Quantum Approximate Optimization Algorithm for the Max-Cut Problem: JavaScript Programming Language Implementation

Dmytro Sapozhnyk
Department of Software Engendering, Zhytomyr Polytechnic State University, Chudnivska Str. 103, 10005 Zhytomyr, Ukraine
phd121221_sdo@student.ztu.edu.ua

Keywords: Quantum Approximate Optimization Algorithm (QAOA), Max-Cut, JavaScript, Quantum Algorithms.

Abstract: In this paper, we present the implementation of the Quantum Approximate Optimization Algorithm (QAOA) for the Max-Cut problem using the JavaScript programming language. The Max-Cut issue, which involves partitioning the vertices of a graph into two subsets such that the number of edges between the subsets is maximized, is a well-known NP-hard difficulty with numerous practical applications, including network design and resource allocation. The implementation of QAOA in JavaScript is a significant step towards integrating quantum computing with modern web technologies, thus broadening access to quantum algorithms among software developers. Quantum algorithm implementation leverages the principles of quantum mechanics, such as superposition and entanglement, to approximate solutions to combinatorial optimization issues. The quantum.js framework, developed in the context of this research, facilitates the construction and manipulation of quantum circuits in a web environment. The framework includes functions for building quantum circuits, optimizing the parameters of the QAOA algorithm, and visualizing the resulting quantum states. By enabling the execution of quantum algorithms in a web-based setting, this work demonstrates the potential for utilizing quantum computing capabilities within popular web development environments. The results highlight the efficiency of QAOA in providing approximate solutions to the Max-Cut, offering a promising alternative to classical optimization methods. Future work will focus on enhancing the framework by adding cloud-based quantum computing capabilities, expanding the documentation, incorporating additional quantum-hybrid algorithms, and improving the user interface of the associated web application.

1 INTRODUCTION

Quantum computing is a revolutionary technology based on the principles of quantum mechanics. The idea of quantum computing was first put forward by Paul Benioff and Richard Feynman, Benioff proposed a model of a computer based on quantum mechanics, Feynman, and motivated the use of computers to simulate quantum phenomena [1]. This technology promises significant improvements in various fields, such as security, finance, medicine, communications, and sciences. Quantum mechanics revolutionized the world by redefining Newtonian physics and our understanding of the universe. Its principles, such as entanglement, interference, and superposition, have become the basis of many fields, including quantum chemistry, quantum information theory, quantum cryptography, and quantum machine learning [2].

The advantages of quantum computing over classical computing include exponential acceleration of performance in some specific computing, these include all problems that are based on a complete enumeration of options. It has been theoretically and mathematically proven that they can significantly surpass classical computing systems. Google has experimentally confirmed this with its Sycamore processor [3]. Quantum devices, although in the early stages of development, are already showing great potential for solving complex issues.

Quantum computers are based on key principles of quantum mechanics, such as superposition, entanglement, interference, and the uncertainty principle. Quantum bits (qubits), units of quantum information, can be in a superposition of states 0 and 1 [4]. The main difference from classical bits is superposition, a visualization of classical and quantum bits, can be seen in picture 1. Superposition allows a qubit to be in multiple states at the same time, which provides an exponential advantage over classical computing. Quantum algorithms such as the Variational Quantum Eigensolver (VQE) and the Quantum Approximate Optimization Algorithm (QAOA) use these capabilities to find approximate solutions to combinatorial problems [5]. However, existing quantum devices are noisy due to

decoherence, i.e., the loss of quantum states due to interaction with the environment [6].

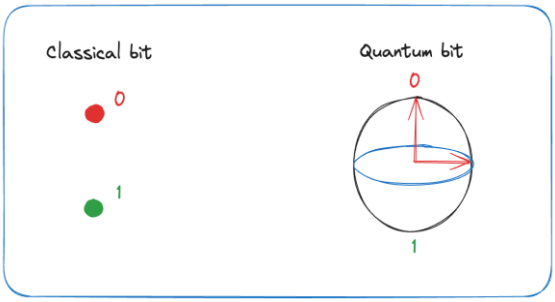

Figure 1: Visualization of classical and quantum bits.

Quantum computing makes it possible to find patterns that are inaccessible to classical computers. They promise to revolutionize many fields, including machine learning, optimization, complex systems modeling, cryptography, and much more. Investments in the development of quantum hardware, software packages, and simulators have already led to the creation of numerous tools for quantum development.

Quantum technologies open up new horizons for scientific research and technological innovation. They allow you to solve problems that are too complex for classical computers, including optimization problems, modeling molecules and materials, cryptography, machine learning, and many others. These computers can provide significant improvements in linear algebra problems, database searches, integer factorization, and quantum system simulations [6].

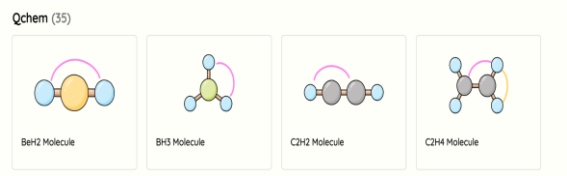

Figure 2: Pennylane imitation of molecules [8].

In medicine, quantum computing can help in the development of new drugs and therapies by simulating the interaction of molecules with great accuracy, in Figure 2 you can see that the Pennylane library has a simulation of molecules for quantum programming. In finance, they can be used to optimize portfolios and predict market trends. In machine learning, quantum algorithms can provide a significant acceleration in model training and processing large amounts of data.

2 RELEVANCE

Quantum computing opens up new possibilities for solving complex computational difficulties by leveraging the principles of quantum mechanics, such as superposition and entanglement. The Quantum Approximate Optimization Algorithm (QAOA) is a promising option for solving combined issues, in particular Max-Cut, due to its ability to approximate optimal solutions using quantum computing.

The Max-Cut problem consists in finding such a partition of the set of vertices of a graph into two subsets that maximizes the number of edges between these subsets. It is one of the classic NP-complex problems in graph theory and combinatorial optimization [7]. It has numerous practical applications in areas such as network design, resource allocation, statistical physics, and machine learning [7]. Since the Max-Cut issue is an NP-complex problem, traditional algorithms are insufficient to effectively solve problems of this kind. This makes the problem of graph section optimization important for both theoretical research and practical applications.

Although QAOA implementations for Max-Cut are already available in languages like Python, the addition of this functionality to JavaScript is significant for several reasons. By enabling quantum algorithms to be executed in this language, we can promote the widespread adoption of quantum computing concepts among web developers and the broader software development community. This accessibility can accelerate educational initiatives, collaborative research, and the integration of quantum algorithms into mainstream technology stacks. This integration is expected to enhance the applicability of quantum algorithms, fostering innovation and collaboration across various disciplines.

3 ANALYSIS OF CURRENT RESEARCH

To date, there is a significant amount of research devoted to the application of quantum algorithms to solve combinatorial problems such as Max-Cut. The main efforts are focused on the development and optimization of algorithms, as well as the creation of tools for their implementation on different quantum platforms. Among such platforms, IBM's Qiskit and Xanadu's Pennylane stand out, which have become the main tools for the development of quantum applications.

Qiskit is an open-source quantum computing software package that provides tools for developing and simulating quantum algorithms, as well as access to IBM quantum computers. Qiskit includes modules for creating quantum circuits, simulating their operation, and executing them on real quantum devices. This tool supports various noise models and allows you to investigate the efficiency of algorithms in the conditions of real quantum systems [9]. Particular attention should be paid to the IBM Quantum Composer software (Figure 3), which allows you to work with quantum bits in a graphical interface. It has a visualization of results, displays a Bloha sphere, and makes it possible to run or export a written circuit for working with quantum bits [10].

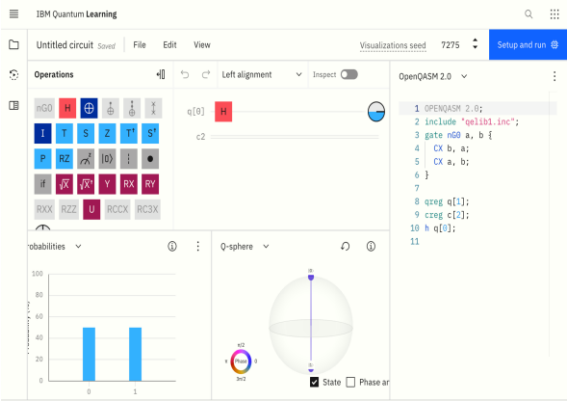

Figure 3: IBM Quantum Composer [10].

IBM does a lot of work to develop quantum technologies, from the development of quantum computers themselves to the creation of educational materials and research. It is noteworthy to mention that on the Learning Quantum resource of IBM, a comprehensive article titled “Solve utility-scale quantum optimization problems” is available, which focuses on the QAOA algorithm for resolving the Max-Cut problem [11].

Pennylane is another powerful tool for quantum computing that enables the development of hybrid quantum-classical algorithms. It integrates with popular machine learning libraries such as TensorFlow and PyTorch, making it easy to combine quantum algorithms with classical neural networks. Pennylane supports a wide range of quantum hardware platforms and simulators, making it a versatile tool for quantum machine learning and optimization research [12]. Pennylane also implements modules that implement work with quantum algorithms, in particular with the algorithm of approximate quantum optimization [13].

Despite significant progress in the development of quantum computing tools, there is a notable lack of implementations of quantum algorithms in the JavaScript programming language. Most current research focuses on the use of Python, which limits the accessibility of quantum computing for web application developers. JavaScript is one of the most popular programming languages and is widely used in all areas of software development. Creating implementations of quantum algorithms in JavaScript will be an important step in spreading quantum computing to a wider range of developers.

4 SOFTWARE IMPLEMENTATION OF THE ALGORITHM

The Max-Cut problem is a classic problem of graph theory and combinatorial optimization, which consists in finding such a partition of the set of vertices of a graph into two subsets that maximizes the number of edges between these subsets, solving the problem using the QAOA quantum algorithm, can significantly accelerate the optimization process and provide a faster solution compared to classical methods.

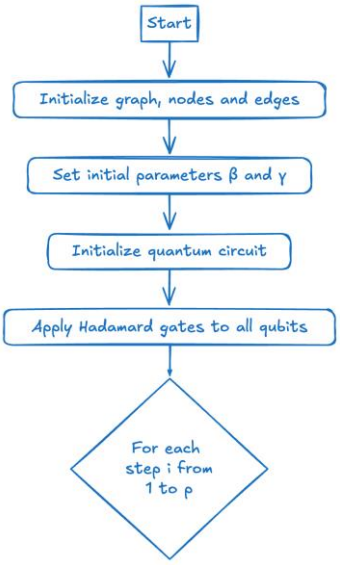

Figure 4: Part one of the flowchart of the QAOA.

Quantum Approximate Optimization Algorithm is one of the most promising approaches for solving complex combination problems such as Max-Cut. QAOA combines quantum computing with classical optimization methods, using the properties of

superposition and entanglement to achieve approximate solutions [14]. To visualize the algorithm, a flowchart was created (Figures 4-6).

The main idea of QAOA is to build a variational quantum circuit, which includes two main components: a phase separator (cost Hamiltonian) and a mixer (mixing Hamiltonian). The phase separator is responsible for encoding the optimization problem into a quantum system, while the mixer facilitates the mixing of states to explore the space of possible solutions.

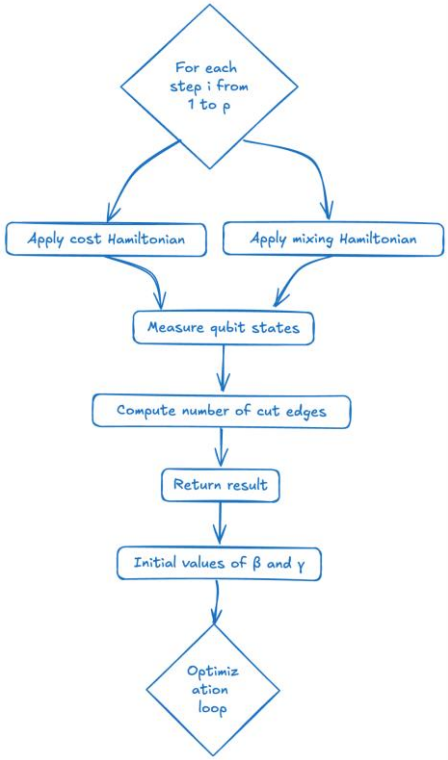

Figure 5: Part two of the QAOA flowchart.

The algorithm begins with the fact that a quantum system is in a state of superposition of all possible classical states. At each step of the algorithm, a phase separator is used, which changes the phase of each state according to the objective function, followed by a mixer, which performs quantum stirring. The process is repeated several times to get as close as possible to the optimal solution.

After several iterations of these transformations, a measurement of the quantum system is performed to obtain a classical solution that corresponds to an approximate optimal result for solving the Max-Cut problem. The efficiency of the algorithm depends on the optimization of parameters, which can be implemented using classical optimization methods [15].

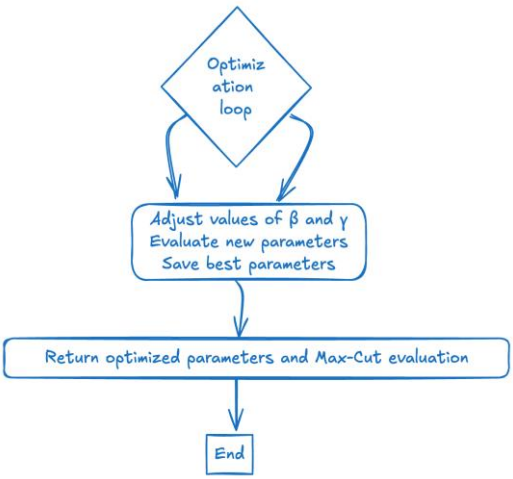

Figure 6: Part three of the QAOA flowchart.

QAOA is a flexible and versatile algorithm that can be applied to various combinatorial problems, thanks to its ability to effectively explore the solution space. Due to its versatility and efficiency, QAOA has the potential to significantly improve the solution of complex problems in various industries. The QAOA algorithm uses the principles of quantum mechanics, such as superposition and entanglement, to efficiently explore a large space of possible solutions [15]. The use of quantum computing in combination with classical optimization methods makes QAOA a powerful tool for solving optimization problems that are too complex for classical algorithms.

To implement QAOA to solve the Max-Cut problem in the JavaScript programming language, the Quantum.js [16] framework, written by the author of the article in the context of a scientific work. It provides tools for working with quantum computing in the JavaScript programming language. The main components of the implementation are functions for constructing quantum circuits and optimizing algorithm parameters.

The implementation begins with the import of the main Circuit class, which is used to create and manipulate quantum circuits.

TypeScript code:

```
import { Circuit } from '../../circuit';
```

The appendZZTerm function adds a phase separator corresponding to the Hamiltonian of the Max-Cut problem. It uses CX and RZ quantum gates to apply a phase shift between qubits.

TypeScript code:

```
   function appendZZTerm(qc: Circuit,
q1: number, q2: number, gamma: number)
{
     qc.cx(q1, q2);
     qc.rz(2 * gamma, q2);
     qc.cx(q1, q2);
   }
```

The appendCostOperatorCircuit function adds a phase separator for all edges of the graph using the appendZZTerm function.

TypeScript code:

```
   function
appendCostOperatorCircuit(qc: Circuit,
edges: Array<[number, number]>, gamma:
number) {
     for (const [i, j] of edges) {
       appendZZTerm(qc, i, j, gamma);
     }
   }
```

The appendXTerm function adds a mixer using RX quantum gates for each qubit.

TypeScript code:

```
   function appendXTerm(qc: Circuit, q:
number, beta: number) {
     qc.rx(2 * beta, q);
   }
```

The appendMixerOperatorCircuit function adds a mixer for all vertices of the graph using the appendXTerm function[17].

TypeScript code:

```
   function
appendMixerOperatorCircuit(qc: Circuit,
nodes: Array<number>, beta: number) {
     for (const n of nodes) {
       appendXTerm(qc, n, beta);
     }
   }
```

The main function getQAOACircuit creates a quantum circuit for QAOA. It takes the vertices and edges of the graph, as well as the β and γ parameters that determine the number of steps of the algorithm.

TypeScript code:

```
   function getQAOACircuit(
     nodes: Array<number>,
     edges: Array<[number, number]>,
     beta: number[],
     gamma: number[]
   ): Circuit {
       const p = beta.length; Number of
QAOA steps
       const qc = new
Circuit(nodes.length);

     // First Step: Apply the Adamartt
Gate Layer
     nodes.forEach((node) =>
qc.h(node));

     // Second Step: Apply p Duty
Operators
     for (let i = 0; i < p; i++) {
       appendCostOperatorCircuit(qc,
edges, gamma[i]);
       appendMixerOperatorCircuit(qc,
nodes, beta[i]);
     }

     // The Last Step: Measure the
Result
     nodes.forEach((node) =>
qc.measure(node));

     return qc;
   }
```

The objectiveFunction function defines the objective function for QAOA. It builds a quantum circuit, runs it, and evaluates the result by calculating the number of edges cut.

TypeScript code:

```
   export function objectiveFunction(
     beta: number[],
     gamma: number[],
     nodes: Array<number>,
     edges: Array<[number, number]>,
     idCircuitDraw?: string
   ): number {
     const qc = getQAOACircuit(nodes,
edges, beta, gamma);

     qc.run();

     if (typeof document !==
'undefined' && idCircuitDraw) {
       const circuitDraw =
document.getElementById(idCircuitDraw);
       if (circuitDraw) {
         circuitDraw.innerHTML =
qc.exportSVG();
       }
     }
     const result = qc.measure() as
number[];

     Evaluation of the result
     return computeMaxCutScore(result,
edges);
```

```
   }
```

The computeMaxCutScore function calculates the number of edges cut for a given partition of the vertices of the graph.

TypeScript code:

```
   function computeMaxCutScore(result:
number[], edges: Array<[number,
number]>): number {
     let score = 0;
     for (const [i, j] of edges) {
       if (result[i] !== result[j]) {
         score++;
       }
     }
     return score;
   }
```

To optimize β and γ parameters, the optimizeQAOAWithCOBYLA function is used, which implements a simple random search optimization approach.

TypeScript code:

```
   export function
optimizeQAOAWithCOBYLA(
     nodes: Array<number>,
     edges: Array<[number, number]>,
     steps: number,
     idCircuitDraw?: string
   ): { beta: number[]; gamma:
number[]; score: number; maxCutScore:
number } {
     let bestBeta: number[] =
Array(steps).fill(Math.PI / 4); Initial
assumption for beta
     let bestGamma: number[] =
Array(steps).fill(Math.PI / 4); Initial
assumption for gamma
     let bestScore =
objectiveFunction(bestBeta, bestGamma,
nodes, edges, idCircuitDraw);
     let bestMaxCutScore = bestScore;

     const maxIterations = 100; Maximum
number of iterations
     const randomStepScale = 0.01;
Scale of Random Change

     for (let iter = 0; iter <
maxIterations; iter++) {
       for (let i = 0; i < steps; i++)
{
         Const Newbeta = [...
Bestbeta];
         constant newgamma = [...
Bestgamma];
         // Changing the beta and gamma
values to a small random increment
         newBeta[i] += (Math.random() -
0.5) * randomStepScale;
         newGamma[i] += (Math.random()
- 0.5) * randomStepScale;

         const newScore =
objectiveFunction(newBeta, newGamma,
nodes, edges);

         if (newScore > bestScore) {
           bestBeta = newBeta;
           Bestgamma = Neugamma;
           bestScore = newScore;
           bestMaxCutScore = newScore;
         }
       }
     }

     return { beta: bestBeta, gamma:
bestGamma, score: bestScore,
maxCutScore: bestMaxCutScore };
   }
```

This implementation demonstrates the integration of the QAOA quantum algorithm with JavaScript, allowing the power of quantum computing to be harnessed on a web page or in a Node.js environment. The use of quantum computing in JavaScript contributes to the spread of quantum technologies among web developers and opens up new opportunities for optimizing complex problems.

5 CONNECTING THE FRAMEWORK TO A WEB APPLICATION

To implement the use of the algorithm, it was decided to build a web application using modern web development technologies [17]. The interaction of the application with the user (Figure 7) can be described as follows:

1) Open the page;
2) Enter graph edges;
3) Click the draw graph button;
4) Click the start calculation button;
5) Get the result.

The architecture of the application is built based on the following components[17]:

1) App.tsx: The main component of React, containing the core logic and rendering of the application;

2) Form.tsx: A form component that provides user input required to run the QAOA algorithm;
3) graph.ts: contains a Graph class that implements basic functionality for creating graphs, such as adding nodes, edges, and graph visualization;
4) useEdges.ts: A React hook for controlling the state of edges in a graph, provides functions for adding, removing, and updating edges;
5) main.tsx: The main login file for the application, which includes setting up and initializing the main components;
6) App.css: A style file for the add-on containing CSS rules for the main components.

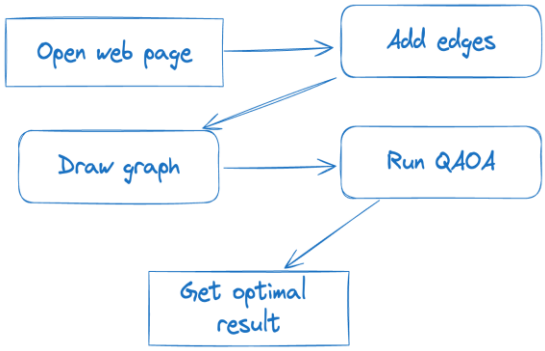

Figure 7: Scheme of the application.

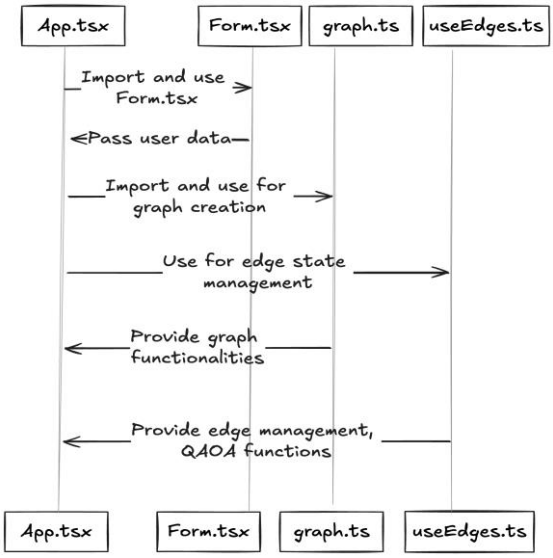

Figure 8: Application component interaction.

The interaction of the components is described in Figure 8. The components interact as follows [17]:

1) Initialization (main.tsx): The main.tsx file initializes the application by connecting the main component of App.tsx;
2) Main component (App.tsx): The App.tsx component imports and uses Form.tsx for user input, graph.js for graph visualization, and other helper components and hooks;
3) Data Entry Form (Form.tsx): Form.tsx is responsible for the user entering parameters to run the QAOA algorithm. This data is transmitted to App.tsx for further processing;
4) Graph (graph.ts): provides an implementation of the Graph class, which is used to create and manage graphs;
5) Edge Management and QAOA (useEdges.ts): The useEdges.ts Hook is used to manage the state of edges in graphs, providing functions for adding, removing, and updating edges, connecting the quantum.js library, and running QAOA algorithm calculations;
6) Styles (App.css): App.css contains CSS rules for the design of the main components of the application.

The Graphical Interface (Figure 9) of the appendix can be divided into 4 zones [17]:

1) Graph construction;
2) A form for constructing a graph;
3) Graph of quantum interaction between qubits;
4) Display of results.

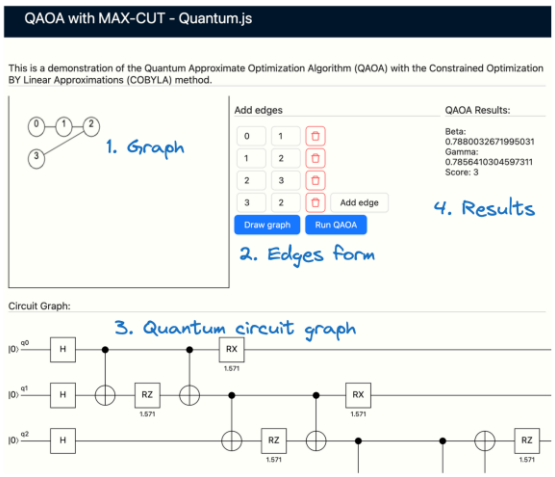

Figure 9: Application GUI.

6 CONCLUSIONS

As a result of the study, a quantum.js framework for high-level interaction with quantum bits was built and an implementation of the QAOA quantum hybrid algorithm was added to solve the Max-Cut problem.

By disabling the quantum.js framework to the web application, a graphical interface was built that allows you to: build and visualize graphs, use the QAOA algorithm in the browser, draw a quantum scheme of interaction between qubits used in the QAOA algorithm. For future improvement of the framework, the following points quantum.js highlighted:

- connect to quantum computers in the cloud;
- add other quantum-hybrid algorithms;
- build the functions of the framework for solving real problems (the task of optimizing resources, solving logistic problems).

To improve the performance of the web application, it is worth paying attention to the user's interaction with the application, improving styles, and adding validation for entering edge values.

The implementation demonstrated the possibility of effective use of quantum algorithms in the web environment, which contributes to the widespread use of quantum computing among web developers. The main achievements were the creation of the quantum.js framework, which provides high-level interaction with quantum bits and the integration of quantum bits computing with web technologies through a modern graphical interface. It has been shown that the QAOA quantum algorithm can significantly improve the optimization process for complex combinatorial problems compared to classical methods. The use of quantum computing in combination with classical optimization methods makes QAOA a powerful tool for solving NP-complex problems. Prospects for further development include connecting to quantum computers in the cloud, expanding documentation, adding new quantum-hybrid algorithms, and improving the user interface of the web application.

REFERENCES

[1] R. Feynman, [Online]. Available: https://en.wikipedia.org/wiki/Richard_Feynman.

[2] A. Almajid, "Summary of Real Quantum Mechanics," 2024. 10.13140/RG.2.2.20837.61928.

[3] “Google’s Sycamore: Exploring the Power of Google’s Quantum Computer,” [Online]. Available: https://medium.com/the-quantastic-journal/googles-sycamore-exploring-the-power-of-google-s-quantum-computer-266374339d5.

[4] “Qubit,” [Online]. Available: https://en.wikipedia.org/wiki/Qubit.

[5] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. Booth, and J. Tennyson, “The Variational Quantum Eigensolver: A review of methods and best practices,” Physics Reports, vol. 986, pp. 1-128, 2022. 10.1016/j.physrep.2022.08.003.

[6] H. Sahu and Dr. Gupta, "Quantum Computing Toolkit From Nuts and Bolts to Sack of Tools," 2023.

[7] “The Max-Cut Problem,” [Online]. Available: https://www.cs.cmu.edu/afs/cs/academic/class/15854-f05/www/scribe/lec02.pdf.

[8] “Pennylane datasets,” [Online]. Available: https://pennylane.ai/datasets/.

[9] “Qiskit,” [Online]. Available: https://docs.quantum.ibm.com/guides.

[10] “IBM Quantum Composer,” [Online]. Available: https://quantum.ibm.com/composer.

[11] “Solve utility-scale quantum optimization problems,” [Online]. Available: https://learning.quantum.ibm.com/tutorial/quantum-approximate-optimization-algorithm.

[12] V. Bergholm et al., "PennyLane: Automatic differentiation of hybrid quantum-classical computations," 2018. arXiv:1811.04968.

[13] “Intro to QAOA,” [Online]. Available: https://pennylane.ai/qml/demos/tutorial_qaoa_intro/.

[14] M. X. Goemans and D. P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming," Journal of the ACM, vol. 42, no. 6, pp. 1115-1145, 1995.

[15] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," 2014. arXiv:1411.4028.

[16] “Quantum.js Framework.” [Online]. Available: https://github.com/EarlOld/quantum.js.

[17] “QAOA-quantum,” [Online]. Available: https://github.com/EarlOld/QAOA-quantum.js.