Solving NP-Problems Using Quantum Optimization Algorithms

Chitresh Bhaskar Chaudhari

chitresh@iitk.ac.in

Abstract. NumPyMinimumEigensolver, QAOA and VQE optimisers of Qiskit framework are used to get an approximate solution to NP-problems. Results of mentioned optimisers are compared with the classically obtained result. I found out that NumPyMinimumEigensolver finds the exact solution as the classical method and also take significantly less time than the classical algorithm. Among QAOA and VQE, QAOA gives the more accurate and closer solution. Also, time taken by QAOA to optimise is significantly lesser than the classical counterpart. On the other hand, VQE performs well but not as better as QAOA and NumPyMinimumEigensolver. VQE also took more time to optimise.

1 Introduction

Quantum computing is an emerging branch of modern-day computer science having a wide range of applications. Quantum computers are believed to be capable of solving computation problems which are nearly impossible to solve and practically take an infinite amount of time on classical computers. Quantum Computers can be used for solving some challenging graph problems in mathematics.

In this project, I implemented some chosen NP-Complete problems and reduced those problems to search and optimisation problems. I implemented this reduced problem in python code and solved it using quantum optimisation algorithms. I solved the selected problems using NumPyMinimumEigensolver[7], QAOA[8] and VQE[13] packages of qiskit. Qiskit [1] is the novel framework developed by IBM research for quantum computing.

Before diving into project it is necessary to know some basic information about quantum computing, Qiskit and graph optimisation problems.

1.1 Quantum Computing

Quantum computing is becoming more popular in recent days dues to its ability to solve problems which are much difficult to solve using classical computing. It is believed that if quantum algorithms run on quantum computers, their processing speeds improve exponentially compared to their classical counterparts.

The basic building block of quantum computing is qubit which can be considered as the counterpart of bit in classical computing [11]. A qubit can be in superpositions of states 0 and 1. While a classical bit is represented by an electrical voltage value or a wire current value, a quantum bit can take any linear combination of the two basic states 0 and 1 called a quantum state(Ψ) of the circuit. This property of superposition has many applications in computing and is considered as one of the reasons for the superiority of quantum algorithms over the classical one. Basis states 0 and 1 f qubit are denoted as $|0\rangle$ and $|1\rangle$. Any quantum state(Ψ) can be denoted as a superposition of these two states with complex coefficients a and b:

$$|\Psi\rangle = a\,|0\rangle + b\,|1\rangle\,, |a|^2 + |b|^2 = 1$$

The quantum computation process begins with a system set in a specific quantum state, which is then converted into a superposition of multiple basis states. Unitary transformations are performed on the quantum state according to the required operations of the algorithm. Finally, measurement is carried out, resulting in the qubits collapsing into classical bits[3]. Another important phenomenon of quantum computing is quantum entanglement which creates a bond between particles, even if they are separated, which results in an effect that the change of the state of one particle affects the other.

1.2 Qiskit

Qiskit[1] is an open-source framework developed by IBM research for quantum computation. It's development started at the beginning of 2017, but most of the significant additions are added after 2018. It provides different algorithms, simulators and noise models. It consists of 4 main components Qiskit Aer, Qiskit Terra, Qiskit Ignis and Qiskit Aqua. Aer contains mainly simulators and emulators.

Aqua[8] is the most useful component of qiskit and is written in the python programming language. It delivers high-level mechanisms to develop and build quantum algorithms. It focuses on finding the solution of real-life problems in the field of optimisation, chemistry, AI and finance using quantum computing. It provides some ready to use algorithms such as QAOA, VQE, QFT and Grover's algorithm. It also has some sample examples showing the evaluation of optimisation problems using VQE. I used some of these ready to use algorithms for solving selected graph optimisation problems.

Qiskit is free and open to everyone. It is integrated well with IBM Quantum Experience, and users can run their programs on real quantum devices. Anyone with valid registration with IBM Q experience can run their quantum circuits on IBM's free to use quantum computers. The maximum amount of qubits in accessible quantum computers is 5. Qiskit also provides some simulators to simulate quantum circuit on your personal device without access to real quantum hardware. In this project, I used IBMQ-qasm Simulator, which allows us to carry out operation up to 32 qubits.

1.3 Search and Optimisation problems

Graph problems[2] are a generalised version of many real-life problems. Most of these graph problems can be reduced to search and optimisation problems. Algorithms can be developed

for the particular problem such that it searches optimal solution to the problem in a known set of possible solutions.

1.3.1 P problems

Some problems can be solved in a time which relates to input in a polynomial manner. They can be solved in polynomial time, i.e. time required is a polynomial function of input $O(n^2)$. These problems are generally referred to as **P**.

1.3.2 NP problems

Another class of problems is **NP**[6]. Nearly all of the search problems can be classified as NP problems. NP problems cannot be solved in polynomial time. The time required to solve NP problems increases exponentially with input. In NP problems, there are two types of problem **NP-Complete** and **NP-hard**. NP-hard problems can't be solved nor verified in polynomial time. In the case of NP-Complete problems, if the solution is known to us, we can verify it in polynomial time. In this project, I solved several NP-complete problems using quantum optimisation algorithms in qiskit framework and compared results obtained from each method. I also compared the time taken by each algorithm to find an optimised solution to the problem.

2 Problems and Approach

I have chosen the following search and optimisation problems for our product. These are generalised search and optimisation problems. We can reduce any search problems into these problems and evaluate them efficiently.

2.1 Max-Cut problem

Max-Cut is an NP-complete problem, with applications in network science, clustering and statistical physics[5]. The problem is basically dividing the given graph into two complementary sets such that their elements have a maximum number of edges between one set and its complementary set.

The formal definition of this problem is the following: Consider an n-node undirected graph $G = (V, E)$ where $|V| = n$ with edge weights $w_{ij} > 0, w_{ij} = w_{ji}$, for $(i, j) \in E$. A cut is defined as a partition of the original set V into two subsets. The cost function to be optimised is [12]

$$C(x) = \sum_{i,j} w_{ij} x_i (1 - xj)$$

In the above cost function, w_{ij} represents ij^{th} entry of adjacency matrix such that $w_{ij} = 1$ if there is edge between i^{th} and j^{th} vertices. By randomly assigning $x_i = 0$ or $x_i = 1$ to each node i I maximised above cost function and obtained cost for $n = 4$ to $n = 11$.

Maxcut problem can also be solved using quantum optimisation algorithms like QAOA and VQE. For the problem to be solved on a quantum computer, we need to convert cost

function in ising hamiltonian operator[4].

$$H = \sum_i w_{ij} Z_i + \sum_{i<j} w_{ij} Z_i Z j$$

Above Hamiltonian operator can be obtained directly using maxcut library in qiskit framework by giving adjacency matrix as an argument. One can obtain this manually using '.to_ising()'[10] on $C(x)$ created using QuadraticProgram package of qiskit.

2.2 Independent Set of Vertices

Independent set of vertices is an NP-Hard problem with applications in genetics and other many fields. The problem is basically finding the largest set of vertices such that no two vertices of the set have an edge in between them.

The formal definition is the following: Consider n-node graph $G = (V, E)$ where $|V| = n$. We have to find the largest set such that there is no edge between any two nodes in the set. Cost function need to be optimised is

$$C(x) = \sum_{n=0}^{n-1} x_i - M \sum_{i,j \in E} x_i . x_j$$

In the above cost function, $x_i = 1$ if the node is in the set and $x_i = 0$ if the node is not in the set. The first term gives a number of points in the set and the second term offers penalty when wrong vertices are marked in the set. I maximised the above cost function by iterating through all possible sets and got cost which is equal to the number of nodes in an independent set of vertices. I also solved this particular problem using optimisation algorithms which give an approximate solution. I converted the above cost function into qubo form using QuadraticProgram[9] package of qiskit. This created qubo is then optimised by quantum optimiser QAOA and VQE. I also optimised it using classical optimiser NumPyMinimumEigensolver. I recorded results for n=4 to n=20 and also recorded the time taken for solving the problem.

2.3 3-Coloring problem

Graph colouring problem is NP-complete problem. In the graph colouring problem, all the nodes are coloured in a way such that no two adjacent nodes have the same colour. It is proved that any planar graph can be coloured in this way using only three ways. Our problem is, can we colour randomly generated graph using only three colours such that neighbouring nodes are coloured using different colours.

The formal definition is the following: Consider n-node graph $G = (V, E)$ where $|V| = n$. I defined three variable for each node $x_{v,R}, x_{v,G}$ and $x_{v,B}$ such that $x_{v,i} = 1$ if v^{th} node is coloured by $i \in C$ and $C \equiv \{R, G, B\}$. Following function was need to be optimised[4].

$$H = \sum_v^n (1 - \sum_{i \in C} x_{v,i})^2 + \sum_{u,v \in E} \sum_{i \in C} x_{u,i} x_{v,i}$$

n	brute force		NumpyEigensolver		QAOA		VQE	
	Cost	time	Cost	time	Cost	time	Cost	time
4	2.8	0.0001995	2.8	0.0235217	2.8	0.053	2.8	0.082
5	4.0	0.0012054	4.0	0.0189568	4.0	0.07	4.0	0.126
6	6.2	0.0033866	6.2	0.0327173	6.2	0.062	6.0	0.174
7	7.8	0.0061838	7.8	0.0277352	7.8	0.205	7.8	0.204
8	11.6	0.0149591	11.6	0.041497	11.6	0.091	11.6	0.238
9	13.6	0.0385022	13.6	0.065017	13.6	0.101	13.2	0.304
10	15.8	0.1035177	15.8	0.039893	15.8	0.123	15.6	0.365
11	19.4	0.2411546	19.4	0.053471	19.4	0.142	19.0	0.445
12	24.6	0.92281586	24.6	0.15384	24.4	0.191	23.6	0.683
13	28.2	2.32965641	28.2	0.28271	27.6	0.24	27.2	0.912
14	30.6	4.83018355	30.6	0.28417	29.2	0.463	30.0	1.523
15	37.4	10.4718661	37.4	0.43806	36.8	0.529	36.0	1.902

Figure 1: Data Obtained for Max-cut Problem

In the above equation, first summation term make sure that only one of the $x_{v,R}, x_{v,G}$ and $x_{v,B}$ is one that is node is coloured only by one colour. The second summation term gives penalty when adjacent nodes are coloured using the same colour. I minimised the objective function by randomly assigning colours to each node. If the solution is possible, that is we can colour given graph using only three colours objective function minimises to zero otherwise gives non-zero value. I minimised this function using classical optimiser NumPyMinimumEigensolver and also with quantum optimiser QAOA & VQE. I generated random graphs for n=4 to n=10 and recorded obtained results and time is taken to solve the problem.

3 Experimental Results

3.1 Max-Cut problem

I maximised the above-mentioned cost function using a brute force algorithm. I obtained hamiltonian operator from randomly generated adjacency matrix using 'max_cut.get_operator()' function in max_cut library of qiskit[5]. I passed this hamiltonian operator into QAOA and VQE for optimisation and recorded optimal value. I also created minimum eigensolver using classical optimiser NumPyMinimumEigensolver and optimised the quadratic program of the cost function created by QuadraticProgram. I created a random adjacency matrix and solved the test case using all four methods. I repeated this method five times for each n and took an average of them as the final result. Data obtained is as shown in figure 1:

3.2 Independent set of vertices

For this problem, I minimised the created QuadraticProgram(qubo) using NumPyMinimumEigensolver, QAOA and VQE. I also implemented a brute force algorithm to maximise above qubo by taking each possible combination. I created a random edge matrix and solved

n	brute force		NumpyEigensolver		QAOA		VQE	
	Cost	time	Cost	time	Cost	time	Cost	time
4	2.6	0.0006585	2.6	0.0483877	2.6	0.027	2.6	0.098
5	2.2	0.0019631	2.2	0.0444595	2.2	0.021	2.2	0.114
6	3.0	0.0027653	3.0	0.0483246	3.0	0.025	3.0	0.138
7	3.0	0.0047828	3.0	0.0442991	3.0	0.028	2.8	0.182
8	3.4	0.0139773	3.4	0.0809985	3.4	0.031	3.4	0.223
9	3.4	0.0279306	3.4	0.0919490	3.4	0.04	3.0	0.276
10	4.2	0.0629500	4.2	0.0488839	4.2	0.042	3.8	0.356
11	3.6	0.1002798	3.6	0.0394656	3.6	0.043	3.0	0.462
12	4.8	1.1908369	4.8	0.2649870	4.4	0.039	3.8	0.571
13	4.2	1.6870797	4.2	0.3020028	3.4	0.067	3.6	0.787
14	5.0	1.2807737	5.0	0.1072928	4.0	0.111	3.4	1.367

Figure 2: Data Obtained for Independent set of vertices problem

qubo formed by that matrix using all these four methods. I repeated the above step of generating random edge matrix five times for each n and took an average of all five results as the final result. Data obtained are shown in figure 2:

3.3 3-Colouring problem

For this particular problem, I first implemented a brute force algorithm which takes all possible combination and checks them against randomly generated adjacency matrix. I generated the random graph and checked whether we could colour that graph or not by minimising above mentioned cost function. If it minimises to zero, then colouring is possible. I minimised it using NumPyMinimumEigensolver, QAOA and VQE. I repeated above step 5 times for each n and finally recorded an average of then as the final result. Plots of data obtained are shown in the following table:

n	brute force		NumpyEigensolver		QAOA		VQE	
	Cost	time	Cost	time	Cost	time	Cost	time
4	5	0.0.000536	5	0.0416946	5	0.05	5	0.728
5	5	0.00060696	5	0.1300543	4	0.17	4	1.435
6	5	0.0057885	4	1.1360928	2	0.277	1	1.925

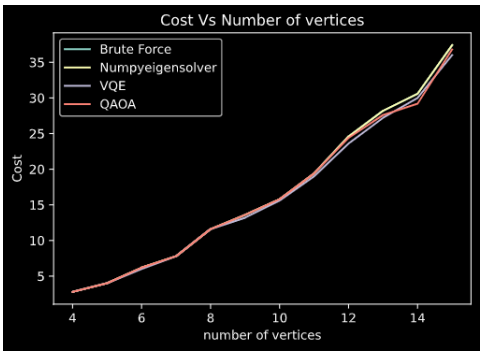
Figure 3: results obtained for 3 colouring problem

4 Observations

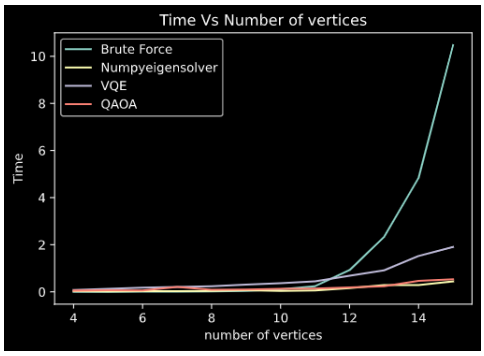
4.1 Max-cut

Plots of experimental results of max-cut problems are shown in figure 4. It is clear that classical optimiser NumPyMinimumEigensolver gives exactly the same cost as a brute force

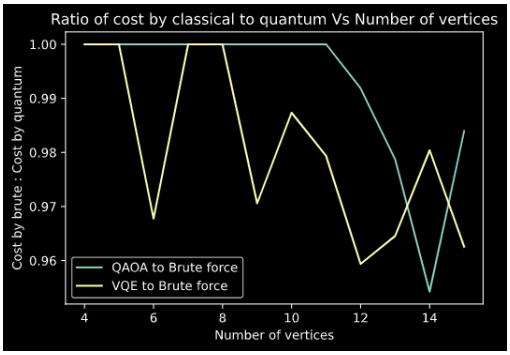
algorithm. So, we can say conclusively that NumPyMinimumEigensolver can be used to solve NP problems in the place of brute force. Also, one plus point for NumPyMinimumEigensolver is that it takes much lesser time in comparison to the brute force algorithm. From the figure, we clearly see that time of execution increases exponentially by brute force algorithm whereas, for others, it's non-exponential growth. If we plot the ratio of costs by a quantum method and classical method, we get fig 1c. We can observe that QAOA performs at par with classical that means give an exact same solution as classical methods until $n = 11$. After $n = 11$ its little less performing. On the other hand, VQE is not giving that much-desired results as compared to QAOA, but still, it's good. As you can see the worst ratio is nearly 0.95, which is not that bad and is acceptable.



(a) Cost Vs Number of Vertices



(b) Time Vs Number of Vertices

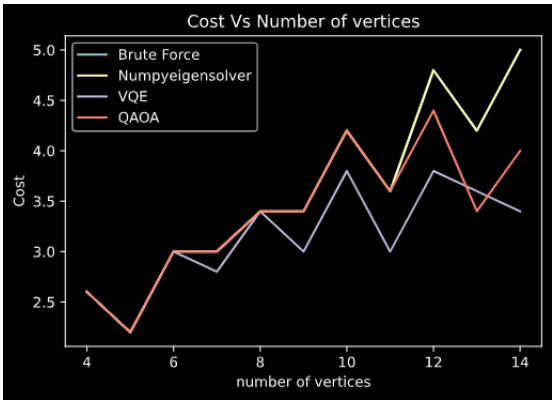


(c) Ratio of Cost by quantum optimiser to classical result vs number of points

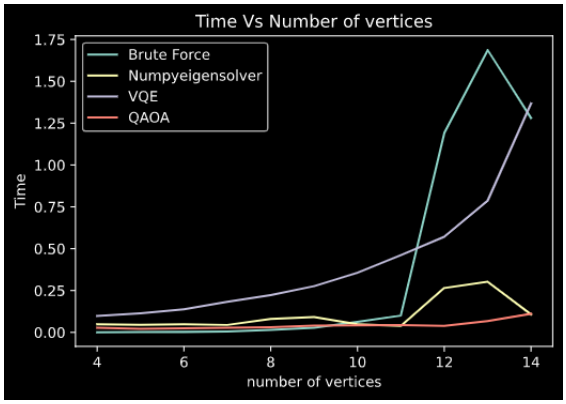
Figure 4: Graphical representation of results for Max-cut problem

4.2 Independent Set of Vertices

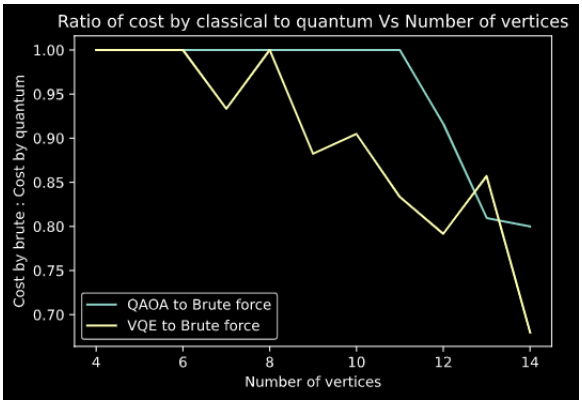
Plots of results for this problem are given in figure 5. In this case, also NumPyMinimumEigensolver is giving the same cost as a brute force algorithm and executing is lesser time. The pattern for a time of execution remains nearly the same for brute force, NumPyMinimumEigensolver and QAOA. Whereas, VQE seems to increase more than QAOA with increasing n. If we observe the accuracy of QAOA and VQE, again QAOA performs better than VQE. In this case, also QAOA gives an exact solution to $n = 11$. VQE is not as much accurate as QAOA but acceptable. The lowest ratio I get is near 0.65 for VQE whereas its 0.80 for QAOA.



(a) Cost Vs Number of Vertices



(b) Time Vs Number of Vertices



(c) Ratio of Cost by quantum optimiser to classical result vs number of points

Figure 5: Graphical representation of results for Independent set of vertices problem

4.3 3-colouring

Due to limited resources and computational power, I recorded data for only $n = 4, 5, 6$. Here Cost represents the number of times a graph is coloured in 3 colours out of 5. As we can see from the table, we get the same result by brute force and NumPyMinimumEigensolver except $n = 6$. In this case, NumPyMinimumEigensolver takes little more time than brute force. QAOA and VQE performed nearly the same for all n. Like former examples here also QAOA takes lesser time than VQE to execute.

5 Conclusion

From the above all observations, we conclude that it is possible to solve NP-problems using Quantum optimiser. NumPyMinimumEigensolver can be used as a substitute for classical brute force algorithm which gives exact value and also takes lesser time. Quantum optimisers also give satisfactory results. Among both quantum optimisers, QAOA is better than VQE in both accuracy and time of execution. So, QAOA should be prefered. If we see problem wise, max-cut is solved most efficiently by quantum optimisers, and 3-colouring problem is solved with the lowest efficiency.

Acknowledgement

I want to express genuine gratitude towards my mentor Mr.Diraj Madan for giving me this excellent opportunity and motivation for this project, and also for guiding me throughout the project. I would also like to thanks IBM for open-source qiskit framework and IBMQ Experience.

References

[1] Héctor Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110.

[2] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. 1st ed. USA: McGraw-Hill, Inc., 2006. ISBN: 0073523402.

[3] Y. H. Lee, M. Khalil-Hani, and M. N. Marsono. *An FPGA-Based Quantum Computing Emulation Framework Based on Serial-Parallel Architecture*. en. Research Article. Apr. 2016. DOI: https://doi.org/10.1155/2016/5718124. URL: https://www.hindawi.com/journals/ijrc/2016/5718124/ (visited on 10/23/2020).

[4] Andrew Lucas. "Ising formulations of many NP problems". In: *Frontiers in Physics* 2 (2014), p. 5. ISSN: 2296-424X. DOI: 10.3389/fphy.2014.00005. URL: https://www.frontiersin.org/article/10.3389/fphy.2014.00005.

[5] *Max-Cut and Traveling Salesman Problem — Qiskit 0.23.0 documentation*. URL: https://qiskit.org/documentation/tutorials/optimization/6_examples_max_cut_and_tsp.html.

[6] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[7] *NumPyMinimumEigensolver — Qiskit 0.23.0 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.aqua.algorithms.NumPyMinimumEigensolver.html.

[8] *QAOA — Qiskit 0.23.0 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.aqua.algorithms.QAOA.html.

[9] *Quadratic Programs — Qiskit 0.23.0 documentation*. URL: https://qiskit.org/documentation/tutorials/optimization/1_quadratic_program.html.

[10] *QuadraticProgram.to_ising — Qiskit 0.23.0 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.optimization.QuadraticProgram.to_ising.html.

[11] Ramya Shankar and Ramya ShankarA cheerful. *What is Quantum Computing - techiechief*. Sept. 2020. URL: https://techiechief.com/what-is-quantum-computing/.

[12] *Solving combinatorial optimization problems using QAOA*. en. URL: https://community.qiskit.org/textbook/ch-applications/qaoa.html.

[13] *VQE — Qiskit 0.23.0 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.aqua.algorithms.VQE.html.