Solving Vehicle Routing Problem using Variational Quantum Eigensolver

Sagar Gaur
School of Basic Sciences, $IndianInstituteofTechnologyMandi, Kamand, Mandi175005, India.$
(Dated: May 27, 2023)

Vehicle routing problem (VRP) is considered to be one of the industrially significant optimization problems. The VRP is initially formulated as a graph problem. The graph is then optimized for minimal cost path which represents cost-effective route to be taken. In this paper, both classical and quantum computing approaches for solving the VRP are discussed. Classical methods is based on CPLEX optimizer while quantum method is based on the so called variational quantum eigensolver (VQE). The technical details, code are presented in a detailed manner. The optimal paths as suggested by both classical and quantum approaches are visualized. The optimal path and the associated cost obtained using VQE is compared with the classical approach. The obtained values are comparable ensuring the validity of the quantum approach. Therefore, in this work a simple demonstration of solving VRP using quantum approach is performed.

I. INTRODUCTION

Vehicle routing problem (VRP) is one of the industrially significant combinatorial optimization problems. The objective is to identify the route or the path that works out at a minimal cost. In general, the problem is converted into a graph and is solved with the objective function with desired constraints. Let's say I have to draw a road map (cities and roads) on paper. This can be done with the help of points(cities) and edges(paths) connecting vertices. Let u,v,w, and z be cities and we have road-map as shown in Figure 1, which depicts that u,v, and w are connected to each other by road but 'z' is only connected to 'w'. Points u,v,w, and z are vertices and the lines connecting them are edges, and the resulting whole diagram is called a graph. Similarly, graphs can be used to represent electrical networks circuit, network connections between computers, etc. In the early 18Th century, there was a recreational mathematical puzzle called the Königsberg bridge problem. Finding a solution to this problem, opened a completely new field in mathematics called Graph Theory. In this, we use graphs to solve the problems like traveling salesman problem and vehicle routing problem, which are discussed in this paper.

In general, a graph has two components, nodes and edges. A node is basically a point and an edge is a line that connects two nodes. For e.g., in 2D map of a state or country, the cities can be considered as the nodes and the roads connecting them can be considered as the edges. We can attach direction to the edges when the roads connecting cities are only one way. The resulting graph is a directed graph otherwise called an undirected graph. We can attach a certain weight to every edge in a graph such as the cost of travel between the cities is called a weighted graph otherwise called an unweighted Graph.

A. Graph

Simple Graph G(V, E) is a collection of set V(G) of elements called vertices and set E(G) of **distinct unordered pairs of distinct elements** called edges. we called V(G) a set of vertices and E(G) a set of edges of G. An Edge is written as VW if it joins vertex V and W. For example(Figure 1).

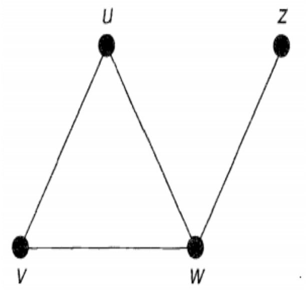

Figure 1: Graph

Here,V(G) is v,u,w,z and E(G) is uv,uw,wz and wv.If in a certain graph two vertices are joined by two or more edges then that graph is called Multi edge graph and an edge that connects a vertex to itself is called a loop. A resulting object is called **general graph or graph** in which there is no multi-edge and loops are allowed. Hence, every graph is a simple graph but every simple graph is not a graph.

B. Types of graphs

1. Null graph

A graph for which $E(G) = \{\Phi\}$ i.e a graph without any edges. (Figure 2)

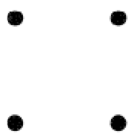

Figure 2: Null graph

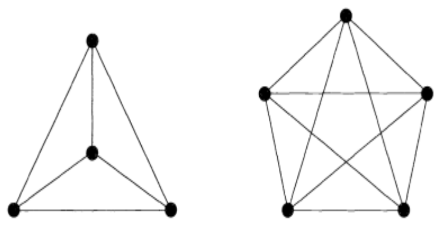

Figure 3: Complete Graph

2. *Complete graph*

A simple graph in which each pair of different vertices are linked by an edge is called a complete graph. Hence, we have ${}^nC_2 = \frac{n(n-1)}{2}$ number of edges in a complete graph shown in Figure 3.

3. *Connectivity*

Walk: A walk in a G is an ordered sequence of edges i.e vw, wx, xy, yz, zz, zy, yz and also by $v \to w \to x \to y \to z \to z \to y \to w$, in which every two vertexes are adjacent(connected by an edge). (Figure 4) Starting point of the walk is called **initial vertex** and the endpoint is called **final vertex**. The number of edges in the walk is called is length.

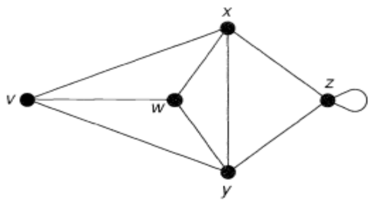

Figure 4: Connectivity

II. THE TRAVELLING SALESMAN PROBLEM

A handbook for travelling salesmen from 1832 mentions this problem but contains no mathematical treatments. It was mathematically formulated in the 19th century by Irish mathematician William Rowan Hamilton.

The travelling salesman problem asks the question: "We are given a list of cities and the distances between each pair of cities. We have to find the shortest possible path such that the salesman visits each city once and only once and returns to the city from where he started?" .

This problem can be solved by combinatorial optimization, which is a subfield of mathematical optimization in which we search for the optimal object from the finite set of objects among the various discrete feasible solutions. There are $(n-1)!$ such tour exist and we have to find one tour for which we have to minimize the cost function which can be total time taken, total distance travelled or total money spent. The brute force method in which we go through every possible path satisfying the constraint of TSP and compare the cost function value for each path and then give an optimal path is not a good way for solving the problem as the number of cities increases this problem becomes difficult to solve. It is an NP-hard problem with time complexity $O(n!)$.

One can write this as a graph problem as well: Given a complete weighted graph where vertices are cities and edges would represent the roads and weight would be the distance or cost of the road. Now, we have to find a tour with a minimum weight such that we end on the same point where we have started after visiting every city once and only once. To solve TSP we have to set up an integer linear program. We have several formulations but I will discuss MTZ formulation.

A. The MTZ Formulation

In this formulation, we label the cities with the numbers 1,2.....,n and take $w_{ij} \geq 0$ to be the distance from city i to j.Now, for a certain possible path(Consistence with the constraints of TSP but may not have minimum cost function) x_{ij} can take two values .
$x_{ij} = 1$, i and j are connected.
$x_{ij} = 0$, i and j are not connected.
x_{ij} could be thought of as a matrix of elements 0 and 1 such that we have one unique matrix x_{ij} for every possible path. For a particular possible path i.e. particular x_{ij}, We can calculate the cost function(tour length) which we have to minimize. Hence, we have to minimize tour length

$$f = \sum_{i=1}^{n} \sum_{j=i,j=1}^{n} w_{ij} x_{ij}$$

and also we have 2n linear constraints

1. $\sum_{i=1,i\neq j}^{n} x_{ij} = 1$ for j=1,2,.....n

2. $\sum_{j=1,j\neq i}^{n} xji = 1$ for i=1,2,.....n

1st constraint is true as there is always only one edge that is from 'i' to any other vertex. 2nd constrain is true because there is always only one edge coming to ith city.
We have constraints to remove sub-tour:

1. $u_i - u_j + (n-1)x_{ij} \leq n-2 \; for \; 2 \leq i \neq j \leq n$

2. $1 \leq u_i \leq n-1 \; for \; 2 \leq i \leq n$

Here, u_i is a variable that keeps track of the order in which cities are visited. If $u_i \leq u_j$ means that j visited after city i. u_i can be taken as the number of edges traveled while reaching city 'i' from city 1. This problem can be extended and give rise to a more general problem called the vehicle routing problem.

III. THE VEHICLE ROUTING PROBLEM

Logistics is a major industry of about USD 8183 billion according to data of 2015. In logistics, most companies have a number of vehicles to serve a number of clients during day time and have a depot where vehicles are kept overnight. The problem can be stated as "How to design a tour from depot to the depot through all clients so that we can minimize cost function which can be vehicle miles travelled, time spent, or similar objectives".

A. The Model

Let the number of nodes(clients+depot) = n, and the number of available vehicles = K. In this formulation, we label the clients with the numbers 1,2.....,n-1 and depot by 0. We take $w_{ij} \geq 0$ to be the distance from city i to j. Now, for a certain possible path(Consistence with the constraints of TSP but may not have minimum cost function) x_{ij} can take two values.
$x_{ij} = 1$, i and j are connected.
$x_{ij} = 0$, i and j are not connected.
x_{ij} could be thought of as a matrix of elements 0 and 1 such that we attach one unique matrix x_{ij} for every possible path.
If any two clients i and j have a link between them, we write i $\sim$ j. Let $\delta(i)^+$ be a set of nodes to which i has a link and $\delta(i)^-$ is a set of nodes that are linked to i for a graph(roadmap). If we say j $\in \delta(i)^+$,then i $\sim$ j. In addition, we will consider 'n' more continuous variable attached to each node u_i to eliminate sub-tour. VRP can be formulated as:

1. We find x_{ij} for which f is minimum.

$$f = \sum_{i=0,j=0}^{n} w_{ij}x_{ij} \tag{1}$$

2. We have node visiting constraints as every node should be visited and exited only once.

$$\sum_{j\in\delta(i)^+} x_{ij} = 1 \; \sum_{j\in\delta(i)^-} x_{ji} = 1 \tag{2}$$

3. Depot visiting constrain

$$\sum_{j\in\delta(0)^+} x_{0j} = K \; \sum_{j\in\delta(0)^-} x_{j0} = K \tag{3}$$

4. Next is the sub-tour elimination constraint:

$$u_i - u_j + Qx_{ij} \leq Q - q_i \;\; \forall i \sim j, i, j \neq 0$$

$$q_i \leq u_i \leq Q \;\; \forall i \neq 0$$

where Q is vehicle capacity and q_i is the demand of each client.

B. The solution

The solution to the problem is obtained by utilizing the CPLEX package implemented in Python. CPLEX employs the branch and bound technique along with the cut method to discover an approximate solution for the Vehicle Routing Problem (VRP). We can represent a vector of decision variables as follows:

$$\vec{x} = [x_{00}, x_{01}, x_{02}, ..., x_{10}, x_{11}, ..., x_{(n-1)(n-2)}]$$

we find 'x' for which 'f' is the minimum.

IV. CLASSICAL SOLUTION USING CPLEX

The paper addresses a problem that is resolved using the CPLEX package in Python. The term "CPLEX" is derived from the "Simplex" algorithm, which happens to be the pioneering commercial optimizer implemented in the C language. CPLEX utilizes the branch and bound technique along with the cut method to approximate a solution for the Vehicle Routing Problem (VRP).

Now, I will give some information about how we can create a model to solve an optimization problem. More information could be found on IBM documentation on Cplex. First, we have to create a problem instance by using ".Cplex()" method from cplex as shown in code below.

```
import cplex
model = cplex.Cplex()
```

1. We can add the coefficient of function to be minimized or maximized, decision variables, upper bound and lower bound using "Cplex.variable" interface.
2. Sense of the problem i.e minimisation or maximisation is set through "Cplex.Objective" interface.
3. "Cplex.linear_constraints" package is used to add linear constraints to "model"
4. .solve() is used to solve the model we have and we can access the output by the "Cplex".solution interface.

A. Methods of CPEX

1. variables.add(obj, lb, ub, names, types):Decision variables can be added to this problem using this method.
   (a) object : It should be the list of coefficient of decision variables. Its order is based on the decision variable list.
   (b) lb : A list of lower bounds on the variables.
   (c) ub : A list of upper bounds on the variables.
   (d) types : It is a list of the type of variables.We have to use "C" for continuous variable and "I" for integer variable and "B" for binary variable
2. (a) lin_expr: It is a list of list, first of decision variable numbers and then their coefficients.
   (b) senses: It is a list of type of constraints."E" is for equality, "L" is for less than equal to and "G" is for greater than or equal to constraints.
   (c) rhs: It is a list of the right-hand side of constraints.
3. ObjectiveInterface.set_sense(sense) Using this we can set a sense that whether we have to minimize or maximize the cost function. As shown in following code

```
model.objective.set_sense(model.objective.sense.maximize)#to set objective maximum
model.objective.set_sense(model.objective.sense.minimize)#to set objective minimum
```

V. PROGRAM IN PYTHON TO SOLVE VRP

```


#For importing important packages
import numpy as np
import cplex                      #importing cplex for optimization
import matplotlib.pyplot as plt # importing matplotlib.pyplot to plot graph
import math
#fixing the values of node and no. of vehicles
num=3                        #Numbr of clients plus one depot (n=num)
v=2                          #Number of vehicles(K=v)
Q=1                          #Capacity of a vehicle
q=0.1                        #Demand of each Client
#Generating nodes at random places
np.random.seed(15)
xp=(np.random.rand(n)-0.5)*10        #x coordinate of clients
yp=(np.random.rand(n)-0.5)*10       #y coordinates of clients
plt.plot(xp,yp,'o')                 # ploting points using matplotlib package
distance = np.zeros([num, num]) #initializing distance matrix to store between nodes
#for loop to calculate the distance between the nodes
for i in range (0,num):
    for j in range(0,num):
        distance[i,j]=((xp[i]-xp[j])**2 + (yp[i]-yp[j])**2)
for i in range(len(xp)):
        plt.annotate(i, (xp[i]+0.15, yp[i]+0), size=16, color="r")
plt.plot(xp[0], yp[0], "r*", ms=20)
plt.grid()
#Setting up the problem
model=cplex.Cplex()
obj1 = list(distance.reshape(1, num**2)[0])+[0.0 for x in range(0, num - 1)]
ub1=list(1 for i in range(0,num**2+num-1))
lb1=list(0 for i in range(0,num**2+num-1))
types1="".join(list('I' for i in range(0,num**2)))+ "".join(list('C' for i in range(0,num-1)))
model.variables.add(obj=obj1, lb=lb1, ub=ub1, types=types1)
model.objective.set_sense(model.objective.sense.minimize)
#Given the constrain of the problem
my_rhs = (2 * ([v] + [1 for x in range(0, num - 1)])+ [(Q - q) for x in range(0, (num - 1) ** 2 - (num - 1))]+ [0 for x in range(0, num)])
r=list()

for i in range(0, num):
    c = [x for x in range(0 + num * i, num + num * i)]
```

```
41     co = [1 for x in range(0, num)]
42     r.append([c, co])
43 for i in range(0, num):
44     c = [x for x in range(0 + i, num**2,
          num)]
45     co = [1 for x in range(0, num)]
46     r.append([c, co])
47 #Following code is to eliminate subtours
48 for i in range(0, num):
49     for j in range(0, num):
50         if (i != j) and (i * j > 0):
51             c = [i + (j * num), num**2 +
                    i - 1, num**2 + j - 1]
52             co = [1, 1, -1]
53             r.append([c, co])
54 for i in range(0, num):
55     c = [(i) * (num + 1)]
56     co = [1]
57     r.append([c, co])
58 my_sense = ("".join(["E" for x in range
       (0, 2 * num)])+ "".join(["L" for x
       in range(0, (num - 1) ** 2 - (num -
       1))])+ "".join(["E" for x in range
       (0, num)]))
59
60 model.linear_constraints.add(lin_expr=r,
        senses=my_sense, rhs=my_rhs)
61 model.solve()
62 x = model.solution.get_values()
                    #x value is now
       changes with otimized value of x
63 x = np.array(x)
64 x=list(x)
65 cost = model.solution.
       get_objective_value()     #cost is
       the optimum value of the cost
       function
66 #Visualizing the solution
67 title_str="optimized_path"
68 plt.figure()
69 plt.scatter(xp, yp, s=200)
70 for i in range(len(xp)):
71         plt.annotate(i, (xp[i]+0.15, yp[
               i]+0), size=15, color="o")
                    #to give numbers to the
                 nodes in red colour
72 plt.plot(xp[0], yp[0], "r*", ms=18)
73 plt.grid()
74 for i in range(0, num**2):
75
76     if x[i] > 0:
77         x = i // num
78         y = i % num
79         plt.arrow(xp[x],yp[x],xp[y] - xp
               [x],yp[y] - yp[x],
               length_includes_head=True,
               head_width=0.22,)
80 plt.title(title_str + " cost = " + str(
       int(cost * 100) / 100.0))
81 plt.show()
```

1. First we have stored coefficient x_{ij} and u_i of $f = \sum_{i=0,j=0}^{n} w_{ij}x_{ij} + \sum_{i=0}^{n-1} u_i$ into my_obj
2. In line 25, my_ub is the list of uper bound on the variables
3. In line 26, my_lb is the list of lower bound on the variables.
4. In line 27, my_types is list of types of variable store the types of the variable we are using: 'I' is for integer type variable and 'C' is for continuous variable.
5. In line 28, we passed these variables into variable.add method
6. In line 35, my_rhs contains the list of the right-hand side of constrain.
7. 'r' is a list of list, first of decision variable numbers and then their coefficients in the constraint
8. In line 59, my_sense is the list of the type of constraints we have. 'E' is for equality constraint and 'L' is for the less than.
9. After that we passed the variable rows,my_senses and my_rhs into the model.linear_constraints (line 62)

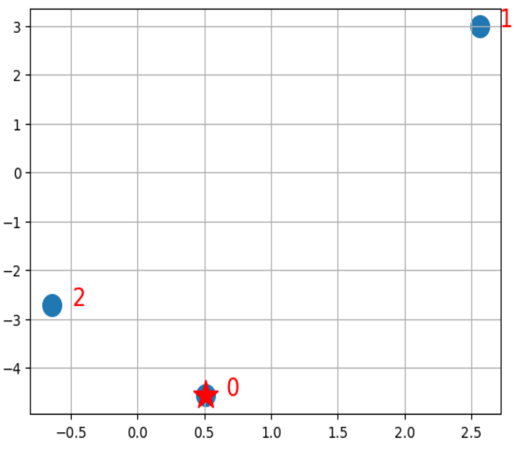

Figure 5: Plot of clients and depot: depot is shown by zero and clients are shown by 1,2

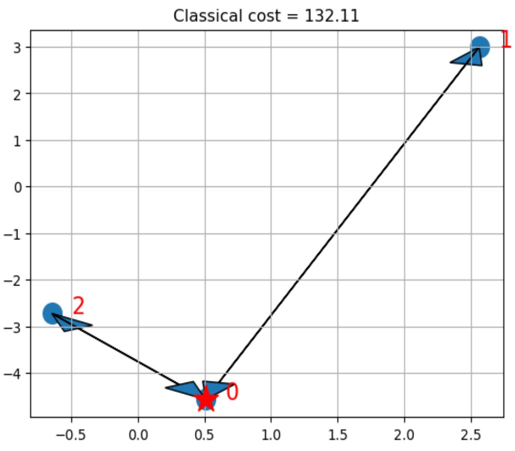

Figure 6: Minimum Cost: depot is shown by 0 and clients are shown by 1,2

VI. RESULTS : VISUALIZING PATH OF MINIMUM AND MAXIMUM COST

1. First, we have some random coordinates for clients and depot as shown in figure 5.
2. We have calculated the minimum cost route using the program. (shown in figure 6)
3. For the classical approach the code used to optimize the representative graph has been developed based referring to the existing IBM's CPLEX optimizer. The result thus obtained matches well the IBM's optimizer.

VII. QUANTUM SOLUTION

The real-world quantum problem either has high time complexity or we can solve it by relaxing its constraint in polynomial time complexity. Generally, an increment in the size of the problem increases the resources required and hence increases the hardness of the problem. As required resources increase exponentially, this makes the problem computationally unsolvable. Researchers currently working to make the available algorithms more efficient and to make new efficient algorithms. In terms of computational time complexity, VRP is an NP-hard problem and there does not exist any polynomial time-taking algorithm to solve VRP. We can look forward to the quantum algorithm which can solve this problem with less time complexity.

Quantum computing devices have advantages over classical devices as they can use resources like quantum entanglement and superposition to solve problems more efficiently. Nevertheless, quantum hardware is still noisy and today quantum devices are also known as Noise Intermediate-Scale Quantum (NISQ) devices. Noise comes into the devices due to the imperfect gates and qubit decoherence. Decoherence is there due to the the interaction of qubit with the environment due to which it losses the information and coherence. Hence, they can only run the algorithm with a limited circuit depth. We will solve the problem using variational quantum eigensolver(VQE) is a hybrid algorithm i.e. it also uses a classical optimizer with the quantum computer to solve a particular optimization problem introduced by E. Farhi, J. Goldstone, S. Gutmann in 2014. Quantum Annealing[11] is another technique to solve optimization problems and D-Wave's quantum computers, known as D-Wave systems, are designed to implement this specific approach. Hence, there are two techniques to solve optimization problems, one is gate based and another is using quantum annealing. VQE is gate-based and the Quantum approximation optimization algorithm(QAOA) is annealing-based computing. In this NISQ era, VQE is most widely used to carry out computation. In this paper, I will use VQE to solve the VRP, and VQE is expected to prove quantum supremacy in solving VRP. First, I will discuss how adiabatic quantum computing can solve the problem and explain VQE.

A. Adaibatic Quantum Computing(AQC)

It is a theoretical framework for a quantum computer that operates based on the adiabatic theorem. In contrast, the circuit model represents quantum algorithms using circuits composed of quantum gates to perform unitary operations. Edward Farhi, a professor at MIT and the creator of the Quantum Approximate Optimization Algorithm (QAOA), was also among the first to propose adiabatic quantum computation in year 2000.

Its focus centers around two fundamental elements: the mixer Hamiltonian $\hat{H}_{mixer}$ and the cost Hamiltonian $\hat{H}_{cost}$. Hamiltonians serve as representations of the energy states within a quantum system.

In essence, the AQC methodology involves initiating the process with a readily attainable ground state, embodied by the ground state of the mixer Hamiltonian $\hat{H}_{mixer}$. The objective then entails a gradual progression of the system, transitioning from the easily accessible ground state towards the sought-after quantum state housed in the ground state of the cost Hamiltonian $\hat{H}_{mixer}$. Defining s(t) a continuous function such that $s(0) = 0$ and $s(T) = 1$ is known as *schedule*. The value of T should be large for adiabatic theorem to hold. The Hamiltonian of the system at an instant can be written as :

$$\hat{H}(t) = (1 - s(t)\hat{H}_{mixer} + s(t)\hat{H}_{cost} \quad (4)$$

By Adiabatic Theorem, we can say for the entire time evolution [0,T] we will be in the ground state of $\hat{H}(t)$. To know time evolution under this time-dependent Hamiltonian we need to solve hard integral:

$$U(t) = \exp[\frac{-i}{\hbar}\int_0^t H(t)dt] \quad (5)$$

But we can make it less severe with the Trotterization technique [9].

. We partition the time interval of $U(T)$ into smaller intervals of Δt, ensuring that the Hamiltonian remains nearly constant within each interval. This discretization enables us to employ a simplified formula for the time-independent Hamiltonian. In this context, we denote U(b, a) as the representation of time evolution from the initial time a to the final time b,

$$\begin{aligned} U(T,0) &= U(T, T-\Delta t)U(T-\Delta t, T \\ &\qquad - 2\Delta t).....U(\Delta t, 0) \\ &= \prod_{j=0}^{p} U(j\Delta t, (j-1)\Delta t) \\ &\approx \prod_{j=1}^{p} e^{-iH(j\Delta t)\delta t} \end{aligned} \tag{6}$$

We can improve our approximation by increasing the p and decreasing the Δt. Using trotter -Suzuki formula i.e $e^{i(A+B)x} = e^{iAx}e^{iBx} + O(x^2)$ and putting in Hamiltonian $H(j\Delta t) = (1 - s(j\Delta t))H_{mixer} + s(j\Delta t)H_{cost}$ in equation 6

$$\begin{aligned} U(T,0) \approx \prod_{j=1}^{p} (e^{-i(1-s(j\Delta t))H_{mixer}\Delta t} \\ e^{-is(j\Delta t))H_{cost}\Delta t}) \end{aligned} \tag{7}$$

Consequently, we can approximate adiabatic quantum computation (AQC) by iteratively allowing the system to evolve under the influence of the problem Hamiltonian H_{cost} for a small duration of $s(j\Delta t)\Delta t$, followed by the driver Hamiltonian H_{mixer} for a small duration of $(1 - s(j\Delta t))$. The unitaries representing these operations can be efficiently constructed as $U = e^{(-iH\Delta t)s}$, where is a number ranging from 0 to 1 and incorporates the scaling factor determined by $s(j\Delta t)$.

$$\begin{aligned} U = U(\hat{H}_{mixer}, \beta_o)U(\hat{H}_{cost}, \gamma_o)... \\ U(\hat{H}_{mixer}, \beta_p)U(\hat{H}_{cost}, \gamma_p) \end{aligned} \tag{8}$$

In the realm of gate-model quantum computation[10], this implies that we start with an initial product state $|\psi_{GS}^{HM}\rangle$, and then apply a series of parameterized gates to generate the state $|\psi^{HC}\rangle$. By finding the optimal values for the parameters $\{\beta^*, \gamma^*\}$, we can obtain the ground state of the Hamiltonian $\hat{H}_{cost}$. The classical processor provides the parameters $\{\beta, \gamma\}$, which are further refined through a classical optimization routine based on measuring the energy of the final state $|\psi^{HC}\rangle$. Consequently, the Quantum Approximate Optimization Algorithm (QAOA) falls within the category of hybrid quantum-classical variational algorithms.

B. Gate-based quantum computing

Gate-based quantum computing is a computational paradigm that utilizes quantum gates to perform operations on qubits. Qubits are the quantum counterparts of classical bits, capable of existing in superposition states. Quantum gates, such as Pauli-X, Hadamard, and Controlled-NOT, manipulate the quantum states of qubits. By arranging these gates in quantum circuits, computations can be performed. Measurements extract classical information from the quantum state. Gate-based quantum computing holds promise for applications in quantum simulations, algorithms, and error correction. Currently, quantum computer is Noisy refered as Noise Intermediate-Scale Quantum (NISQ). VQE (Variational Quantum Eigensolver) on NISQ (Noisy Intermediate-Scale Quantum) devices refers to the implementation of the VQE algorithm specifically designed to run on current-generation quantum computers with limited qubit coherence and high error rates.

Running VQE on NISQ devices poses unique challenges due to their inherent noise and error rates. These devices are prone to errors in qubit operations and measurements, limiting the accuracy and reliability of computations. Additionally, the number of qubits and gate operations available in NISQ devices is typically limited, constraining the size and complexity of the quantum systems that can be effectively simulated.

1. *Variational Quantum Eigensolver(VQE)*

VQE stands for Variational Quantum Eigensolver. In contrast to Quantum Annealing, it is a gate-based quantum algorithm used for solving problems related to quantum chemistry, material science, and optimization. VQE is designed to leverage the potential of quantum computers to solve problems that are challenging for classical computers.

The VQE algorithm aims to find the lowest energy state (ground state) of a given quantum system. It is based on a hybrid approach that combines classical and quantum computations. Here's a simplified explanation of how VQE works[12]:

1. Hamiltonian: The Hamiltonian represents the system's energy operator and is typically defined as the sum of kinetic energy terms and potential energy terms. In the case of quantum chemistry, it represents the electronic structure of a molecule.

$$H = \sum_i c_i O_i$$

Here, O_i are the operators corresponding to the individual terms in the Hamiltonian, and c_i are their corresponding coefficients.

2. Ansatz: An ansatz is a parameterized quantum circuit that is used to prepare a trial state $|\psi(\theta)\rangle$ depending on a set of parameter θ. The ansatz is designed to capture the features of the ground state of the system. It is typically a combination of quantum gates acting on an initial state, such as the all-zero state $|0\rangle$ or a Hartree-Fock state.

$$|\psi(\theta)\rangle = U(\theta)\,|0\rangle$$

3. Expectation Value: The expectation value of an operator O_i with respect to trail state $|\psi\rangle\,(\theta)$ is computed as:

$$\langle O_i\rangle = \langle\psi(\theta)|\,O_i\,|\psi(\theta)\rangle$$

4. Cost Function: The cost function, also known as the objective function, is defined as the expectation value of the Hamiltonian. The goal is to find the minimum value of the cost function, which corresponds to the approximation of the ground state energy.

$$E(\theta) = \frac{\langle\psi(\theta)|\,H\,|\psi(\theta)\rangle}{\langle\psi|\,|\psi\rangle}$$

5. Optimization: The last step involves finding the optimal set of parameters θ that minimizes the cost function. This is achieved using classical optimization algorithms such as gradient descent or the Nelder-Mead method. We will use the Simultaneous Perturbation Stochastic Approximation (SPSA) as a classical optimizer in our program.

6. By iteratively adjusting the parameters and evaluating the cost function, the VQE algorithm aims to find the set of parameters that yield the lowest energy approximation to the ground state of the Hamiltonian.

VQE is considered a promising algorithm because it allows for the calculation of properties and energies of molecules or materials, which can be useful in areas like drug discovery, catalyst design, and materials science. However, it is important to note that VQE's performance is currently limited by the noise and error rates in current quantum hardware. Ongoing research and advancements in quantum computing technology are expected to improve the effectiveness and scalability of VQE and other quantum algorithms. Now, we describe how classical optimizer in VQE.

In the context of Variational Quantum Eigensolver (VQE), the Simultaneous Perturbation Stochastic Approximation (SPSA) optimizer is commonly used to find the optimal parameters for the variational quantum circuit.

VQE is an algorithm used to estimate the ground state energy of a given quantum system using a parameterized ansatz wavefunction and a classical optimization routine. The variational quantum circuit is constructed with adjustable parameters, typically represented by angles, that define the operations applied to the qubits. The objective of VQE is to find the set of parameters that minimizes the expectation value of the Hamiltonian of the quantum system. VQE is hybrid algorithm and use classical optimizer. We will use SPSA optimizer. SPSA is well-suited for optimizing the parameters in VQE because it does not require explicit computation of gradients. Instead, SPSA estimates the gradients by evaluating the objective function at two perturbed parameter settings. This stochastic approach allows for efficient exploration of the parameter space, even in the presence of noise and when the gradient information is not readily available.

The SPSA optimizer iteratively adjusts the parameters based on the estimated gradients, aiming to minimize the objective function. At each iteration, SPSA randomly perturbs the parameters and measures the corresponding energy expectation values. By using these measurements, SPSA updates the parameter values to drive the objective function towards a minimum.

The advantages of using SPSA in VQE lie in its ability to handle noisy measurements and its efficiency in exploring high-dimensional parameter spaces. VQE often involves quantum systems with limited qubit coherence and noise, and SPSA's stochastic nature allows it to cope with these challenges effectively. Additionally, SPSA's capability to estimate gradients using a small number of function evaluations makes it well-suited for optimizing variational quantum circuits that typically have a large number of parameters.

Overall, SPSA is a popular choice for optimizing the parameters in VQE due to its robustness, efficiency, and ability to handle noisy measurements. It enables the VQE algorithm to find approximate ground state energies of quantum systems using variational quantum circuits and classical optimization routines. We will use real amplitude ansatz for solving the problem.he ansatz real amplitude is a concept commonly used in quantum computing and quantum simulations. In quantum systems, an ansatz refers to an educated guess or a trial wavefunction that is used as a starting point for solving a quantum problem. The real amplitude ansatz specifically focuses on constructing a wavefunction that consists of real-valued amplitudes.

By restricting the amplitudes to be real numbers, the real amplitude ansatz simplifies the calculations and computational requirements compared to more general complex-valued ansatz constructions. This can be advantageous in certain cases where the problem at hand can be effectively described by real-valued wavefunctions.

To solve a problem using VQE, we need to convert our problem in Quadratic Program(QP).

VIII. QUADRATIC PROGRAM

QUBO stands for Quadratic Unconstrained Binary Optimization. It is a mathematical formulation used to represent optimization problems in the field of computer science and operations research. QUBO is particularly relevant in the context of quantum computing, as it is a common problem representation for certain types of optimization problems that can be solved using QAOA and VQE.

QUBO, the objective is to find an assignment of binary variables that minimizes or maximizes a quadratic objective function. The variables can take binary values (0 or 1), and the objective function is quadratic, meaning it involves terms with pairs of variables and their coefficients. The general form of a QUBO problem can be expressed as: minimize (or maximize)

$$\sum_{i,j} Q_{ij}x_i x_j + \sum_i g_i x_i + c \tag{9}$$

where Q_{ij} represents the coefficient of the quadratic term for variables x_i and x_j, c_i represents the coefficient of the linear term for variable x_i, and x_i, x_j are binary variables. which in matrix form can be written as :

$$x^T Q x + g^T x + c \tag{10}$$

where Q is the matrix corresponding to Q_{ij} and g is the vector-matrix corresponding to element g_i . To convert a problem into QUBO we need the following points to be obeyed :

1. Define the decision variables: Identify the variables that represent the components or elements of your problem. Each variable should have a binary value, typically denoted as x_i (where i represents the index or label of the variable).

2. Formulate the objective function: Express the objective of your problem as a function to be minimized or maximized. The objective function should involve the decision variables and reflect the goal or measure of success for your problem. The objective function can be linear, quadratic, or a combination of both.

3. Encode constraints (if any): Identify the constraints in your problem that need to be satisfied. If a constraint is of the form inequality, a slack variable is introduced to convert inequality constraints into equality constraints, allowing them to be directly incorporated into the formulation of the objective function. The introduction of slack variables allows us to rewrite the original problem as an equality-constrained linear programming problem.

4. Map variables to binary variables: Convert the original variables and introduced slack variables into binary variables. Assign binary variables to represent the different states or values that these variables can take.

5. The penalty method involves introducing penalty parameters that control the strength of the penalties imposed on constraint violations. The penalty term is added to the objective function and quantifies the extent to which the constraints are violated. The objective becomes a combination of the original objective function and the penalty term, aiming to minimize or maximize the combined value. The penalty term typically takes the form of a weighted sum of constraint violations, where each violation is squared or raised to a power to emphasize larger violations. The squared or powered term ensures that the penalty increases more rapidly as the constraints are violated more severely.

6. Determine the coefficients: Assign appropriate coefficients to the terms in the QUBO formulation, considering the strengths and importance of each term. Ensure that the coefficients accurately reflect the relationships between variables and constraints in your problem.

For the VRP, the variables x_{ij} in equations 1, 2 and 3 are already binary. Now we formulate a new objective function that contains the essence of the constraint such that the minimum of the new function is the minimum of the old objective function of VRP(equation 1). We consider the case $K = n - 1$, In this case, subtour elimination constraints become redundant and the problem is only in terms of $\mathbf{z}$. Now, the

left constraints are only equality type. Hence, we do not need to introduce any slack variable. To remove the constraint we add penalties to the objective function as shown below:
Let say we have to minimize a function f(x) subjected to constraint constraint 11 and 12

$$h(x) = 0, i = 0, ..., m \tag{11}$$

$$g_j(x) \leq 0, j = 1, ..., r \tag{12}$$

We can rewrite this problem equivalently to minimize a new objective function 13

$$f(x) + A[\sum_{(i=1)}^{m} (h_i(x))^2 + \sum_{j=1}^{r} [max(0, g_j(x))]^2] \tag{13}$$

In equation 13, the expression

$$\sum_{(i=1)}^{m} (h_i(x))^2 + \sum_{j=1}^{r} [max(0, g_j(x))]^2$$

is called penalty function P(x) and A is called penalty coefficient.

In our case, we are only left with the equality constraints 2, 3.

Now we formulate a new objective function for VRP by adding penalties to our previous objective function 1. and call it augmented Lagrangian (H) given as:

$$H_{VRP} = H_A + H_B + H_C + H_D + H_E \tag{14}$$

$$H_A = A \sum_{i \sim j} w_{ij} x_{ij} \tag{15}$$

$$H_B = A \sum_{i \in 1,2,....n-1} \{(1 - \sum_{j \in \delta(i)^+} x_{ij})\}^2 \tag{16}$$

$$H_B = A \sum_{i \in 1,2,....n-1} \{(1 - \sum_{j \in \delta(i)^-} x_{ij})\}^2 \tag{17}$$

$$H_D = A(K - \sum_{j \in \delta(0)^-} x_{0j}) \tag{18}$$

$$H_D = A(K - \sum_{j \in \delta(0)^-} x_{j0}) \tag{19}$$

Here, A>0 is penalty coefficient which is large. H_{VRP} is augmented Lagrangian which we will minimize. Now, we have to write H_{VRP} in the standard form 9. Before, doing so we define some variables to be used. We represent all variables x_{ij} by a vector matrix as

$$\vec{x} = [x_{00}, x_{01}, x_{02}, ..., x_{10}, x_{11}, ..., x_{(n-1)(n-2)}]^T \tag{20}$$

Next, we define attached to every node i: $z_{\delta(\vec{i})^+}$ and $\vec{z}_{\delta(i)^-}$. The vector $\vec{z}_{\delta(i)^+}$ is $\vec{x}$ with

$$x_{ij} = 1, x_{kj} = 0 \ \ if \ k \neq i, \forall j, k \in \{0, ..., n-1\}$$

Similarly, the second vector $\vec{z}_{\delta(i)^-}$ is $\vec{x}$ with

$$x_{ji} = 1, x_{jk} = 0 \ \ if \ k \neq i, \forall j, k \in \{0, ..., n-1\} \tag{21}$$

We can represent 14 in the standard QUBO form function 10 and get Q, g and c as follows :

$$Q = A[[\vec{z}_{\delta(0)^-}, ..., \vec{z}_{\delta(n-1)^-}]^T [\vec{z}_{\delta(0)^-}, ..., \vec{z}_{\delta(n-1)^-}] + (I_n \otimes J_{(n-1,n-1)})] \tag{22}$$

$$g = W - 2Ak((e_o \otimes J_{(n-1)}) + [\vec{z}_{\delta(0)^-}]^T + 2A(J_n \otimes J_{n-1}) \tag{23}$$

$$c = 2A(n-1) + 2AK^2 \tag{24}$$

Where, I is identity matrix, J is a matrix having all elements 1 and e_o is the basis vector $[1, 0, 0, ..., 0]^T$ and W is weight matrix. From this, we can construct Ising Hamiltonian for Vehicle Routing Problem. We substitute Q, g, and c in equation 10. Next, We construct the ising hamiltonain of VRP.

IX. ISING HAMILTONIAN

The Ising Hamiltonian used in statistical mechanics to represent a mathematical formulation for ferromagnetic substance. This model contains the variables representing atomic magnetic dipole moments "spins" in one of the two states that can be +1 or -1. The atoms having spins are placed in lattice periodic arrangement in all directions, due to which atom interacts with each other. The system tend to have the lowest energy possible but due to thermal excitation there is disrupt in this tendency. As a model of reality, this model enables for identification of phase transition. Following Hamiltonian can be written for the total energy:

$$H_{ising} = -\sum_i \sum_{j<i} J_{ij} s_i s_j - \sum h_i s_i + d \tag{25}$$

In this equation:
H_{ising} represents the Ising Hamiltonian. J_{ij} represents the interaction strength between spins i and j adjacent spins. and h represents the application of the external magnetic field.

The Ising Hamiltonian can be used to model a variety of physical systems, such as magnetic materials or spin glasses. In quantum computing, it is often employed in the context of quantum annealing, where the objective is to find the ground state of the Hamiltonian, which corresponds to the lowest energy configuration of the system.

Now, we have to convert our QUBO into Ising Hamiltonian form[13]. To do this we expand the equation 9 and use 22 ,23 and 24. Then we rewrite all the binary variable $x_{ij} \in \{0,1\}$ using spin variable $s_{ij} \in \{-1,+1\}$ using[14] :

$$x_{ij} = \frac{s_{ij}+1}{2}$$

. We group together quadratic, linear and constant term in the expression. we get :

$$H_{ISING} = \sum_i \sum_{j<i} J_{ij} s_i s_j - \sum h_i s_i + d \tag{26}$$

$$J_{ij} = -\frac{Q_{ij}}{2} \quad \forall i \leq j \tag{27}$$

$$h_i = \frac{g_i}{2} + \sum_j \frac{Q_{ij}}{4} + \sum_j \frac{Qji}{4} \tag{28}$$

and

$$d = c + \sum_i \frac{g_i}{2} + \sum_i \sum_j \frac{Q_{ij}}{4} \tag{29}$$

Substituting σ_i^z in place of s_i. Where σ_i^z is the Pauli z gate operator in i^th qubit. With this quantum mechanical description of H_{ISING} is complete.

X. EXPERIMENTAL DETAILS

In the present work, variational quantum eigensolver (VQE) as implemented in IBM's quantum software development kit, the Qiskit is used. The problem formulation and approach toward the solution within Qiskit's VQE class is presented below.

1. Converting the problem into equivalent binary formulation(IH-QP) and solving the problem for 3 nodes and 2 vehicles.

2. Represent the problem as a quadratic program instance. Construct a QUBO optimization problem as an instance of the Quadratic Program. We get quantum Hamiltonian as (PauliSumOp(SparsePauliOp(['IIIIIZ', 'IIIIZI', 'IIIZII', 'IIZIII', 'IZIIII', 'ZIIIII', 'IIIIZZ', 'IIZIZI', 'IIZZII', 'IZIZII', 'ZIIIIZ', 'ZZIIII'], coeffs=[6101.65599199+0.j, 6129.9512975 +0.j, 6101.65599199+0.j, -21.44760604+0.j, 6129.9512975 +0.j, -21.44760604+0.j, 3066.15878995+0.j, 3066.15878995+0.j, 3066.15878995+0.j, 3066.15878995+0.j, 3066.15878995+0.j, 3066.15878995+0.j]), coeff=1.0), 30770.538852114867)

   where, Z represents Pauli z gate and I represent identity gate

3. Solve the problem using VQE and classical optimizer. We have used SPSA as a classical optimizer and real amplitude ansatz. Circuit details in VQE :
   Number of Qubits=6
   Number of R_y gates=24
   Number of C_x gates=15
   circuit depth=12
   Total number of gates = 39

4. Visualize the quantum and classical solution for 3 nodes and 2 vehicles.

XI. QUANTUM PROGRAM

The program is written in Python language using libraries of qiskit(IBM) for optimization.

```python
#For importing required package
from qiskit_optimization import QuadraticProgram
from qiskit_optimization.algorithms import MinimumEigenOptimizer
from qiskit.utils import algorithm_globals
from qiskit.algorithms.minimum_eigensolvers import SamplingVQE
from qiskit.algorithms.optimizers import SPSA
from qiskit.circuit.library import RealAmplitudes
from qiskit.primitives import Sampler
#Defining function to be used to evaluate cost from a binary representation of path
def bin_rep(num,veh,distance,xs=0):           #For binary representation
    M = np.max(distance) * 100   # M is large number introduced due to penality method
    # w represent the weight
    distance_vec = distance.reshape(num**2)
    w_l = [distance_vec[y] for y in range(num**2) if distance_vec[y] > 0]
    w = np.zeros(num * (num - 1))
    for k in range(len(w_l)):
        w[k] = w_l[k]

    # Here we define variable to be used
    nI = np.eye(num)
    n1I = np.ones([num - 1, num - 1])
    n1v = np.ones(num)
    n1v[0] = 0
    nvI = np.ones(num - 1)
    neg_n1v = np.ones(num) - n1v

    zt = np.zeros([num, num * (num - 1)])
    for k in range(num):
        count = k - 1
        for l in range(num * (num - 1)):
            if l // (n - 1) == k:
                count = k
            if l // (n - 1) != k and l % (n - 1) == count:
                zt[k][l] = 1.0
    vn = np.sum(zt[1:], axis=0)
    #Q coefficient of quadratic program
    qQ = M * (np.kron(nI, n1I) + np.dot(zt.T, zt))
    # g coefficient of quadratic program
    qg = (w- 2 * M * (np.kron(n1v, nvI) + vn.T)
            - 2 * M * veh * (np.kron(neg_n1v, nvI) + zt[0].T))
    # Here c is constant
    qc = 2 * M * (num - 1) + 2 * M * (veh**2)
    try:
        max(xs)
        # finding cost from binary representation of path
        fun = (lambda x: np.dot(np.around(x), np.dot(Q, np.around(x)))+ np.dot(g, np.around(x))+ c)
        co = fun(xs)
    except:
        co = 0
    return co,qQ,qg,qc
bcost,qQ, qg, qc = bin_rep(num,veh,distance,xs=z)
print("cost(binary):", bcost, "cost(classical):", cost)
#To create quadratic program
qp = QuadraticProgram()
for j in range(num * (num - 1)):
    qp.binary_var(str(j))
qp.objective.quadratic = qQ
qp.objective.linear = qg
qp.objective.constant = qc
#Giving Quadradatic program model  to VQE optimizer to find optimized cost
algorithm_globals.random_seed = 10456
solv_by_vqe = SamplingVQE(sampler=Sampler(), optimizer=SPSA(), ansatz=RealAmplitudes())
opti = MinimumEigenOptimizer(min_eigen_solver=solv_by_vqe)
r = opti.solve(qp)
# compute cost of the obtained result
l, _, _, _, = bin_rep(num,veh,distance,xs=r.x)
#q_sol is for quantum solution represent path and q_cost represent cost for optimized path
q_sol=r.x
q_cost=l
#To Visualise the optimized path
def visual(tit,xp, yp, x, C, num, veh):
    plt.figure()
    plt.scatter(xp, yp, s=180)
    for k in range(len(xp)):
        plt.annotate(k, (xp[k] + 0.18, yp[k]), size=15, color="r")
    plt.plot(xp[0], yp[0], "r*", ms=18)

    plt.grid()

    for k in range(0, num**2):

        if x[k] > 0:
            kx = k // n
            ky = k % n
            plt.arrow(xp[kx],yp[kx],
                xp[ky] - xp[kx],
                yp[ky] - yp[kx],
                length_includes_head=True,
                head_width=0.23,)

    plt.title(tit + " cost = " + str(int(C * 100.00) / 100.00))
    plt.show()
#visualizing the solution
x_quantum = np.zeros(num**2)
kk = 0
for ll in range(n**2):
    if ll // num != ll % num:
        x_quantum[ll] = q_sol[kk]
        kk += 1

# Visualization of Quantum Solution
visual("Quantum",xp, yp, x_quantum, q_cost, num, veh)

# and visualize the classical for comparison
if x is not None:
    visual("Classical",xp, yp, x, cost, num, veh )
```

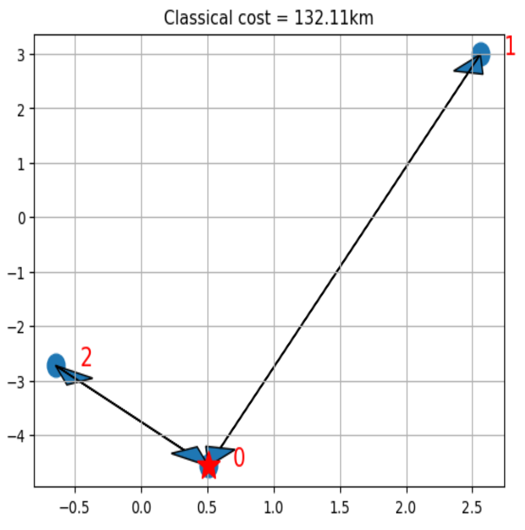

Figure 7: 1 and 2 are clients and 0 is depot

XII. RESULT: VISUALIZING THE QUANTUM AND CLASSICAL SOLUTION PATH FOR MINIMUM COST

1. We get the Quantum optimized result for 3 nodes and 2 vehicles as shown in fig 8.

2. We get the classical optimized result for 3 nodes and 2 vehicles as shown in fig 7

3. We can see classical and quantum solutions are coinciding.

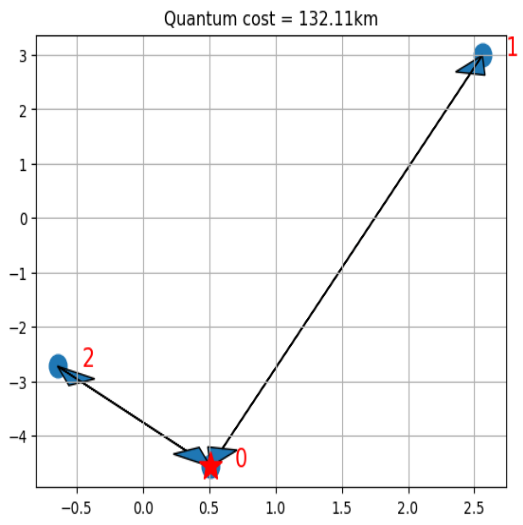

Figure 8: 1 and 2 are clients and 0 is depot

XIII. CONCLUSION

Quantum information and computing is a fast-emerging field of research. Quantum algorithms which are developed based on the concepts inherent to quantum mechanics such as the superposition of states, entanglement, interference, and tunneling. On the other hand, the problem of vehicle routing has been addressed which is central for the logistics and transportation industry. The problem is combinatorial in nature and scales exponentially with an increase in the no. of associated variables. With the emergence of quantum computing, in this NISQ era, variational quantum algorithms are expected to solve this problem in polynomial time. With this motivation, in this work, an attempt has been made to find the optimal path using variational quantum eigensolver (VQE). The results obtained using VQE (cost=132.11 Km) are compared with the classical optimization technique and are found to be in good agreement with the classical values (Cost=132.11 Km). In the future, from the quantum computing perspective, would like to solve the VRP using quantum annealing-based approach and compare the results with the classical and VQE values.

XIV. ACKNOWLEDGMENTS

I would like to express my sincere gratitude and appreciation to my guide Dr.Raghav and co-guide Dr.C.S. Yadav for their invaluable guidance and support throughout the preparation of this report. Their expertise, insights, and advice have greatly contributed to the quality and success of this work.

[1] C. E. Miller, E. W. Tucker, and R. A. Zemlin (1960). “Integer Programming Formulations and Travelling Salesman Problems”. J. ACM. 7: 326–329.doi:10.1145/321043.321046.

[2] D. L. Applegate, R. M. Bixby, V. Chvátal, and W. J. Cook (2006). The Traveling Salesman Problem. Princeton University Press, ISBN 978-0-691-12993-8.

[3] IBM Documentation on Vehicle Routing Problem: https://qiskit.org/documentation/optimization/tutorials/07_examples_vehicle_routing.html

[4] Lecture series on Advanced Operations Research by Prof. G.Srinivasan, Department of Management Studies, IIT Madras.https://www.youtube.com/watch?v=-cLsEHP0qt0

[5] User manual of cplex of IBM: https://www.ibm.com/docs/en/icos/12.8.0.0?topic=cplex-users-manual

[6] Documentation on Max-cut and Travelling salesman problem by IBM: https://qiskit.org/documentation/optimization/tutorials/06_examples_max_cut_and_tsp.html

[7] Python for Everybody book by Charles Severance: https://www.py4e.com/

[8] IBM ILOG CPLEX PYTHON API.https://courses.ie.bilkent.edu.tr/ie400/wp-content/uploads/sites/8/2021/12/IBM-ILOG-CPLEX-PYTHON-API.pdf

[9] Yin Sun et al. Adiabatic Quantum Simulation Using Trotterization. 2018. eprint: arXiv:1805.11568.

[10] D. Aharonov, W. V. Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev, Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation. 45th Annual IEEE

Symposium on Foundations of Computer Science (2004).

[11] Quantum annealing using d wave : https://docs.dwavesys.com/docs/latest/c_gs_2.html

[12] McClean, J. R., Romero, J., Babbush, R., Aspuru-Guzik, A. (2016). The theory of variational hybrid quantum-classical algorithms. New Journal of Physics, 18(2), 023023.

[13] A. Lucas, Ising formulations of many NP problems, Front. Phys. 2, 5 (2014).

[14] D-Wave Systems: Difference between BQM, Ising,and QUBO problems? Retrieved December 16, 2019, from https://support.dwavesys.com/hc/enus/community/posts/ 360017439853-Difference-between-BQM-Ising-and-QUBO-problems

[15] E. Farhi, J. Goldstone, and S. Gutmann, A Quantum Approximate Optimization Algorithm, arXiv:1411.4028 [quant-ph] (2014).

[16] T. Albash, and D. Lidar, Adiabatic Quantum Computing, Rev. Mod. Phys. 90, 015002