QAOA concept and applications via games programming. Lecture for students.

Louis, Rawan, Vasilii, Uday, Vivek

(Dated: April 2022, 2022)

Abstract

In this lecture, we introduce the Quantum Approximate Optimization Algorithm (QAOA) and illustrate, using interactive games, how it can be used to solve real-world Combinatorial Optimization Problems. For this, we first explore what Combinatorial Optimization Problems are with the help of the classic example: The Max-Cut Problem. Then we proceed to learn about the QAOA through an important real-world application problem – The Dominating Set Problem – using an interactive exercise, and show how it offers a quantum advantage. Lastly, we also discuss the n-Queens Problem of placing n non-attacking queens on an $n \times n$ chessboard and see what role QAOA could possibly play in solving that.

I. INTRODUCTION TO QAOA

The Quantum Approximate Optimization Algorithm (QAOA) is a variational algorithm that can be used to find approximate solutions to combinatorial optimization problems. This definition is made clear once we figure out what variational algorithms and combinatorial optimization problems are. We aim to mostly use the illustrations and the games to make the students understand the concepts of QAOA rather than spending too much time on the textbook definitions.

A. VARIATIONAL ALGORITHM

Variational algorithms are generally used to find out the lowest eigenvalue and the corresponding eigenvector for a quantum system with a certain Hamiltonian. For this, we choose a trial state parameterized by certain parameters and then vary these parameters to minimize the energy. Since we never get to values lower than the lowest energy level, we obtain an upper bound on the ground state energy and hence an approximation of the ground state itself.

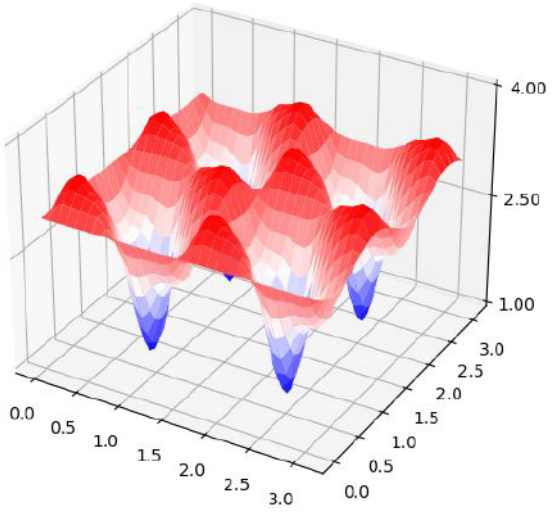

FIG. 1: A Typical Energy landscape

B. COMBINATORIAL OPTIMIZATION PROBLEM

Combinatorial optimization problems involve finding an optimal object out of a finite set of objects. One such problem corresponding to finding optimal bitstrings among a finite set of bitstrings is the Max-Cut problem which involves portioning nodes of a graph into two sets, such that the number of edges between the sets is maximum.

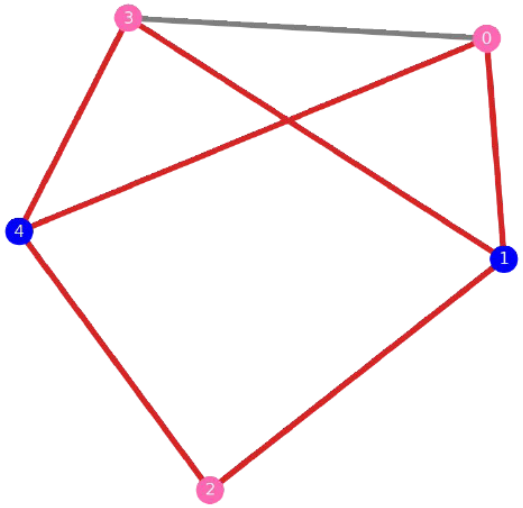

FIG. 2: The Max-Cut problem

Now we take quite some time and use the blackboard to explain the Max-Cut problem to ensure that the students get a good feel about what combinatorial optimization problems are about. We also remind them that these are not limited to this problem and that other problems exist like the Travelling Salesman Problem, the Minimum Spanning Tree problem and the knapsack problem. We insist that the students look these up after the lecture.

C. THE QAOA ALGORITHM

Now, the QAOA algorithm can be applied to solve these kind of combinatorial optimization problems. This is a variational algorithm which finds optimal parameters such that the optimized quantum state encodes the solution to the problem. This is done according to the following steps:

1. Start with an initial state which is a superposition of all the quantum states

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{b \in \{0,1\}^n} |b\rangle \tag{1}$$

2. Evolve using the problem Hamiltonian H_P by applying the unitary $U(H_P) = e^{-i\gamma H_P}$

3. Evolve using the mixing Hamiltonian H_B by applying the unitary $U(H_B) = e^{-i\beta H_B}$

4. Find optimal parameters $(\beta_{opt}, \gamma_{opt})$ using a classical optimization algorithm such that the expectation value $\langle\psi(\beta_{opt}, \gamma_{opt})| H |\psi(\beta_{opt}, \gamma_{opt})\rangle$ is minimized.

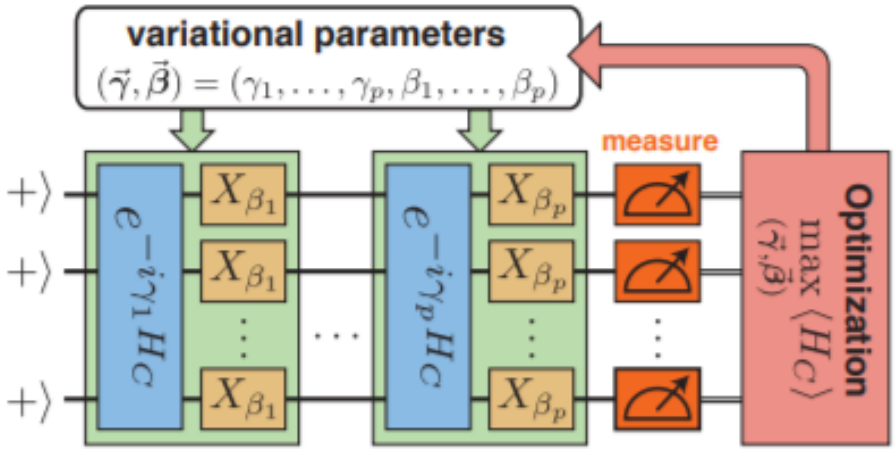

FIG. 3: Schematic of the algorithm

We would then proceed to give a demonstration of the solution of a simple Max-Cut problem using Qiskit and illustrate how the bitstrings that have the highest probability are indeed the assignments of the graphs that gives the maximum edges between the two partitions.

II. DISCUSSION AND APPLICATIONS

QAOA is used in finding solutions to combinatorial optimization problems beyond the capabilities of what classical algorithms can achieve. In order for the students to get a firmer grasp on the topic, we illustrate the concept with two examples here: One dealing with the Dominating Set Problem and the other with the slightly advanced n-Queens Problem. The n-Queens problem is discussed as an advanced concept in the next section.

A. THE DOMINATING SET PROBLEM

Let's have a look at an example of the Dominating Set Problem. Consider utilizing drones to do surveillance over a set of places (specifically nodes A, B, C, D, and E). However, the drones do not need to be able to monitor all of the areas from a single location. The drones

can monitor many regions at once by putting itself in specific areas. Now let us consider the region described in this problem as nodes in a graph, and their line of sight as the edges linking the line of sight (as shown in figure 4).

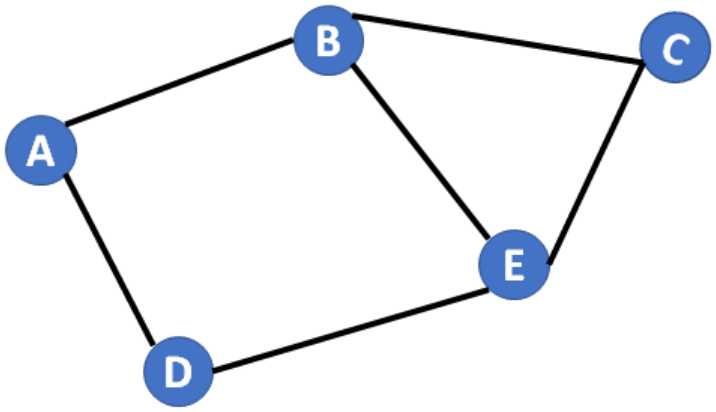

FIG. 4: Graphical interpretation of the areas to be supervised.

From the above figure 4, it is clear that it is not possible to survey all the nodes with one drone and a minimum of two drones are necessary. The complexity of the problem will increase exponentially as the numbers of nodes increases. The problem statement can be further extended to numerous examples.

We consider a very simple model as shown in Figure 5 for implementing QAOA in Qiskit. The circuit diagram and the histogram results are shown in Figure 6 and Figure 7 respectively. The key feature here is the construction of the controlled-OR gate (see Exercise 1 for further details on this). We define the cost function of the DSP as

$$c(z) = \sum_{i=0}^{n=1} D_i(z) + T_i(z)$$

where $T_k(z) = 1$ if the k^{th} node is connected to some i^{th} node where $z_i = 1$, and $T_k(z) = 0$ otherwise. Also $D_k(z) = 1$ if $z_k = 0$ and 0 if $z_k = 1$

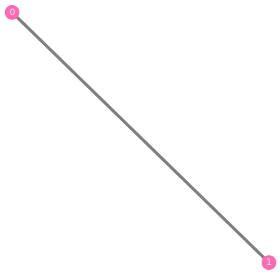

FIG. 5: Simple model for implementing QAOA

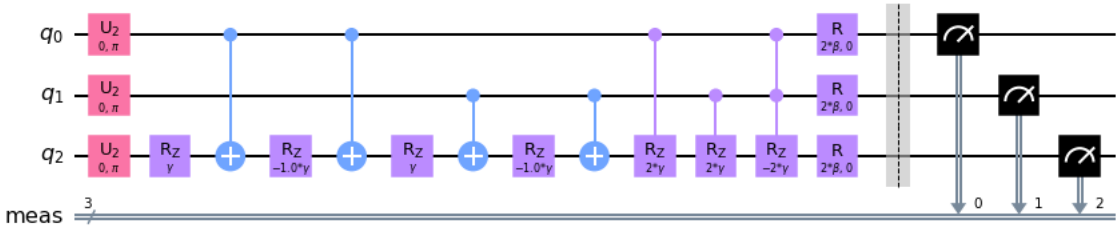

FIG. 6: Circuit diagram

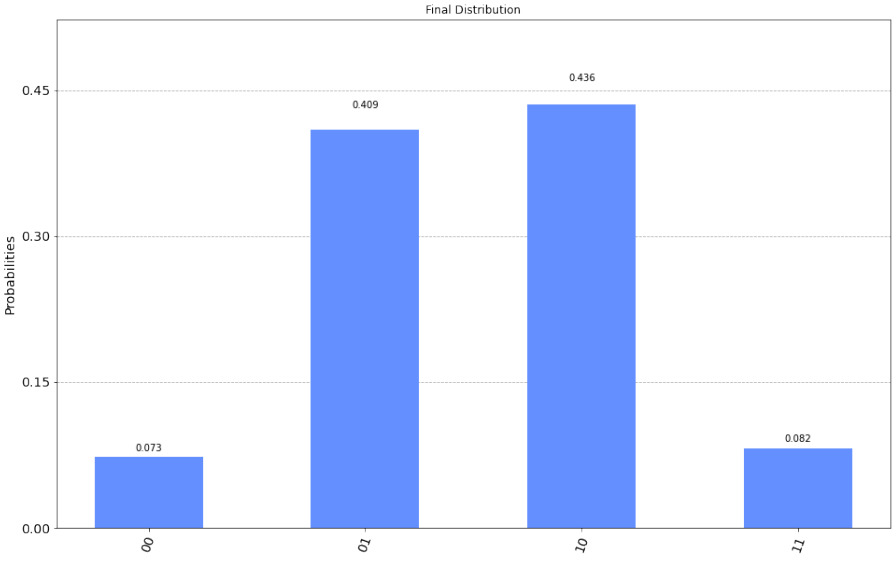

FIG. 7: Final Results

III. THE ADVANCED CONCEPTS FOR THE N-QUEENS-PROBLEM PUZZLE

In our previous demonstration, we utilize the QAOA algorithm to solve one of the NP-complete problem. For these problems encompass all decision problems in class NP that can be verified in polynomial time [1]. However, the time taken to obtain a solution for an NP complete problem is not polynomial. There are many classic examples for it such as: (1) the traveling salesman problem [2], (2) the subset sum problem [3], (3) the Hamiltonian cycle problem [4], (4) the knapsack problem[5], (4) satisfiability problem and the N-Queens Problem [6].

In this section, we will demonstrate how to utilize the quantum computation to solve this NP-complete question and compare it to the classic solution. The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other. The criteria for a particular configuration to be a solution to the N-Queens Problem are stated below. We shall collectively refer to them as the N-Queens Criteria.

- **Row Criteria:**
  The total number of queens in each row is 1.
- **Column Criteria:**
  The total number of queens in each column is 1.
- **Diagonal Criteria:**
  The total number of queens in every possible diagonal is either 0 or 1.

A. Classical Solution: Las Vegas Algorithm

A randomized algorithm is also known as Las Vegas algorithm if it always returns the correct answer, but its runtime bounds hold only in expectation.In Las Vegas algorithms, runtime is at the mercy of randomness, but the algorithm always succeeds in giving a correct answer. The all possibility of the N-queens problem is N^N, thus for a classic eight-queens puzzle there are 16,777,216 possibilities to put those queens on the chess board. Following code is the demonstration and the result of this code is shown in Fig. 5.

```
99 1
100 2 def get_queens_problem_solution(size = N, time_limit = -1):
101 3
102 4     chessboard = initialize_chessboard(size)
103 5
104 6     row = 0
105 7     abort_solution = False
106 8     while row < size \
107 9             and not abort_solution \
108 0             and (time_limit == -1 or time.time() < time_limit):
109 1
110 2         available_squares = get_available_squares(row, chessboard)
111 3
112 4         if len(available_squares) == 0:
113 5              abort_solution = True
114 6
115 7         else:
116 8             choice = get_random_square(available_squares)
117 9             chessboard[row][choice] = 1
118 0
119 1
120 2         row += 1
121 3
122 4     if abort_solution is True \
123 5             or (time_limit != -1 and time.time() > time_limit):
124 6         raise Exception('A solution could not be found.')
125 7
126 8     return chessboard
```

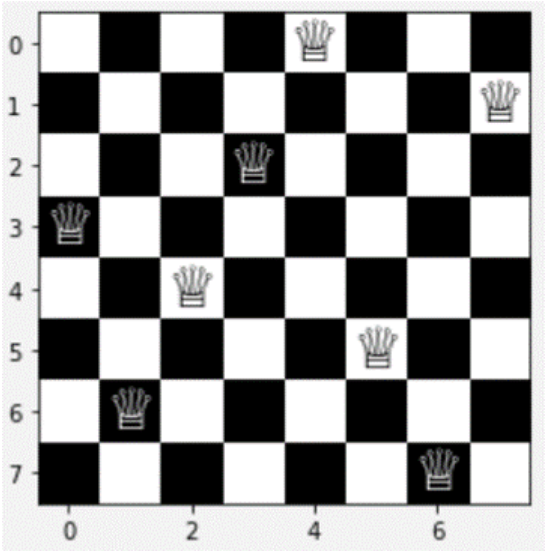

FIG. 8: The result of eight queens puzzle solving by the las vegas algorithm

The result shown in Fig. 5 provides us on of the possibility of the eight-queens puzzle. If we want to calculate every possibility those according to the algorithm (shown in Fig. 6) it will take a very long run time to solve all of the possibilities.

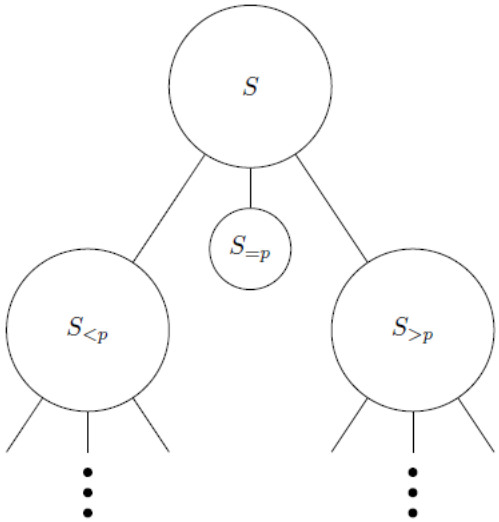

FIG. 9: The Las Vegas algorithm. Las Vegas algorithms is the randomized quick sort algorithm which picks a pivot at random, and then partitions the elements into three sets: all the elements less than the pivot, all elements equal to the pivot, and all elements greater than the pivot.

However we generate one of the result for the eight-queens puzzle, there are still other 11 possibilities which we couldn't calculate these in once shown in Fig.7.

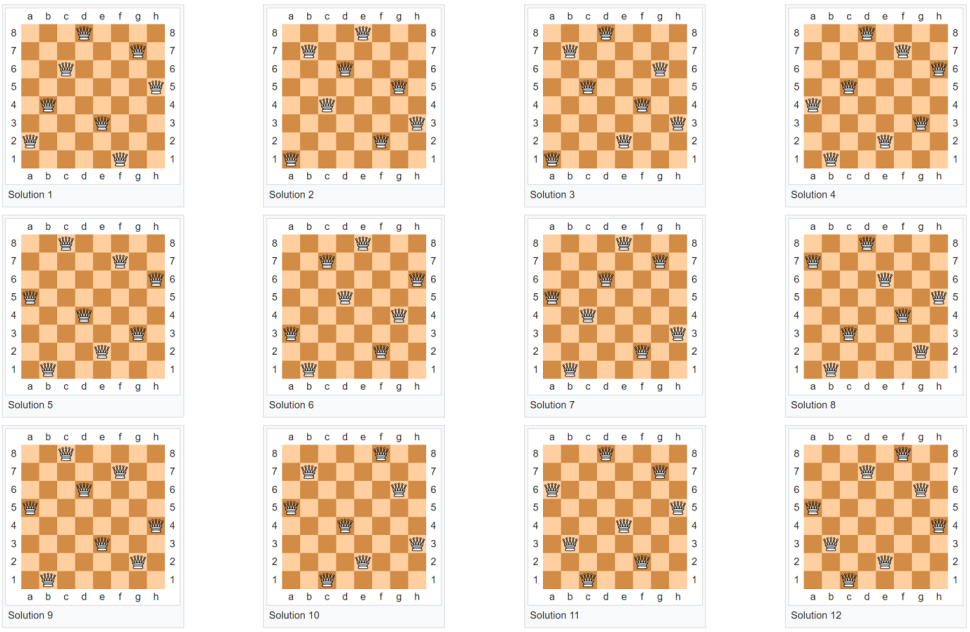

FIG. 10: The all possibility of the eight-queens puzzle result.

B. Quantum Approach to Solve the QUBO Problem

We formulate this problem by representing the N × N chessboard as an N × N matrix. The positions of queens and the vacant spaces are denoted by 1s and 0s respectively. We represent each row of the resultant matrix as an N-qubit basis state. Hence, the N-qubit quantum register for all possible row configurations having one queen can be represented as,

$$|\psi\rangle = \frac{1}{N}[|R_1\rangle + |R_2\rangle + ... + |R_n\rangle] \tag{2}$$

where R_i represents the $i - th$ row of the $N \times N$ identity matrix.

Then we based Rounak et al.'s research to generate the N-Queen Solver [7]. The detail for the algorithm is discussed in their paper. Here, we focus on two part which shown in Fig. 8.

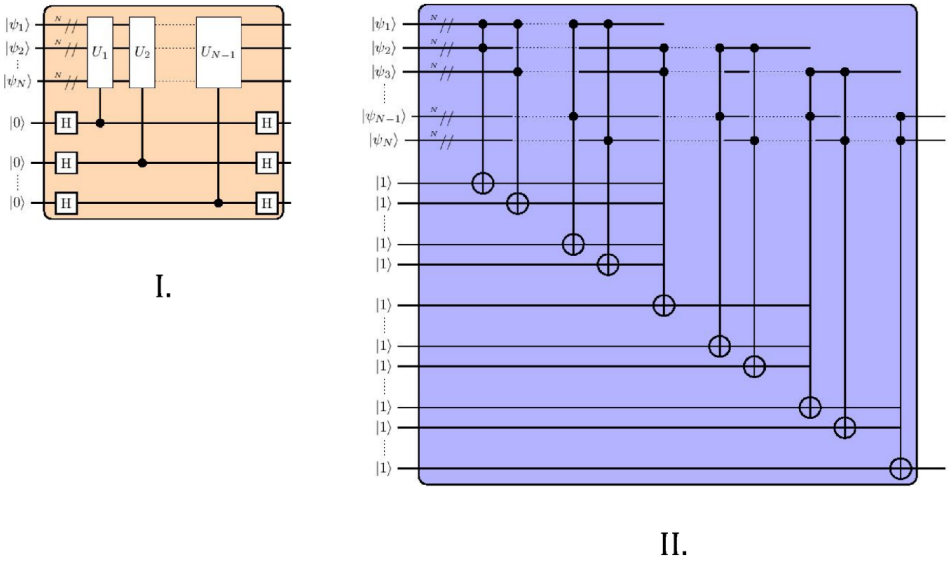

FIG. 11: The all possibility of the eight-queens puzzle result.

- **Part I.**

  The generalized quantum circuit performs the column check operation in order to distinguish the basis states satisfying the Column Criteria for reducing the search space. $N-1$ ancillary qubits along with a series of $N-1$ controlled unitary operations are used where the ancillas act as the controls. The operation Ui denotes a set of z operations on the ith qubit of all the N blocks. Thus, each controlled unitary operation in turn represents a series of N controlled-Z operations. The set of ith qubits from every block represents the ith column of the chessboard. The ith ancillary qubit flips to $|1\rangle$ if the ith column sum is odd. Since the system of qubits satisfies the Row Criteria, the total sum of all N column sums is N. As a result, all the $N-1$ ancillas flip if and only if each column sum is unity, which is only possible for basis states satisfying the Column Criteria. The circuit entangles the $N-1$ ancillas to the system. The reduced search space consists of basis states with all the N 1 entangled qubits in — 1i state. The reduced search space is the set of configurations which are row permutations of the N × N identity matrix

- **Part II.**

  The generalized quantum circuit performs the diagonal check on the basis states belonging to the reduced search space, following the column check operation. A total of

N2N2 additional qubits are required for this purpose. A series of 3-qubit operations are performed. In the circuit, the first control block of any 3-qubit operation, is the one encoding Qi, i.e. the queen in the ith row of the chessboard. The second control block of the 3-qubit operation, say — j i encodes the matrix elements in the jth row of the chessboard which are diagonal to Qi. Since for different configurations from the reduced search space, the queen Qi can be present in any one out of N columns for any given row i, the qubit belonging to block ϕ encoding Q_i could be any one out of the N qubits of the block, say $|phi\rangle$ for some x 1, 2, ..., N. Here, each 3-qubit operation represents a set of Toffoli gates, such that a 3-qubit operation with the first and second control blocks being ψ_{ii} and ψ_{ji} respectively denes a set of Toffoli gates with the first control qubit belonging to the set — qixi : x [1, N] N and the second control qubit belonging to the set — qjyi for a given — qixi, where — qjyi is the set of qubits in block $|\psi_{ji}\rangle$ encoding matrix elements (in the jth row) which are diagonal to — qixi. This leads to entanglement of the ancillas to the system. The basis states satisfying the Diagonal Criteria will not flip any auxiliary qubit. Hence, all the $fracN^2N2$ entangled qubits are in — 1i state. Overall, the result of this circuit is that only the basis states which satisfy the N-Queens Criteria will have all the entangled ($\frac{N^2}{2} + \frac{N}{2}1$) ancillas in — 1i state, and hence can be distinguished from the other basis states

The construction of the quantum circuit for the N-queens algorithm are attached in the supplementary material. With the quantum circuit we constructed we can simulate it by the quantum simulator:

```

from qiskit import *
from qiskit.quantum_info import Operator
from qiskit.compiler import transpile

from qiskit.providers.aer import AerSimulator
simulator = AerSimulator(method='matrix_product_state', enable_truncation=
    True) # we can try this or the extended stabilizer
tcirc = transpile(qc, simulator, optimization_level=3)

```

```
extended_stabilizer_simulator = AerSimulator(method='extended_stabilizer')



result = simulator.run(tcirc, shots=2000, memory=True).result()
memory = result.get_memory(tcirc)
counts = result.get_counts()

sorted_keys = sorted(counts, key=counts.get, reverse=True)

import numpy as np

a =[]

aci_target = "111111111"
for line in sorted_keys:
    if aci_target in line:
        d = line[0:16]
        #print(d)
        a.append(d)



#print(a)




for i in range(len(a)):
    d_list = []
    c = a[i]
    #print(c)
```

```
    for j in range(16):
        d_list.append(int(c[j]))
    #print(d_list)
    answer = np.reshape(d_list, (4, 4))
    draw_solution(answer)
```

Then we can utilize the ancillary qubits in our setup to target the correct result we want to solve in once. And the result of our demonstration for the four queen puzzles is shown in Fig. 12. The result of this solution is correct and complete. We can definitely use the classical Las Vegas algorithm to solve it. However, all the possibilities of the 4-queens puzzle problem is 256. Therefore, the quantum approach is much more efficient and we can solve the solution in the once try instead of try and error many times. This fascinating idea we make us solve the optimization problem much more quicker and efficient.

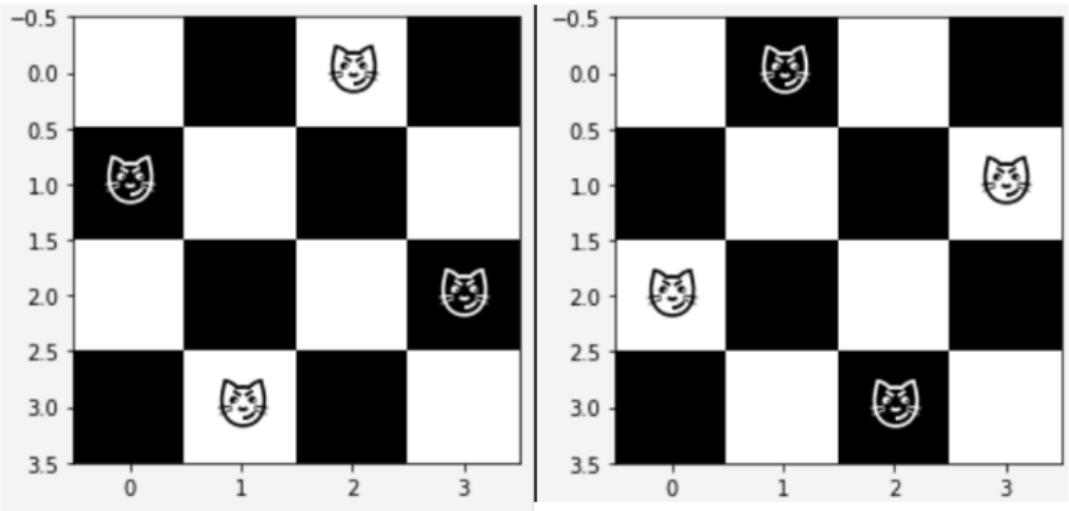

FIG. 12: The all possibility of the eight-queens puzzle result.

IV. CONCLUSION

Through this lecture we intend to introduce Masters students with a QC background to the concepts of combinatorial optimization problems and the implementation of QAOA using Qiskit to solve them. We do this by making use of interesting and illustrative examples like the Dominating set problem and the n-Queens problem. This lecture would last about 45-60 minutes with a major emphasis being on the illustrative examples, so that the students would get a firm grasp of the concepts.

REFERENCES

[1] J. D. Ullman. "NP-complete scheduling problems". In: *Journal of Computer and System sciences* 10.3 (1975), pp. 384–393.

[2] R. L. Karg and G. L. Thompson. "A heuristic approach to solving travelling salesman problems". In: *Management science* 10.2 (1964), pp. 225–248.

[3] A. Caprara, H. Kellerer, and U. Pferschy. "The multiple subset sum problem". In: *SIAM Journal on Optimization* 11.2 (2000), pp. 308–319.

[4] J. Plesn et al. "The NP-completeness of the Hamiltonian cycle problem in planar diagraphs with degree bound two". In: *Information Processing Letters* 8.4 (1979), pp. 199–201.

[5] P. C. Chu and J. E. Beasley. "A genetic algorithm for the multidimensional knapsack problem". In: *Journal of heuristics* 4.1 (1998), pp. 63–86.

[6] V. Torggler, P. Aumann, H. Ritsch, and W. Lechner. "A quantum n-queens solver". In: *Quantum* 3 (2019), p. 149.

[7] R. Jha, D. Das, A. Dash, S. Jayaraman, B. K. Behera, and P. K. Panigrahi. "A novel quantum N-Queens solver algorithm and its simulation and application to satellite communication using IBM quantum experience". In: *arXiv preprint arXiv:1806.10221* (2018).

V. SUPPLEMENTARY

```
test = ''' OPENQASM 2.0;
include "qelib1.inc";

qreg qr[25];
creg cr[25];


h qr[0];
h qr[3];
h qr[4];
```

```
h qr[7];
h qr[8];
h qr[11];
h qr[12];
h qr[15];

x qr[19];
x qr[20];
x qr[21];
x qr[22];
x qr[23];
x qr[24];
x qr[0];
x qr[3];
x qr[4];
x qr[7];
x qr[8];
x qr[11];
x qr[12];
x qr[15];

ccx qr[0],qr[3],qr[1];
ccx qr[4],qr[7],qr[5];
ccx qr[8],qr[11],qr[9];
ccx qr[12],qr[15],qr[13];

x qr[0];
x qr[3];
x qr[4];
x qr[7];
```

```
x qr[8];
x qr[11];
x qr[12];
x qr[15];

ccx qr[0],qr[3],qr[2];
ccx qr[4],qr[7],qr[6];
ccx qr[8],qr[11],qr[10];
ccx qr[12],qr[15],qr[14];

cx qr[2],qr[0];
cx qr[2],qr[3];
cx qr[6],qr[4];
cx qr[6],qr[7];
cx qr[10],qr[8];
cx qr[10],qr[11];
cx qr[14],qr[12];
cx qr[14],qr[15];


h qr[16];
h qr[17];
h qr[18];

cu1(3.14159265358979) qr[16],qr[0];
cu1(3.14159265358979) qr[16],qr[4];
cu1(3.14159265358979) qr[16],qr[8];
cu1(3.14159265358979) qr[16],qr[12];
cu1(3.14159265358979) qr[17],qr[1];
cu1(3.14159265358979) qr[17],qr[5];
```

```
cu1(3.14159265358979) qr[17],qr[9];
cu1(3.14159265358979) qr[17],qr[13];
cu1(3.14159265358979) qr[18],qr[2];
cu1(3.14159265358979) qr[18],qr[6];
cu1(3.14159265358979) qr[18],qr[10];
cu1(3.14159265358979) qr[18],qr[14];


h qr[16];
h qr[17];
h qr[18];

ccx qr[0],qr[5],qr[19];
ccx qr[1],qr[4],qr[19];
ccx qr[1],qr[6],qr[19];
ccx qr[2],qr[5],qr[19];
ccx qr[2],qr[7],qr[19];
ccx qr[3],qr[6],qr[19];

ccx qr[0],qr[10],qr[20];
ccx qr[1],qr[11],qr[20];
ccx qr[2],qr[8],qr[20];
ccx qr[3],qr[9],qr[20];

ccx qr[0],qr[15],qr[21];
ccx qr[3],qr[12],qr[21];

ccx qr[4],qr[9],qr[22];
ccx qr[5],qr[8],qr[22];
ccx qr[5],qr[10],qr[22];
```

```
ccx qr[6],qr[9],qr[22];
ccx qr[6],qr[11],qr[22];
ccx qr[7],qr[10],qr[22];

ccx qr[4],qr[14],qr[23];
ccx qr[5],qr[15],qr[23];
ccx qr[6],qr[12],qr[23];
ccx qr[7],qr[13],qr[23];

ccx qr[8],qr[13],qr[24];
ccx qr[9],qr[12],qr[24];
ccx qr[9],qr[14],qr[24];
ccx qr[10],qr[13],qr[24];
ccx qr[10],qr[15],qr[24];
ccx qr[11],qr[14],qr[24];

measure qr[0] -> cr[24];
measure qr[1] -> cr[23];
measure qr[2] -> cr[22];
measure qr[3] -> cr[21];
measure qr[4] -> cr[20];
measure qr[5] -> cr[19];
measure qr[6] -> cr[18];
measure qr[7] -> cr[17];
measure qr[8] -> cr[16];
measure qr[9] -> cr[15];
measure qr[10] -> cr[14];
measure qr[11] -> cr[13];
measure qr[12] -> cr[12];
measure qr[13] -> cr[11];
```

```
measure qr[14] -> cr[10];
measure qr[15] -> cr[9];
measure qr[16] -> cr[8];
measure qr[17] -> cr[7];
measure qr[18] -> cr[6];
measure qr[19] -> cr[5];
measure qr[20] -> cr[4];
measure qr[21] -> cr[3];
measure qr[22] -> cr[2];
measure qr[23] -> cr[1];
measure qr[24] -> cr[0];



'''
from qiskit import QuantumCircuit
qc = QuantumCircuit.from_qasm_str(test)
```