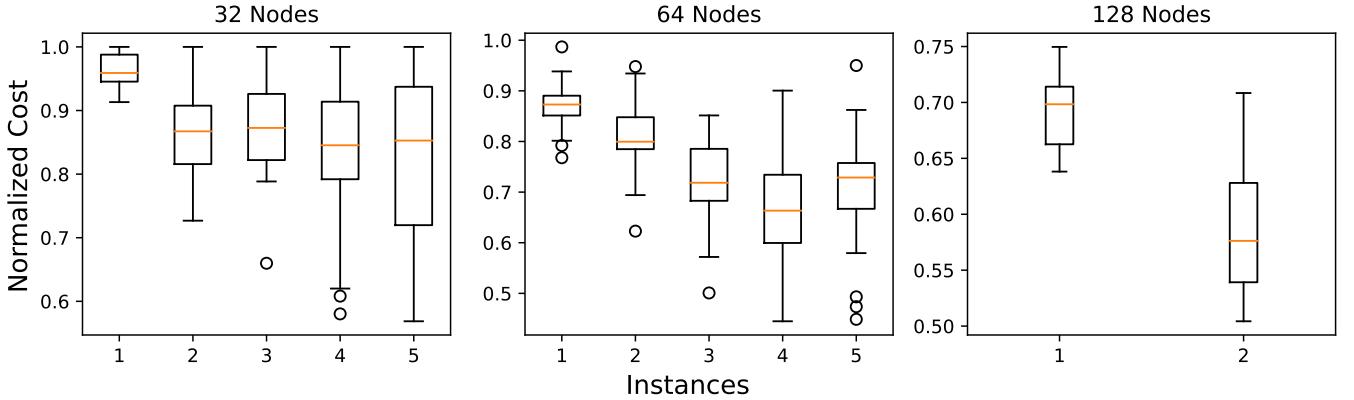


Figure 5. MAXIMUM WEIGHTED INDEPENDENT SET problem for 32, 64 and 128-node graphs using a quantum simulator. Each instance was run on a quantum simulator with GA 50 times for graph sizes of 32 and 64 and 10 times for graph size of 128.

Size	GA iterations	Solution as % of Optimum	% of Feasible Solutions
64	50	54.8	60
64	100	66.6	96
64	200	77.8	100

Table V. 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.

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

Table VI. Data for time taken for various instance sizes in the QPU and in the quantum simulator

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 reduc-

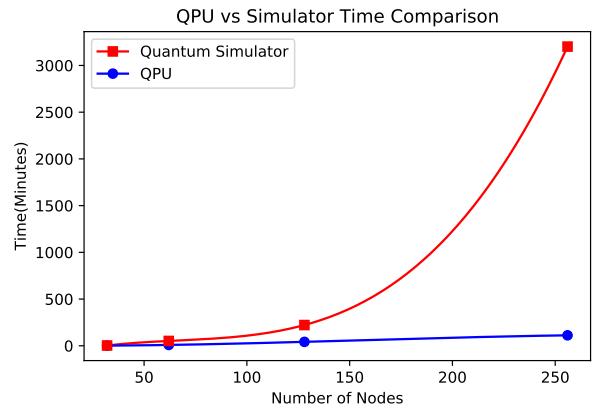


Figure 6. Plot demonstrating the time taken by the quantum simulator versus the time taken to solve the same instance on real hardware

tion as mentioned in section 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.

To our knowledge, it is the first time that graph problems of such sizes (256 MAXIMUM CUT, 64 MINIMUM PARTITION) have been executed on real universal gate-based quantum computers.

## V. DECLARATIONS

### A. Ethical Approval and Consent to participate

Not applicable.

### B. Consent for publication

Not applicable.

### C. Data availability statement

The authors will make all data available upon reasonable requests.

### D. Competing interests

A part of the methodology presented in the manuscript is protected by a provisionally patent claim "Method for optimizing a functioning relative to a set of elements and associated computer program product" submission number EP21306155.9 submitted on 26.8.2021.

### E. Funding

Y.C. and M.R. acknowledge funding from European Union's Horizon 2020 research and innovation programme, more specifically the  $\langle \text{NE} | \text{AS} | \text{QC} \rangle$  project under grant agreement No. 951821.

### Appendix A: Appendix: 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 2 without any loss of generality. If the size of the Hamiltonian matrix is  $n'$  which is not a power of 2, we can easily convert it to a size of  $n = 2^{\log_2(n')}$ , which is a power of 2. The extra space in the matrix is filled with 0's.

This Hamiltonian can now be represented on  $N = \log_2(n)$  qubits. Consider the set  $J = \{S_{i_1} \otimes S_{i_2} \dots \otimes S_{i_N} | i_1, i_2, \dots, i_N \in \{0, 1, 2, 3\}\}$  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: Appendix: 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: Appendix: 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 :**  $\min \left( \sum_{i=1}^n w_i x_i - \sum_{i=1}^n w_i (1 - x_i) \right)^2$

- 
- [1] C. J. Hillar and L.-H. Lim, Most tensor problems are np-hard, *Journal of the ACM (JACM)* **60**, 1 (2013).
- [2] G. J. Woeginger, Exact algorithms for np-hard problems: A survey, in *Combinatorial optimization—eureka, you shrink!* (Springer, 2003) pp. 185–207.
- [3] A. Sánchez-Arroyo, Determining the total colouring number is np-hard, *Discrete Mathematics* **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* (UC Riverside, 2010) pp. 34–34.
- [5] D. S. Hochba, Approximation algorithms for np-hard problems, *ACM Sigact News* **28**, 40 (1997).
- [6] T. N. Bui and C. Jones, Finding good approximate vertex and edge partitions is np-hard, *Information Processing Letters* **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 Journal on Matrix Analysis and Applications* **31**, 2802 (2010).
- [8] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm, arXiv:1411.4028 (2014).
- [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**, 10.3390/a12020034 (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, *Nature communications* **5**, 1 (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 Science and Technology* **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, *Physical Review Research* **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, *Nature Reviews Physics* **3**, 625 (2021).
- [16] L. Bittel and M. Kliesch, Training variational quantum algorithms is np-hard, *Physical Review Letters* **127**, 120502 (2021).
- [17] M. Lubasch, J. Joo, P. Moinier, M. Kiffner, and D. Jaksch, Variational quantum algorithms for nonlinear problems, *Physical Review A* **101**, 010301 (2020).
- [18] N. Mariella and A. Simonetto, A quantum algorithm for the sub-graph isomorphism problem, arXiv preprint arXiv:2111.09732 (2021).
- [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. Runger, G. H. Booth, et al., The variational quantum eigensolver: a review of methods and best practices, *Physics Reports* **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 Computer Science* **2**, 1 (2021).
- [22] J. Larkin, M. Jonsson, D. Justice, and G. G. Guerreschi, Evaluation of qaoa based on the approximation ratio of individual samples, *Quantum Science and Technology* (2022).
- [23] R. Herrman, L. Trefftz, 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, *Physical Review X* **10**, 021067 (2020).
- [25] G. G. Guerreschi and A. Y. Matsuura, Qaoa for max-cut requires hundreds of qubits for quantum speed-up, *Scientific reports* **9**, 1 (2019).
- [26] M. J. Rančić, Noisy intermediate-scale quantum computing algorithm for solving an n-vertex maxcut problem with  $\log(n)$  qubits, *Physical Review Research* **5**, L012021 (2023).
- [27] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, 2011).
- [28] J. Preskill, Quantum Computing in the NISQ era and beyond, *Quantum* **2**, 79 (2018).
- [29] J. Preskill, Quantum computing 40 years later, arXiv (2021).
- [30] R. P. Feynman, Simulating physics with computers, *International Journal of Theoretical Physics* **21**, 10.1007/BF02650179 (1982).
- [31] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, Perspectives of quantum annealing: Methods and implementations, *Reports on Progress in Physics* **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, *Nature Physics* **5**, 19 (2009).
- [34] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM review* **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* (1996) pp. 212–219.
- [36] I. Kerenidis and A. Prakash, A quantum interior point method for lps and sdps, *ACM Transactions on Quantum Computing* **1**, 1 (2020).
- [37] A. Pothen, H. D. Simon, and K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs, *SIAM journal on matrix analysis and applications* **11(430)** (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, *Nature Physics* **17**, 332 (2021).
- [39] D. Winderl, N. Franco, and J. M. Lorenz, A comparative study on solving optimization problems with exponentially fewer qubits, arXiv preprint arXiv:2210.11823 (2022).
- [40] J. C. Bezdek and R. J. Hathaway, Some notes on alternating optimization, in *AFSS international conference on fuzzy systems* (Springer, 2002) pp. 288–300.
- [41] R. M. Karp, Reducibility among combinatorial problems, In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds) Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston, MA. (1972).
- [42] J. A. Ruiz-Vanoye et al., Survey of polynomial transformations between np-complete problems, *Journal of Computational and Applied Mathematics* **235**, 4851 (2011).
- [43] F. Glover, G. Kochenberger, and Y. Du, Quantum bridge analytics i: a tutorial on formulating and using qubo models, 4OR, Springer **17(4)**, 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 (Society for Industrial and Applied Mathematics, USA, 2007) p. 53–61.
- [45] M. X. Goemans and D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *Journal of the ACM (JACM)* **42**, 1115 (1995).