https://www.sworldjournal.com/index.php/swj/article/view/swj11-01-072
DOI: 10.30888/2663-5712.2022-11-01-072

UDC 004.621.8

IMPLEMENTATION OF THE QUANTUM GENETIC ALGORITHM IN THE ENVIRONMENT PYTHON

Skakalina E. V. / Скакалина Е.В.
c.t.s., as.prof. к.т.н., доц.
ORCID: 0000-0002-6441-3467
National University "Yuri Kondratyuk Poltava Politechnics"
Pershotravnevyi ave. 24, Poltava, UA-36011, Ukraine
Національний університет «Полтавська політехніка імені Юрія Кондратюка», Полтава, проспект Першотравневий 24, 36011

***Abstract**. In modern conditions of total digitalization of the entire world economy, the development of quantum algorithms is one of the topical areas of research. The advantage of quantum algorithms is to reduce the problem solving time by parallelizing operations by generating entangled quantum states and then using them. This advantage (quantum acceleration) is the most advantageous in solving the problem of modeling the dynamics of complex systems and enumerating mathematical problems. The island model of genetic algorithms gives a significant advantage in determining the computational load due to parallelization at the level of the main algorithm. Therefore, research into the possibilities of a quantum genetic algorithm is essential for solving many complex problems.*

***Keywords :** quantum programming, genetic algorithms, qubit, simulator, Amadar operator*

Introduction .

The first quantum revolution took place in the second half of the 20th century and led to the emergence of lasers, transistors, nuclear weapons, and subsequently mobile phones and the Internet. The technologies of the first quantum revolution are used in computers, mobile phones, tablets, digital cameras, communication systems, LED lamps, MRI scanners, scanning tunneling microscopes, etc. The volume of the market for the corresponding products in the world is $3 trillion a year. At the same time, "Moore's law", according to one of the statements of which, the performance of processors should double every 18 months, no longer works. Since the end of the 20th century, the world has been on the threshold of the second quantum revolution. In the first quantum revolution, technologists and devices were built on the control of collective quantum phenomena. In the second quantum revolution, technologies will be built on the ability to control complex quantum systems at the level of individual particles, such as atoms and photons. Technologies based precisely on such a high level of control over individual quantum objects are commonly referred to as quantum technologies.

Quantum computers are devices that can solve any algorithmic problem using qubits - that is, they are quantum analogues of universal classical computers. The computational capabilities of a quantum computer are determined by two main characteristics: the number of qubits and their quality (the level of errors in performing elementary operations).

The functioning of a quantum computer is extremely difficult to model using classical algorithms. Simulating a quantum computer with only 50 qubits already requires enormous computing resources, which are at the limit of the capabilities of existing classical supercomputers. Accordingly, a supercomputer with more than 50 qubits can already demonstrate "quantum superiority" - the ability to solve those problems that cannot be solved using modern classical supercomputers.

It is predicted that in a number of tasks a quantum computer will be able to give a multiple acceleration compared to existing supercomputer technologies. Examples are cybersecurity, optimization (including in logistics and finance), artificial intelligence (training and acceleration of neural networks), database search, big data processing and complex systems modeling (materials and chemical reactions).

But to increase the computing power of a quantum computer, it is not enough to increase the number of qubits; the degree of control over them is also important. An uncontrolled impact on a quantum system from the environment can lead to the phenomenon of decoherence, which, in turn, leads to errors in the course of calculations.

Technologies for quantum programming

Many software packages have been written to simulate quantum computers, and they are connected in such a way that they can be used with Python. This programming language remains just as accessible and easy to use even for such complex and ambiguous tasks. Special modules such as qiskit, cirq, ocean are installed for quantum programming. Qiskit is IBM's open source software development kit for working with quantum computers [1]. This module simplifies the development of quantum applications by offering the resources needed to interact with quantum systems and simulators. Suitable for end users without experience in quantum development. With four Qiskit packages - Aqua, Terra, Ignis, and Aer, you can work with both simple and complex algorithms. You can use this add-on in two ways: either run it locally or in the cloud without installing any software using IBM Computer Lab. Cirq is Google's Python-based library of software for writing, managing, and optimizing quantum schemas and then running them on quantum systems or simulators. Beginning in 2021, Cirq is working on offering access to one of Google's real quantum computers. Meanwhile, it is still possible to create algorithms and schemes and test them on a quantum simulator. Google offers a variety of tutorials to help beginners move from zero to expert level of quantum simulation with Cirq. Ocean is a set of tools provided by D-Wave to solve complex problems using quantum computers. The software performs the calculations required to convert random problems into a form that can be solved by a quantum computer[2].

To develop my own algorithm, I chose the qiskit module. More about him. It works in two modes. The first is, like everything, a regular local emulator. The second is to run the program on a real quantum computer IBM Q.

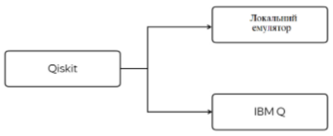


Figure 1 - Modes of operation of the selected module

This computer now looks like a barrel that maintains an extremely low temperature of about three microns. And IBM provides the ability to use cloud technology to connect and run your code directly on this machine.

There are several computer installations for sharing. One of them is 5-qubit, there are 15-qubit, 16-qubit, we even have 20. The qubit for this computer is just a vector in two-dimensional space, which has a certain basis. An infinite amount of information can be stored in one qubit through a vector form.

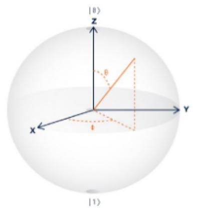


Figure 2 - Qubit as a vector represented in the three-dimensional sphere

Quantum simulator operations

The first thing we are used to in classical imperative programming is to calculate the value of a variable, for example, in Python. We want to read what state the qubit is in. But we never know the exact values of α and β. If we try to look at the qubit, read, we get either 0 or 1 with the corresponding probabilities. Probabilities are simply projections on the corresponding basis vectors. The second thing we can do is assign a variable to another, but in the quantum world we can't do that. That is, there is a qubit that we cannot read, we cannot appropriate. What can you do? Qubit is a vector[3]. The vector can be taken and rotated around the sphere. To rotate, you can invent a matrix that makes this rotation. All operations on qubits are matrices. They are called unitary. The first thing we are used to in classical imperative programming is to calculate the value of a variable, for example, in Python. We want to calculate the state of the qubit. But we never know the exact values of α and β. If we try to look at the qubit, calculate, we get either 0 or 1 with the corresponding probabilities. Probabilities are simply projections on the corresponding basis vectors. The second thing we can do is assign a variable value to another, but in the quantum world we can't do that. That is, there is a qubit, which we can not calculate, we can not assign it a certain value. What can you do? In the most general sense, a qubit is a vector. The vector can be taken and rotated around the sphere. To rotate, you can invent a matrix that makes this rotation. All operations on qubits are matrices. They are called unitary.

$$UU^{\dagger} = I$$

Figure 3 - Complex-coupled matrix

This icon means a transposed and complex-connected matrix. Its property is that for any operation there is a reverse. That is, no matter how we move the vector, we can always return it to its original position.

There are, of course, those operations to which we are accustomed in classical programming. For example, the negation operator (NOT):

$$|0\rangle \to |1\rangle \qquad\qquad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$|1\rangle \to |0\rangle$$

Figure 4 - Operator NOT

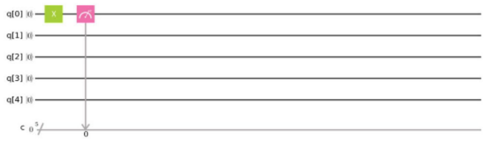


Figure 5 - Application of the NOT operator on the simulator circuit

In quantum programming, circuits are usually used[4]. On the visual simulator from IBM, you can track the work of operators through the interaction with qubits. The scheme usually looks like we can place our operators on five horizontal lines. Five lines means five qubits. Operators are denoted as blocks. The Quantum Composer library contains many different classes of gates: single qubit gates, such as yellow idle operation; green class of Pauli operators representing bit-flips (X, equivalent to classical NE); somersault phase (Z); and a combined bit-flip and phase-flip (Y). Clifford operations are also proposed, which are a blue class of gates such as H, S and S’ gates for generating quantum superpositions and complex quantum phases, as well as the previously mentioned two-qubit C-NOT funnels. The red gate is two-phase, which is important to ensure quantum calculation of its power. To measure the state of any qubit, a standard measurement operation is used, which is a simple Z-projection assigned to a classical bit in the classical bit register. The gray barrier allows to separate parts of the scheme from a visual line; when using optimization, it stops optimizing tools across the barrier. Quantum algorithm (scheme) begins with the preparation of qubits in well-defined states, here the ground state, | 0, then perform a series of one- and two-qubit gates in time with subsequent measurement of qubits.

The same blocks can be written in Python, using the qiskit library. To begin with, it is necessary to initialize the quantum register from one qubit, and the classical register. A classical register is required to record the measurement result. That is, transformations are made with the quantum register, the result is classical - 1 or 0.

Operators such as AND or OR do not work in quantum programming because they are not completely reversible. In other words, getting "0" in operation "and" we will never be able to find out what the initial values were. However, another CNOT operator appears:

```
if q[0]:
    q[1] = not q[1]
```

$$CNOT|00\rangle \to |00\rangle$$
$$CNOT|01\rangle \to |01\rangle$$
$$CNOT|10\rangle \to |11\rangle$$
$$CNOT|11\rangle \to |10\rangle$$

Figure 6 - CNOT operator

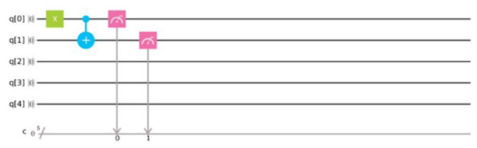


Figure 7 - Application of the CNOT operator on the simulator circuit

To describe the next operator, I want to draw an analogy with the game "eagle or tail", which involves the user and the computer. Two players take turns to toss an imaginary coin, but the result is not visible immediately. For a computer, this is usually a sample of 0 or 1. Players have two attempts, after which the computer gives the result - who won, and the winner is the one who has more "eagles". If you play with a normal computer, without which we can no longer imagine our lives, then both players have equal chances - 50 to 50. In practice, it turns out - about 50% of the winnings. 372 people took part in the same game, but their opponent was a quantum computer. The result was quite expected - 97% of computer winnings. By the way, those 3% multiple losses are due to operating system failures. Here's how it works: A normal computer takes each side of a coin for one bit, ie 0 or 1, and its chip is set to the position of the coin at a given time. The quantum computer works in a completely different way. Qubit, more dynamic, it is characterized by a binary state. It can exist in the superposition, or in another - situations of overlap 0 and 1. This state can be called not just indefinite, but rather a state in the range from 0 to 1. In fact, with some probability it may be 0, and with some - 1 This probability can be 80/20 or 54/46, 70/30, etc. The key idea is that we have to abandon the exact values of 0 and 1 and allow some ambiguity. Thus, during the game, the quantum computer creates a dynamic combination of eagle and tail (0 and 1), so regardless of whether the player wins the winning combination or not, the superposition remains unchanged. A quantum computer can separate 0 or 1 at the very last moment, when calculating the result, for example, and easily get a 100% winning combination, so its opponent is doomed to lose. This situation can be simulated using the Hadamard operator[5]:

$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Figure 8 - Hadamard operator

This operator in the quantum simulator scheme is denoted by block H:

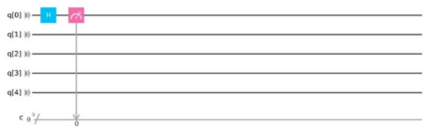


Figure 9 - Application of the Hadamard operator on the simulator circuit

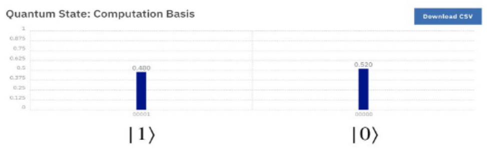


Figure 10 - The result of the work of the operator Hadamard

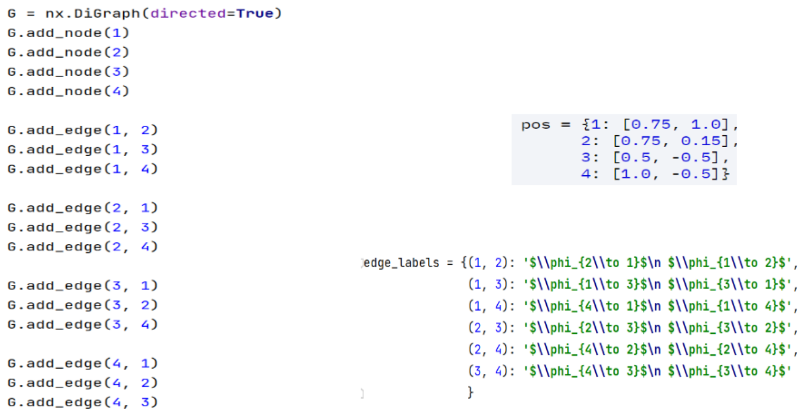
Algorithm description and implementation on QASM simulator

Quantum genetic algorithm is a GA that combines quantum operators (rotation, measurement, quantum chromosomes) with classical genetic operators (crossover and mutation). It can be used for educational and research purposes.

Consider the problem in terms of graphs:

- ✓ cities are represented as vertices and cost / path as edges;
- ✓ the specified distances will be coded in the form of phases;
- ✓ each city has a certain path to another, each path has a value, ie there is a task to minimize losses;
- ✓ unitary operators are built, the vectors of which are computational basis states, and the eigenvalues are different combinations of these phases;
- ✓ then we apply a phase estimation algorithm to certain eigenstates, which gives us possible common distances for all routes;
- ✓ After obtaining distances, we can search for this information using a quantum search algorithm to find the minimum to find the shortest possible distance, as well as the route traveled. This gives us a quadratic acceleration compared to the classical method of brute force for a large number of cities.

Consider the case when the number of nodes in our graph is 4 (n = 4). To do this, use the network module for and *matplotlib* [6]. In Python we describe the vertices and sides of the graph, add the phase designations on each edge and visualize:



```
G = nx.DiGraph(directed=True)
G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)

G.add_edge(1, 2)
G.add_edge(1, 3)
G.add_edge(1, 4)

G.add_edge(2, 1)
G.add_edge(2, 3)
G.add_edge(2, 4)

G.add_edge(3, 1)
G.add_edge(3, 2)
G.add_edge(3, 4)

G.add_edge(4, 1)
G.add_edge(4, 2)
G.add_edge(4, 3)
```

```
pos = {1: [0.75, 1.0],
       2: [0.75, 0.15],
       3: [0.5, -0.5],
       4: [1.0, -0.5]}
```

```
edge_labels = {(1, 2): '$\\phi_{2\\to 1}$\n $\\phi_{1\\to 2}$',
               (1, 3): '$\\phi_{1\\to 3}$\n $\\phi_{3\\to 1}$',
               (1, 4): '$\\phi_{4\\to 1}$\n $\\phi_{1\\to 4}$',
               (2, 3): '$\\phi_{2\\to 3}$\n $\\phi_{3\\to 2}$',
               (2, 4): '$\\phi_{4\\to 2}$\n $\\phi_{2\\to 4}$',
               (3, 4): '$\\phi_{4\\to 3}$\n $\\phi_{3\\to 4}$'
               }
```

Figure 11 - Code in Python to implement a graph on 4 vertices

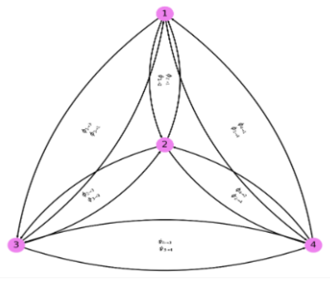
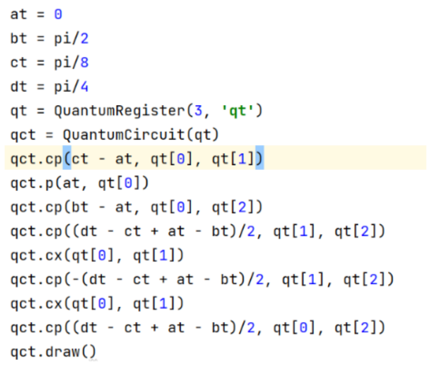


Figure 12 - The result of code execution: a graph with four vertices and a description of the phases between them

In Python we build a scheme of unitary venetians:



```
at = 0
bt = pi/2
ct = pi/8
dt = pi/4
qt = QuantumRegister(3, 'qt')
qct = QuantumCircuit(qt)
qct.cp(ct - at, qt[0], qt[1])
qct.p(at, qt[0])
qct.cp(bt - at, qt[0], qt[2])
qct.cp((dt - ct + at - bt)/2, qt[1], qt[2])
qct.cx(qt[0], qt[1])
qct.cp(-(dt - ct + at - bt)/2, qt[1], qt[2])
qct.cx(qt[0], qt[1])
qct.cp((dt - ct + at - bt)/2, qt[0], qt[2])
qct.draw()
```

Figure 13 - Programming of unitary gates on the scheme of quantum qubits

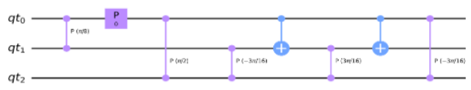


Figure 14 - Layout of unitary gates

Now we display the general scheme of the transport problem using the tools described above.

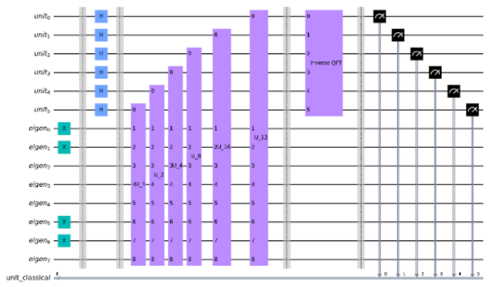


Figure 15 - Scheme of unitary and eigenstates, normalized phases. (All symbols used are described above)

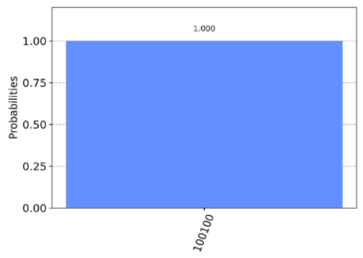


Figure 16 - Result on the QASM simulator

Conclusions

A quantum genetic algorithm has been developed to solve the problem of constructing the optimal path as an example of practical application. Genetic algorithms in various development environments such as Matlab and PyCharm are also presented for reliable testing. The proposed quantum genetic algorithms have a divergent structure: qubit and cutrite representation of the algorithm. The research of the subject area is carried out, the quantum genetic algorithm as a symbiosis of the genetic algorithm and quantum mechanics is designed. A thorough analysis of software analogues and research in the field of quantum genetic algorithms, methods of constructing algorithms and their features, testing on a simulator using test functions, proved as a result of computational verification that the developed algorithms are effective and meet the requirements of tasks, proven optimality of their practical application.

References :

[1] IBM and Ponemon Institute research [Electronic resource] // IBM - Resource access mode: https://www-03.ibm.com/press/ru/ru/pressrelease/50084.wss.

[2] P. W. Shor, "Quantum Computing," Documenta Mathematica, vol. Extra Volume ICM, pp. 467- 1486, 1998, http://east.camel.math.ca/ EMIS/ journals/ DMJDMV/ xvol-icm/ 001 Shor.MAN.htm1.

[3] P. W. Shor, "Algorithms for Quanhim Computation: Discrete Logarithms and Factoring," in Proceedings of the 35th Annual Syrnposiunz on Foundations of Computer Science, pp. 124- 134, 1994.

[4] Li J.-P., Balazs M.E., Parks G.T., Clarkson P. J., A Species Conserving Genetic Algorithm for Multimodal Function Optimization, Evolutionary Computation, V. 10, N. 3, pp.207-234, 2002.

[5] Li J.-P., Balazs M.E., Parks G.T., Clarkson P. J., A Species Conserving Genetic Algorithm for Multimodal Function Optimization, Evolutionary Computation, V. 10, N. 3, pp.207-234, 2002.

[6] R. Nowotniak and J. Kucharski, "Higher-order quantuminspired genetic algorithms," in Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014, pp. 465–470, Poland, September 2014.

Анотація . *У сучасних умовах тотальної диджиталізації всього світового господарства розробка квантових алгоритмів є однією із актуальних напрямів досліджень.*

Перевага квантових алгоритмів полягає у зниженні часу вирішення задачі за рахунок розпаралелювання операцій шляхом генерування заплутаних квантових станів та їх подальшого використання. Вказана перевага (квантове прискорення) є найбільш виграшною при вирішенні задачі моделювання динаміки складних систем та перебірних математичних завдань. Острівна модель генетичних алгоритмів дає значну перевагу щодо обчислювальної навантаження з допомогою розпаралелювання на рівні основного алгоритму. Тому дослідження можливостей квантового генетичного алгоритму мають важливе значення при вирішенні багатьох складних задач.

Ключові слова: *квантове програмування, генетичні алгоритми, кубіт, симулятор, оператор Амадара .*

Стаття відправлена: 23.01.2022 г.