BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 74(3), 2026, Article number: e157570
DOI: 10.24425/bpasts.2026.157570

Quantum approximation optimization algorithm for traveling salesman problem on 5-qubit IQM spark quantum computer

Igor DUDKIEWICZ and Wojciech BOŻEJKO*

Department of Control and Quantum Computing, Wrocław University of Science and Technology, Janiszewskiego 11/17, 50-372 Wrocław, Poland

Abstract. In this paper, we consider a method for solving difficult combinatorial optimization problems on real quantum computers. We focus on the traveling salesman problem as a representative problem for a group of problems where the solution is represented by a permutation. Typically, existing algorithmic solutions use binary matrices to store this permutation – the QUBO (quadratic unconstrained binary optimization) model. We propose a new way of encoding permutations on quantum computers, using a significantly smaller number of qubits than binary matrix encodings. Our method allows for significant performance improvements for any problem whose input or solution is a permutation. We demonstrate an example implementation of the traveling salesman problem on the IQM quantum computers: IQM Spark 5-qubit 'ODRA-5' computer and IQM Radiance 'Garnet' 20-qubit computer.

Keywords: quantum optimization; quantum computing; QAOA; TSP; permutation encoding.

1. INTRODUCTION

Combinatorial optimization problems, such as those found in logistics, scheduling, and machine learning, lie at the heart of many real-world computational challenges. Among them, problems classified as NP-hard, such as the traveling salesman problem (TSP) pose significant difficulties for classical computers due to the exponential growth of the solution space. Quantum computing has emerged as a promising paradigm with the potential to provide polynomial or even exponential speedups for specific classes of such problems. Algorithms like the quantum approximate optimization algorithm (QAOA) and the variational quantum eigensolver (VQE) have opened new directions for attacking these challenges using near-term quantum hardware.

A common requirement across many quantum approaches to NP-hard problems is the need to encode combinatorial constraints using binary variables. For example, problems involving permutations, binary assignments, or graph structures are commonly mapped to quantum circuits using a hot-one binary matrix representation. Although conceptually simple, these encodings are highly inefficient in terms of qubit usage and lead to an explosion in Hilbert space size. For example, representing a permutation of n elements using a binary matrix requires $O(n^2)$ qubits, although only $n!$ configurations are valid, compared to $2^{O(n^2)}$ possible quantum states. In the next section, we wll explore in detail different commonly used and theoretical methods for representing permutations on quantum hardware, as well as their advantages and drawbacks.

This paper focuses on the traveling salesman problem as a canonical example, but the techniques discussed have broader applicability to a wide range of quantum optimization problems. We propose a practical application of a more compact quantum encoding scheme in which the position of each city on the tour is represented as a binary number. This requires only $n \cdot \lceil \log_2(n) \rceil$ qubits, significantly reducing the quantum resource requirements while still allowing the representation of all valid permutations.

To illustrate the theoretical accuracy improvements of this approach, Table 1 presents a comparison between the number of actual permutations, the number of possible solutions using the traditional binary matrix representation, and the number of possible solutions using our proposed method.

Table 1
Comparison of the number of permutations ($n!$) versus the number of possible solutions using traditional binary matrix representation (2^{n^2}) and our proposed approach $\left(2^{n \cdot \lceil \log_2(n) \rceil}\right)$

n	$n!$	2^{n^2}	$2^{n \cdot \lceil \log_2(n) \rceil}$
3	6.00×10^0	5.12×10^2	6.40×10^1
4	2.40×10^1	6.55×10^4	2.56×10^2
5	1.20×10^2	3.36×10^7	3.28×10^4
6	7.20×10^2	6.87×10^{10}	2.62×10^5
7	5.04×10^3	5.63×10^{14}	2.10×10^6
8	4.03×10^4	1.84×10^{19}	1.68×10^7
9	3.63×10^5	1.94×10^{24}	1.34×10^8
10	3.63×10^6	1.27×10^{30}	1.07×10^9

*e-mail: wojciech.bozejko@pwr.edu.pl

Manuscript submitted 2025-08-18, revised 2025-12-18, initially accepted for publication 2026-01-28, published in May 2026.

2. BACKGROUND AND RELATED WORK

2.1. The traveling salesman problem

The traveling salesman problem (TSP) is one of the most extensively studied combinatorial optimization problems in computer science and operations research. Given a set of n cities and the distances between every pair of cities, the goal is to find the shortest possible route that visits each city exactly once and returns to the starting city. Formally, for a complete graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$ representing cities and edge weights d_{ij} representing distances between cities i and j, the TSP seeks to find a permutation π of $\{1, 2, \ldots, n\}$ that minimizes the following:

$$\left(\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}\right) + d_{\pi(n),\pi(1)}\,.$$

The TSP belongs to the class of NP-hard problems, which means that up-to-now no polynomial-time algorithm is known to solve it optimally for all instances, and it is widely believed that no such algorithm exists. The complexity of the problem increases factorially with the number of cities: there are $(n-1)!/2$ possible tours for n cities (which account for rotational and reflectional symmetries). This exponential growth makes the TSP computationally intractable for large instances using brute-force methods. Both symmetrical and asymmetrical TSP are equivalent (transformable).

Despite its computational difficulty, the TSP has numerous practical applications, including vehicle routing, logistics optimization, manufacturing (drill path optimization), DNA sequencing, and VLSI design. The problem also serves as a benchmark for testing new optimization algorithms and heuristics due to its simple formulation, yet complex, solution landscape.

The TSP can be formulated as a quadratic unconstrained binary optimization (QUBO) problem, making it particularly suitable for quantum optimization approaches.

2.2. The quantum approximate optimization algorithm (QAOA)

The quantum approximate optimization algorithm (QAOA) is a hybrid quantum–classical variational algorithm designed to find approximate solutions to combinatorial optimization problems. Given a cost Hamiltonian H_C encoding the objective of the problem and a mixer Hamiltonian H_M, the trial state is prepared as

$$|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \prod_{j=1}^{p} e^{-i\beta_j H_M} e^{-i\gamma_j H_C} |+\rangle^{\otimes n},$$

where $|+\rangle^{\otimes n}$ is the uniform superposition over all bit strings and $\{\gamma_j, \beta_j\}$ are variational parameters. These parameters are optimized by a classical outer loop to minimize the expectation value $\langle\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})|H_C|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$, which serves as the cost function. As the circuit depth parameter p increases, the approximation ratio generally improves, at the cost of deeper and noisier circuits. QAOA was introduced by Farhi, Goldstone, and Gutmann in 2014 [1].

2.3. Permutation encoding methods

Glos *et al*. [2] theoretically explore various permutation encoding strategies for TSP within the QAOA. These encodings aim to efficiently represent TSP solutions on quantum hardware, each with distinct advantages and trade-offs in terms of qubit count, constraint implementation, and circuit depth.

More recently, Ciacco *et al*. [3] provide a comprehensive review of quantum algorithms for optimization problems arising in medicine, finance, and logistics, including routing and vehicle-routing problems closely related to the TSP. Their work contextualizes such encoding strategies within the broader landscape of practically motivated quantum algorithms and highlights the importance of space-efficient representations for near-term quantum devices.

Hot-one encoding. The conventional encoding, currently most commonly used to represent permutations on quantum machines, represents permutations using hot-one vectors, where binary variables indicate whether a specific city is visited at a particular time step. For a traveling salesman problem with N cities, this approach requires $\mathbf{N^2}$ binary variables, as each time-city combination needs its own bit. The encoding includes constraints ensuring that exactly one city is visited at each time step and that each city is visited exactly once throughout the tour. While this representation produces circuits with a manageable depth of $O(N)$, it is highly inefficient in terms of space complexity, requiring significantly more qubits than the information-theoretic minimum.

Binary encoding. An alternative approach encodes cities using a standard binary representation, where each collection of bits represents a city number in base 2. This method requires only $\lceil \log N \rceil$ qubits per time step, resulting in approximately $N \log N$ total qubits, which approaches the information-theoretic lower bound for representing $N!$ permutations. The encoding necessitates validation constraints to ensure that binary representations correspond to valid city indices not exceeding $N-1$. As analyzed by Glos *et al*. [2], while achieving near-optimal qubit efficiency, this higher-order binary optimization encoding introduces deeper quantum circuits with increased computational depth compared to QUBO. The authors conclude that while binary encoding is theoretically valuable, its circuit depth likely exceeds the capabilities of current NISQ hardware. Despite this limitation, we have chosen to use this encoding scheme in our work for two key reasons. First, this approach requires the fewest qubits among all feasible solutions, making it well-suited for the small-scale quantum computers that are currently most widely available. Second, although circuit depth presents a challenge for near-term devices, advances in quantum hardware reliability over the coming years could make algorithms based on this encoding scheme highly practical. Therefore, developing such algorithms now represents a valuable investment in preparing for mid-term quantum computing capabilities.

Mixed encoding. The proposed remedy to the large depth of algorithms using this encoding scheme is a mixed encoding, which trades some extra qubits for shallower, more hardware-friendly circuits. A hybrid approach partitions the time steps into

two groups: some encoded using one-hot vectors and others using binary representation. The refined mixed encoding divides each time step into L bunches of K qubits, where $N = (2^K - 1)L$, with constraints ensuring that at most one bunch per time step contains nonzero bits. This scheme provides a tunable trade-off between space and depth complexity by adjusting the parameter α where $K = \alpha \log N$, resulting in qubit requirements of $\Theta(N^{2-\alpha} \log N)$ and circuit depth of $O(N^{1+\alpha} \log^2 N)$. The interpolation between extremes allows practitioners to optimize for specific hardware constraints, balancing between qubit-limited and depth-limited quantum devices.

Factorial number system (optimal) encoding. The most qubit-efficient representation employs factorial numbering combined with Lehmer codes to directly encode permutations. This method requires only $\lceil \log(N!) \rceil$ qubits, achieving the theoretical minimum for representing all possible routes. The encoding uses factorial representation, where the i-th digit can range from 0 to $i - 1$, which is then converted to permutations through Lehmer codes by sequentially selecting elements from an ordered list. This approach is not at all practical and only serves the purpose of showcasing a theoretical minimal number of qubits needed.

Domain wall encoding. A way to represent permutations on quantum hardware, that is sometimes used in literature, for example by Nicholas Chancellor [4], but was not mentioned by Glos *et al*., is a domain wall encoding. It represents discrete variables using the positions of domain walls in one-dimensional ferromagnetic Ising spin chains, where a domain wall occurs at boundaries between regions of opposite spin orientations. For a discrete variable taking m values, this requires $m - 1$ qubits with exactly one domain wall present in valid states. For the traveling salesman problem with N cities, this translates to $N(N - 1)$ qubits total, representing a small improvement over the standard hot-one matrix.

2.4. Practical implementation examples

Classical TSP solvers. On the classical side, state-of-the-art TSP solutions include the Concorde TSP Solver [5], which uses branch-and-cut methods with problem-specific heuristics, and the LKH (Lin–Kernighan–Helsgaun) algorithm [6], which employs powerful local search strategies. These solvers are highly optimized but inherently classical and do not generalize well to other types of combinatorial problems, and are not well-suited for NP-hard problems.

Permutation encoding in quantum algorithms. As was mentioned in the introduction, many NP-hard problems necessitate a way to represent a permutation in some way. We will reference some recent papers to showcase that it is common to use the nonefficient, in terms of the number of qubits required, hot-one matrix, as well as some papers proposing practical or theoretical improvements over that approach, which while showing some potential improvements, still use a far larger number of qubits than the approach explored in this paper.

Hot-one Matrix and Domain-Wall examples. The simplicity and ease of implementation make the hot-one matrix the most commonly used way to encode a permutation, and it has been the case for years. Already in 2014, in the paper "Ising formulations of many NP problems," Andrew Lucas [7] has shown a QUBO formulation for TSP using a hot-one matrix. Other examples through the years include Papalitsas *et al*. 2019 paper "A QUBO Model for the traveling salesman problem with Time Windows" [8], that extends TSP to include time windows (TSPTW), formulating it as a QUBO problem. Also in 2019, "Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer" by Aramon *et al*. [9] tests various optimization problems, including TSP using a hot-one matrix. Even as recently as this year, papers are coming out using a hot-one matrix, for example, "Comparative Analysis of Quantum Approximate Optimization Algorithms for Solving the travelling salesperson problem – Revised With Data and Further Analysis" by Siddharth Chander [10] uses a hot-one matrix to showcase the advantages of using a hot-start in QAOA.

Proposed improvements. The prominence of the hot-one matrix also leads to a number of papers proposing potential improvements over that approach, something we are also trying to do in this paper. For example, the paper "The Quantum Approximate Algorithm for Solving Traveling Salesman Problem" by Ruan *et al*. [11]. In the paper, the authors propose an approach where, instead of using the conventional hot-one encoding over city-time pairs, they map edges connecting cities directly to qubits, thereby reducing the total qubit count. This edge-based mapping decreases the search space, theoretically allowing solutions with roughly half the qubits required by standard hot-one methods. The study demonstrates the approach via simulations on small TSP instances. No experiments on real quantum hardware are reported, so the practical applicability of this method to NISQ devices remains untested. Furthermore, while using half the number of qubits required for the hot-one matrix is a significant improvement, for large n, the Hilbert space size will still be significantly larger than the actual number of permutations. A more recent paper is "Comparative Study of Variations in Quantum Approximate Optimization Algorithms for the Traveling Salesman Problem" by Qian *et al*. from 2023 [12]. Authors evaluate three distinct QAOA mixer designs – X, XY, and RS mixers – considering their performances in terms of numerical accuracy and optimization cost. They present numerical results obtained from gate-based digital quantum simulators, specifically targeting TSP instances with 3, 4, and 5 cities. The simulations in the study demonstrate that the digital quantum simulation of problem-inspired Ansatz is a promising candidate for finding optimal TSP solutions. While the paper shows theoretical advantages that can be gained by using a well-designed Ansatz, it uses a hot-one matrix and only tackles small problem instances. Finding a problem-specific mixer is much more complex for larger problems, especially ones using more complicated permutation encoding schemes, aiming to reduce the number of qubits used.

2.5. IQM Spark 5-qubit quantum computer

The IQM Spark is a superconducting quantum processor designed as an educational and research-oriented platform for universities and laboratories. Built by IQM Quantum Comput-

ers, the Spark system integrates high-coherence qubits, modular control electronics, and a robust software stack, offering researchers access to cutting-edge quantum hardware in a compact form.

Architecture and technology. IQM Spark is based on transmon qubits implemented with superconducting circuits (see Fig. 1). The processor consists of five fixed-frequency transmon qubits arranged in a linear or nearest-neighbor topology, depending on the specific fabrication layout. These qubits operate at millikelvin temperatures in a dilution refrigerator to preserve quantum coherence.

- **Number of qubits:** 5 superconducting transmon qubits.
- **Qubit connectivity:** Nearest-neighbor coupling (1D linear topology, see Fig. 2).
- **Gate set:** Universal gate set including single-qubit X, Y, Z, H, S, T gates and two-qubit CNOT gates between connected qubits.
- **Average single-qubit gate fidelity:** > 99.9%.
- **Average two-qubit gate fidelity:** > 98.5%.
- **Typical qubit coherence times:**
  - T_1 (Relaxation): 30–60 μs.
  - T_2 (Dephasing): 20–50 μs.

Fig. 1. Photograph of the 'ODRA-5' IQM Spark 5-qubit quantum computer installed at Wroclaw University of Science and Technology, used in our experiments

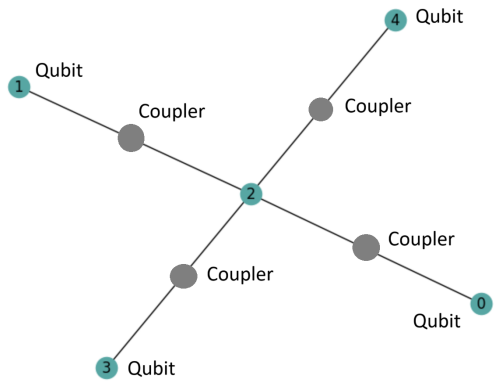

Fig. 2. Qubit connectivity graph of the IQM Spark 5-qubit processor used in our experiments. Nodes represent physical qubits (also used as couplers)

- **Gate time:**
  - Single-qubit gates: 20–40ns.
  - Two-qubit gates (CZ or CNOT): 100–250 ns.
- **Qubit frequency:** 4–6 GHz range (fixed-frequency transmons).
- **Readout:** Dispersive readout via dedicated resonators coupled to each qubit.
- **Readout fidelity:** > 95%, depending on the calibration.

Control and software stack. The system includes IQM's proprietary control electronics and is compatible with standard quantum software frameworks such as Qiskit and Cirq via a customized back-end interface. Researchers can also use the IQM Operator environment, a Python-based software toolkit for pulse-level control and experiment execution.

- **Pulse-level access:** Supported via OpenPulse and IQM's own APIs.
- **Compiler:** Hardware-aware compiler optimizes gate decomposition and routing.
- **Integration:** Supports cloud and on-premise deployment models.
- **Programming languages:** Python (via Qiskit, Cirq, and IQM Operator SDK).

Use cases. IQM Spark is particularly suited for research in quantum algorithms, error mitigation techniques, quantum control, and educational purposes. Its local availability and open access to low-level control make it an ideal platform for developing customized quantum experiments, including variational algorithms such as VQE and QAOA.

3. PERMUTATION VALIDATION

Having established the theoretical advantages of our proposed encoding scheme, namely requiring far fewer qubits, compared to commonly used hot-one matrix, we will now address the critical challenge of validating whether a given quantum state represents a valid permutation. The validation process differs depending on whether n is a power of 2, as this case is naturally better suited for our binary-based approach.

3.1. Validation for n as a power of 2

When n is a power of 2, the validation process becomes relatively straightforward. Since we use $n \cdot \log_2(n)$ qubits to represent the permutation, each element position is encoded using exactly $\log_2(n)$ qubits. The primary requirement for a valid permutation is that each set of $\log_2(n)$ qubits represents a different number from the range $[0, n-1]$.

The core validation operation involves checking that each pair of encoded positions represents different values. This can be implemented using a quantum circuit that compares whether two binary numbers and outputs are different.

The circuit shown in Fig. 3 demonstrates a practical implementation of comparison circuit. Due to current hardware limitations (specifically, access to a 5-qubit quantum computer), the example shows a circuit that checks whether two numbers

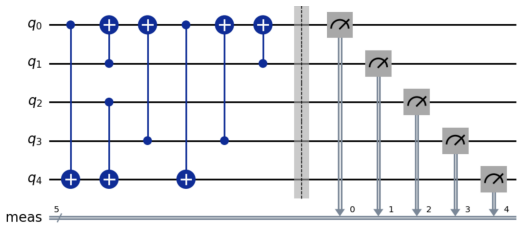

Fig. 3. Quantum circuit for checking whether two binary numbers are different. The circuit compares numbers represented by qubits q_0, q_1 and q_2, q_3, with the result stored in q_4

in the range $[0,3]$, represented by qubits q_0, q_1 and q_2, q_3, are different, with the result stored in q_4.

An important consideration in the circuit design is that q_0 is modified during the computation process. While this modification is completely reversible and undone at the end of the circuit, it presents challenges for current quantum hardware. The additional operations required to restore the original qubit state increase the circuit depth, which in turn decreases the overall accuracy due to decoherence and gate errors. Given the limitations of presently available quantum technology, it would be preferable to use auxiliary qubits for intermediate calculations to avoid the need for state restoration. As quantum computer accuracy improves in the future, this constraint will likely become less problematic.

To validate that all encoded positions in a permutation are distinct, we apply the pairwise comparison circuit to every pair of encoded positions. For a permutation of size n, this requires $\binom{n}{2} = \frac{n(n-1)}{2}$ comparison operations.

3.2. Validation for general *n*

We construct pairwise comparison circuits for n as a power of 2 and range-check circuits for general n, requiring $\binom{n}{2}$ comparisons or explicit forbidden-value checks.

When n is not a power of 2, additional constraints must be enforced to ensure that the encoded positions represent valid indices in the range $[0, n-1]$. We propose two primary approaches to handle this situation:

3.2.1. Range validation approach

The first solution involves adding explicit checks to verify that all encoded numbers are smaller than n. This can be implemented by systematically checking that no encoded position equals n, $n+1$, $n+2$, and so on, up to $2^{\lceil \log_2(n) \rceil} - 1$ (the maximum value that can be represented with $\lceil \log_2(n) \rceil$ qubits).

For each forbidden value $k \geq n$, we can construct a quantum circuit that checks whether any of the encoded positions equals k. This approach requires additional quantum circuits but maintains the compact representation of the permutation.

3.2.2. Expansion approach

The second solution, applicable when sufficient qubits are available, involves expanding the permutation set with dummy elements to make the total size a power of 2. These additional elements are chosen such that they do not affect the optimal solution to the original TSP instance.

This approach eliminates the need for range validation circuits but requires additional qubits to represent the expanded problem. The trade-off between circuit complexity and qubit requirements depends on the specific quantum hardware constraints and the problem size.

Both approaches maintain the fundamental advantage of our encoding scheme while addressing the challenges posed by non-power-of-2 problem sizes. The choice between them should be made based on the available quantum resources and the specific characteristics of the quantum hardware being used.

3.3. Using multi-controlled NOT gates

It is worth noting that if quantum computer accuracy were not a limiting factor, one could check for all permutations using a multi-controlled NOT gates. However, since the number of permutations grows as $n!$, which is much faster than the quadratic growth of pairwise comparisons, this approach would likely never be practical for large values of n. We will demonstrate this alternative approach at the beginning of the next section, since, with the hardware limitations present, it was an appealing approach.

3.4. Special cases

It is clear that the more complex process of permutation validation compared to a hot-one matrix is something that can work against this approach. While even when permutation validation has to be done, this approach offers a potential advantage, especially for large n, we would like to note that there are situations where this permutation encoding scheme can be used without the need for permutation validation. For example, hybrid classical-quantum algorithms, where a classical computer can ensure that a valid permutation is represented. For example, something else we are currently working on is a quantum-classical Tabu search TSP algorithm, where running permutation validation is not needed, since the quantum circuit is always initialized with qubits representing a valid permutation.

3.5. Permutation validation in practice

While all other experiments in this paper were conducted on the 5-qubit IQM Spark quantum computer available at our university, we sought to demonstrate the practical implementation of permutation validation on a larger system. To this end, we developed and executed code remotely on the IQM Radiance 'Garnet' 20-qubit quantum computer installed in Aalto, Finland. Due to the increased number of qubits utilized, both the circuit diagram and histogram would be illegible; therefore, we present the textual results instead, as well as summarize the code implementation here, with the full listing provided in Appendix A (Listing 1). Figure 4 shows the topology of the quantum computer used in this experiment.

The algorithm operates as follows: the first 8 qubits contain the input sequence to be verified. Each pair of qubits is compared, with results stored in the subsequent 6 qubits – outputting

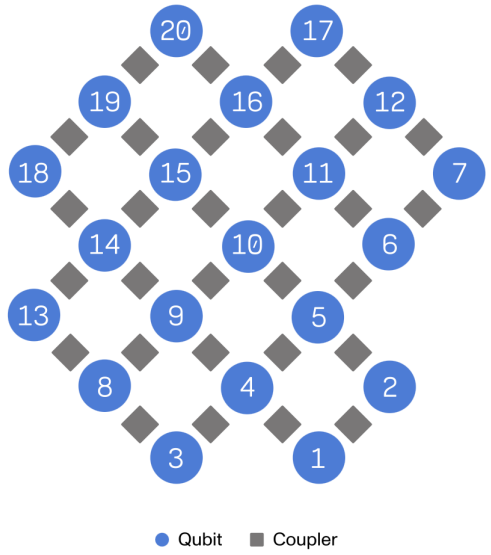

Fig. 4. Topology of the IQM Garnet 20-qubit quantum computer used for permutation validation experiments

0 if the numbers differ and 1 if they are identical. Finally, a logical OR operation is performed across all comparison results to determine the final validation outcome. Here, we used an 'OR' comparison tree to minimize the number of comparisons.

The experimental performance was deemed satisfactory based on our evaluation criteria. We define successful quantum program execution as achieving a scenario where the correct solution exhibits the highest probability, with no other solution approaching comparable frequency. In contrast, experiments where incorrect solutions appeared with frequencies nearly matching the correct solution would indicate inadequate performance, even if the correct solution appeared with high frequency.

Given that we measured 9 qubits in this experiment, achieving the high success percentages observed in our 5-qubit experiments was unrealistic due to the exponential scaling of the quantum state space. The implementation produced the correct output (unmodified input qubits with accurate permutation identification) in slightly over 6.5% of measurements. Critically, no alternative solution exceeded 3% frequency, establishing a clear distinction between correct and incorrect outcomes. Based on this significant gap in solution probabilities, we classify this experiment as successful.

4. PRACTICAL IMPLEMENTATION ON 5-QUBIT HARDWARE

4.1. QUBO construction using binary encoding

We use a binary encoding to represent permutations for the traveling salesman problem (TSP). Each position $p \in \{1,\dots,N\}$ in the tour is assigned a group of $k = \lceil \log_2 N \rceil$ binary variables:

$$x_{p,0}, x_{p,1}, \dots, x_{p,k-1} \in \{0,1\},$$

which collectively represent the city index v_p visited at position p, using the big-endian binary expansion:

$$v_p = \sum_{b=0}^{k-1} 2^{k-1-b} x_{p,b}.$$

Our goal is to express the classical cost function

$$C = \sum_{p=1}^{N} D(v_p, v_{p+1}),$$

where $D(i,j)$ is the distance between cities i and j, and we identify $v_{N+1} \equiv v_1$ to complete the tour.

To algebraically implement this cost in terms of the binary variables $x_{p,b}$, we must express the quantity $D(v_p, v_{p+1})$ using only polynomial terms in these variables.

Encoding value comparisons with $\delta_{x,\beta}$. We define a simple mechanism to check whether a binary variable x matches a desired bit value $\beta \in \{0,1\}$:

$$\delta_{x,\beta} = \begin{cases} x & \text{if } \beta = 1, \\ 1-x & \text{if } \beta = 0. \end{cases}$$

This expression equals 1 if and only if $x = \beta$. It allows us to construct expressions that check whether a group of binary variables matches a specific integer value. For example, to check whether the group $(x_{p,0}, x_{p,1}, x_{p,2})$ encodes the value 5 (which is 101 in binary, big-endian), we write:

$$\delta_{x_{p,0},1} \cdot \delta_{x_{p,1},0} \cdot \delta_{x_{p,2},1} = x_{p,0} \cdot (1 - x_{p,1}) \cdot x_{p,2}.$$

More generally, for any fixed number $i \in \{0,\dots,N-1\}$ with binary representation $\mathrm{bit}_b(i)$ (where $\mathrm{bit}_b(i)$ is the b-th bit in big-endian order), we define an indicator function that outputs 1 only when $x_{p,0},\dots,x_{p,k-1}$ encode the value i:

$$\mathrm{Indicator}(v_p = i) = \prod_{b=0}^{k-1} \delta_{x_{p,b},\mathrm{bit}_b(i)}.$$

Using these Indicators, we express the total distance function as:

$$D(v_p, v_{p+1}) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} D(i,j) \cdot \left(\prod_{b=0}^{k-1} \delta_{(x_{p,b}),\mathrm{bit}_b(i)} \right) \cdot \left(\prod_{b'=0}^{k-1} \delta_{(x_{p+1,b'}),\mathrm{bit}_{b'}(j)} \right).$$

This is a high-degree polynomial in binary variables, which can be quadratized using standard techniques involving auxiliary variables and penalty terms.

Given the current limitations of quantum hardware, specifically the small number of available qubits and the noise constraints that restrict circuit depth, it is feasible to solve instances of the TSP with a highly limited number of cities. In such small-scale settings, the number of possible input encodings remains

tractable. Therefore, instead of deriving a general-purpose cost function, one practical alternative is to construct a problem-specific objective function directly, by assigning the correct output (e.g., tour cost) to each valid input configuration. This approach, while infeasible at large scale, can be performed manually for small cases.

4.2. Initial implementation attempt

Given the constraint of only 5 available qubits, our initial approach was to solve a TSP instance with 3 locations. Using our encoding scheme, this requires 2 qubits to represent the position of each element in the permutation, totaling 6 qubits for the complete representation. Since we lacked the necessary 6 qubits, we fixed the first location at position 1, reducing the problem to determining the positions of the remaining two locations using the first 4 qubits.

With this constraint, only two legal permutation encodings were possible:

- `1001`: representing the second location at position 3 and the third location at position 2.
- `0110`: representing the second location at position 2 and the third location at position 3.

4.3. Validation circuit implementation

The first step involved constructing a quantum circuit to verify whether the first four qubits represented a legal solution. Following the multi-controlled NOT approach mentioned in the previous section, we implemented a circuit that explicitly checks for both valid permutation encodings.

The validation circuit (see Fig. 5) applies NOT gates to q_1 and q_2, then uses a multi-controlled NOT gate to check if the values of q_0, q_1, q_2, q_3 correspond to `1001`. A similar approach is used to check for the `0110` pattern. While this method could theoretically be extended to check all legal solutions for larger values of n, as discussed in the previous section, such an approach would not be optimal for scalability.

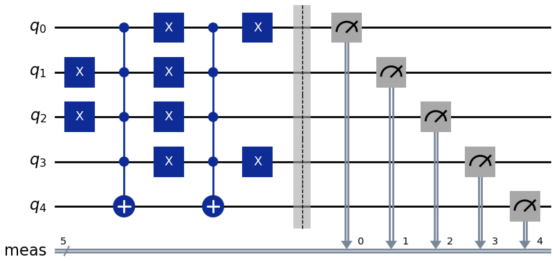

Fig. 5. Quantum circuit for validating permutation encodings. The circuit checks whether qubits q_0 through q_3 represent either `1001` or `0110`

4.4. Hardware performance results

The validation circuit, when transpiled for our quantum hardware, resulted in a circuit depth of 183 with the following gate composition: 133 rotation gates, 93 controlled-Z gates, 5 measurement operations, and 1 barrier. This substantial depth immediately highlighted the challenges of implementing our approach on current quantum hardware.

When testing the complete validation circuit (checking for both `1001` and `0110`), we achieved correct solutions approximately 13% of the time. For example, when providing the input `10010`, the expected output `10011` was obtained with this frequency, though the incorrect output `10010` appeared almost as frequently (with less than 1% difference in probability).

Interestingly, when implementing only half of the circuit (checking for `1001` but not `0110`), the accuracy improved to nearly 30%, and the input pattern `10010` no longer appeared significantly more often than other incorrect outputs.

4.5. Simplified 2-qubits approach

Recognizing that the validation circuit alone would consume most of our available quantum resources before even implementing the QAOA algorithm, we decided to reduce the problem scope further. Instead of using 4 qubits for permutation representation, we simplified to using only 2 qubits: 1 qubit to represent the position of the second location and 1 qubit to represent the position of the third location. It is important to note here that with reduced problem scope, there are simple improvements that could be made to improve performance in this instance, such as not using the extra qubit for permutation validation, or creating a custom mixer only visiting legal solutions. We do not do that as to maintain the same structure that could be used for larger n. That way, if there are improvements made in the accuracy of quantum computers, this approach can be directly translated and used for larger n.

If our approach were to include $d_{\pi(n),\pi(1)}$, both paths would have the same distance, as every distance in our distance matrix would be included, so we did not include returning to the first city in our calculations.

Case study. For this reduced problem, we defined a fixed distance matrix between three cities as follows:

$$D = \begin{bmatrix} 0 & 20 & 31 \\ 20 & 0 & 44 \\ 31 & 44 & 0 \end{bmatrix}.$$

Using this matrix, the total travel cost for visiting cities in the order $1 \to 2 \to 3$ is $D(1,2) + D(2,3) = 20 + 44 = 64$, and for the order $1 \to 3 \to 2$ it is $D(1,3) + D(3,2) = 31 + 44 = 75$.

We then constructed the corresponding QUBO objective based on the binary encoding of the two configurations. The optimal solutions yielded objective values of 64 for the `10` encoding and 75 for the `01` encoding.

The QUBO formula implemented in our QAOA approach was:

$$64 \cdot q_0 + 75 \cdot q_1 - 139 \cdot q_0 \cdot q_1 + 1000 \cdot q_2.$$

In this formulation, q_2 serves as a penalty term to ensure that q_0 and q_1 represent different values. Note that the $-139 \cdot q_0 \cdot q_1$ term is theoretically unnecessary, since q_0 and q_1 are already constrained to be different, but it is used to demonstrate how quadratic interactions between qubits are implemented in QAOA-based cost functions.

4.6. QAOA circuit implementation

We implemented two versions of the QAOA circuit for our simplified problem:

The single-loop implementation, shown in Fig. 6, provides a compact circuit with minimal depth. The double-loop version, illustrated in Fig. 7, includes additional operations to reverse the effects on q_2 (the penalty qubit), but maintains an appropriate circuit depth suitable for our hardware constraints. Note that we uncompute q_2 before applying cost terms to other qubits, therefore disentangling it. Given the small number of qubits in our simplified problem, more than two loops are not necessary for effective optimization and could be counterproductive, by increasing the depth and therefore error rate.

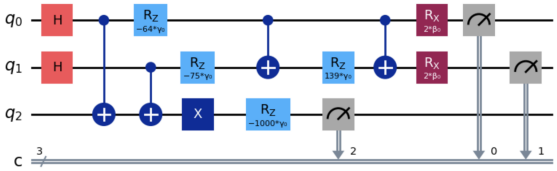

Fig. 6. QAOA circuit implementation with a single loop for the simplified 3-city TSP problem

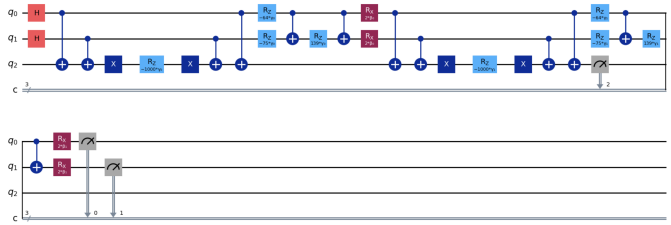

Fig. 7. QAOA circuit implementation with two loops for the simplified 3-city TSP problem

In the following section, we will demonstrate the process of optimizing the γ and β parameters for both single-loop and double-loop QAOA implementations and compare their performance results.

5. QAOA PARAMETER OPTIMIZATION AND RESULTS

This section presents the optimization process and results for both single-loop and double-loop QAOA implementations. We systematically explore parameter tuning and penalty term adjustments to achieve optimal performance on our quantum hardware.

5.1. Single-loop circuit optimization

To find the optimal values of the γ and β parameters for our single-loop QAOA implementation, we employed the COBYLA (constrained optimization by linear approximation) optimizer. This gradient-free optimization method is particularly well-suited for noisy quantum hardware where objective function evaluations can be stochastic.

The complete optimization code implementation is provided in Appendix A (Listing 2).

Initially, we ran the optimizer with the QUBO formulation stated above and obtained optimal parameters $\gamma = 0.5658132151939581$ and $\beta = 1.8766344358449518$. The results of this optimization are presented in the Fig. 8.

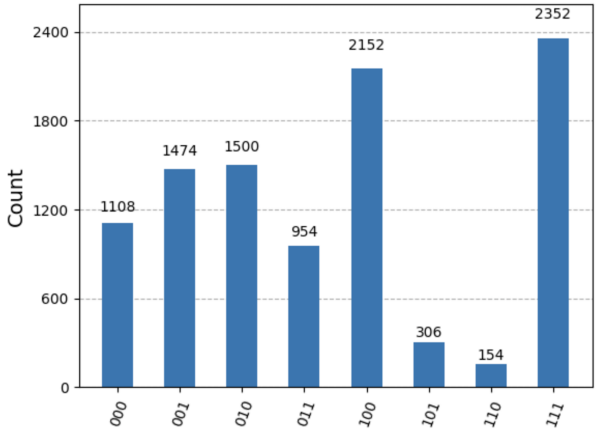

Fig. 8. Initial QAOA results with penalty term = 1000

While we can observe some trends in the results, the correct answer is not even the most common outcome. More concerning is the significant portion of answers where $q_0 = q_1$, which represents illegal solutions since both cities cannot occupy the same position in the tour.

To address this issue, we increased the penalty term from 1000 to 10 000 and re-ran the optimization (see Fig. 9).

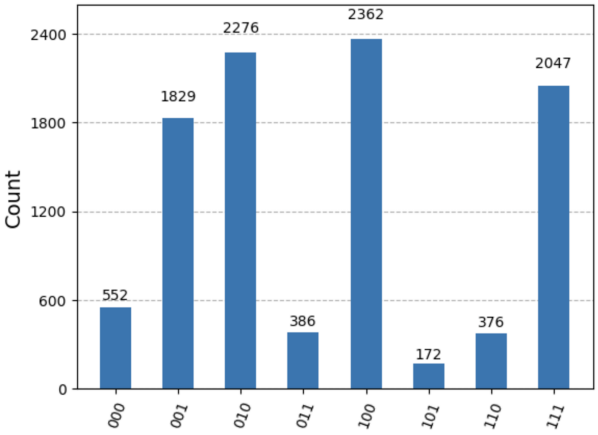

Fig. 9. QAOA results with increased penalty term = 10 000

The frequency of obtaining correct answers increased with the higher penalty term, but the performance remained unsatisfactory for practical applications.

We further increased the penalty term to 100 000 and obtained the results presented in Fig. 10.

Even with this substantial penalty increase, the performance remained poor.

These results demonstrate that there is only so much we can achieve by adjusting the penalty term alone. To further improve

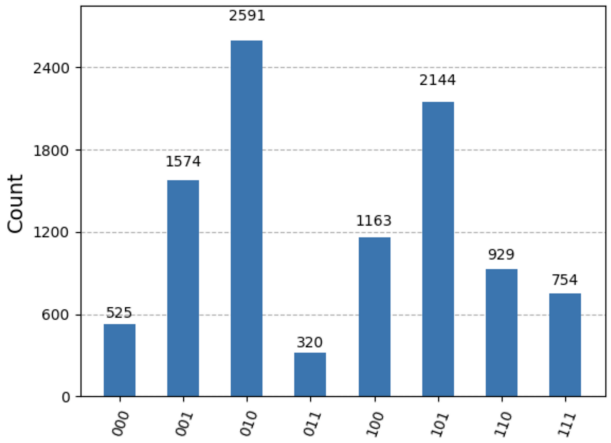

Fig. 10. QAOA results with penalty term = 100 000

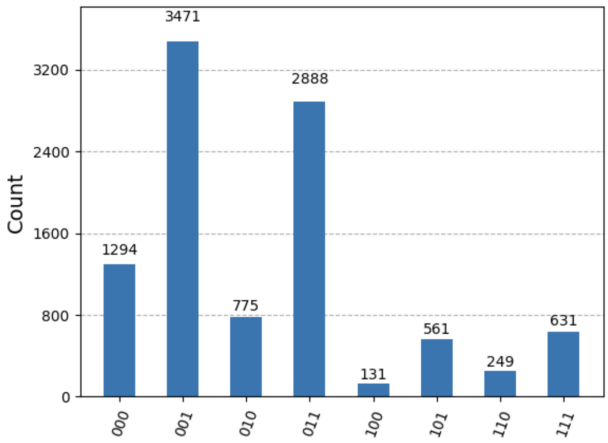

Fig. 12. Double-loop QAOA results with penalty term = 100 000

performance, we will now examine the results obtained by increasing the number of QAOA layers.

5.2. Double-loop circuit optimization

We now evaluate the performance of our double-loop QAOA implementation, which includes two layers of parameterized quantum gates. This approach provides additional variational parameters that may lead to improved optimization landscape exploration.

Starting with the same initial penalty term of 1000, the COBYLA optimizer yielded the following optimal parameters: $\gamma_0 = 0.41930916371319754$, $\beta_0 = 0.37772578545565016$, $\gamma_1 = 0.7024273809487542$, and $\beta_1 = 0.492951447891047$. The results showed immediate improvement compared to the single-loop implementation, see Fig. 11.

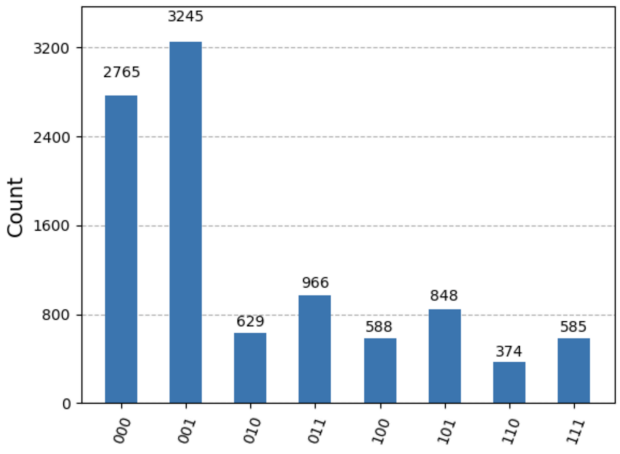

Fig. 11. Double-loop QAOA results with penalty term = 1000

Already with the initial penalty term, the correct answer became the most common outcome. However, the illegal solution `000` (where $q_0 = q_1 = 0$) appeared almost as frequently.

We then increased the penalty term to 100000, as in the single-loop case, and obtained the following optimization results, shown in Fig. 12.

With the increased penalty term, the correct answer `100` remained the most common outcome, but now there was a significant portion of outcomes showing `110`. The optimal parameters for this configuration were: $\gamma_0 = -0.3614353264797744$, $\beta_0 = 0.1161240675036641$, $\gamma_1 = 0.16969098639248045$, and $\beta_1 = 1.5365129015348746$.

Since this is our best outcome, we run the QAOA with this parameters 100 times to get a better idea of accuracy. Figure 13 shows the graph showcasing the outcome. Note that incorrect answer 110 is significantly less common, showing that our initial results were just slightly unlucky, and even then, the correct answer was the most common.

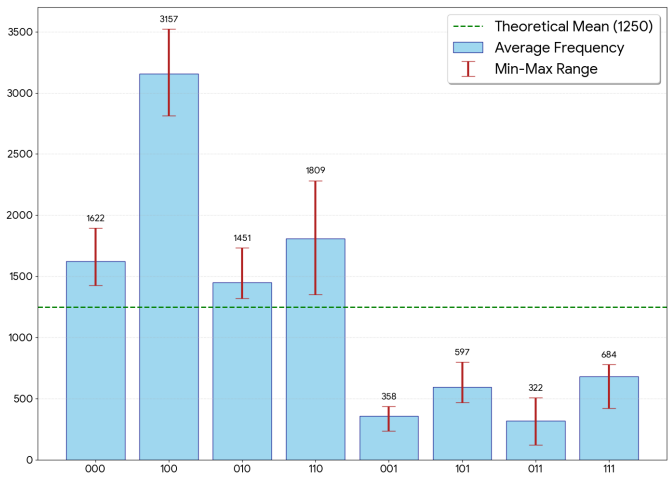

Fig. 13. Final results

5.3. Final remarks

Our experimental investigation of QAOA implementations for the traveling salesman problem reveals key insights into the interplay between circuit depth, constraint enforcement, and quantum hardware limitations.

The single-layer QAOA implementation, even with large penalty terms (up to 100 000), failed to produce a dominant correct solution. This highlights a limitation of shallow quantum

circuits: insufficient expressivity to effectively guide optimization. A three-layer QAOA circuit, not included in detail as the process was already shown in detail and three layer circuit did not improve results, reduced the occurrence of illegal solutions (with $q_2 = 1$), but did not show noticeable preference for the correct solution. This bias stems from small differences in the outcomes of the cost function between valid answers – a realistic trait of TSP instances.

Artificially increasing these differences in the QUBO formulation (e.g., assigning a cost of 2000 to one valid solution and 50 to the other) would improve outcome separation, but reduce practical relevance. Real-world problems such as the vehicle routing problem often involve subtle cost differences between feasible paths, and quantum algorithms must be able to distinguish such small variations.

The two-layer QAOA implementation struck the best balance. With increased sampling, it produced the correct outcome in over 30% of measurements, consistently across multiple runs. No other solution reached comparable frequency, and illegal outcomes remained rare. Notably, three other feasible solutions (with $q_2 = 0$) appeared at varying rates, sometimes approaching 25%.

These findings suggest that machine-induced noise was the primary cause of performance degradation at higher depths, while shallow circuits lacked sufficient optimization power. As quantum hardware improves in gate fidelity and coherence times, deeper QAOA circuits will become more viable and effective for combinatorial problems.

6. CONCLUSIONS

This work demonstrates that near-term quantum algorithms like QAOA can offer promising results for small-scale instances of the traveling salesman problem, provided a careful balance is maintained between circuit depth, constraint enforcement, and quantum hardware accuracy. We have used a 5-qubit 'ODRA-5' IQM Spark quantum computer installed in Wrocław University of Science and Technology since May 2025.

Our results show that while shallow circuits struggle with expressivity, deeper circuits are currently limited by hardware errors. However, the performance of our two-layer implementation suggests that QAOA can already yield meaningful results with today's devices, and will likely improve with future hardware advances.

Furthermore, the purpose of this work is partly to demonstrate a practical feasibility of a new, more efficient permutation representation method for quantum computers that would make solving larger problems viable. In the paper, we showed implementation of that on a 20-qubit IQM Radiance quantum computer, and in the near future, having access to bigger quantum computers (54 and 150-qubit), we aim to demonstrate the validity of this approach further.

The scalability of our encoding scheme, combined with effective constraint handling, lays the groundwork for applying quantum optimization techniques to real-world problems such as the vehicle routing problem. Continued progress in quantum hardware will be key to realizing this potential.

A. PROGRAM LISTINGS

```
from qiskit import QuantumCircuit, QuantumRegister,
    ClassicalRegister

# 19 qubits: 8 inputs + 6 comparisons + 5 OR tree
    nodes
qr = QuantumRegister(19, 'q')
cr = ClassicalRegister(9, 'c')  # 8 input bits + 1
    final OR result
protocol = QuantumCircuit(qr, cr)

# Initialize input bits (e.g., '10110001')
for i, bit in enumerate('10110001'):
    if bit == '1':
        protocol.x(qr[i])

# Define 2-bit comparison pairs: outputs to q[8]
    through q[13]
pairs = [
    ((0, 1), (2, 3), 8),
    ((0, 1), (4, 5), 9),
    ((0, 1), (6, 7), 10),
    ((2, 3), (4, 5), 11),
    ((2, 3), (6, 7), 12),
    ((4, 5), (6, 7), 13),
]

# 2-bit equality check -> sets output=1 iff inputs
    are equal
def compare_pair(circuit, a_bits, b_bits,
    output_bit):
    circuit.cx(qr[a_bits[0]], qr[b_bits[0]])
    circuit.cx(qr[a_bits[1]], qr[b_bits[1]])
    circuit.x(qr[b_bits[0]])
    circuit.x(qr[b_bits[1]])
    circuit.ccx(qr[b_bits[0]], qr[b_bits[1]],
    qr[output_bit])
    circuit.x(qr[b_bits[0]])
    circuit.x(qr[b_bits[1]])
    circuit.cx(qr[a_bits[1]], qr[b_bits[1]])
    circuit.cx(qr[a_bits[0]], qr[b_bits[0]])

# Compute all 6 pairwise equality results
for a_bits, b_bits, out_qubit in pairs:
    compare_pair(protocol, a_bits, b_bits,
    out_qubit)

# Build OR tree: q[14] - q[18]
# OR 1: q[8] OR q[9] -> q[14]
protocol.cx(qr[8], qr[14])
protocol.cx(qr[9], qr[14])
protocol.ccx(qr[8], qr[9], qr[14])

# OR 2: q[10] OR q[11] -> q[15]
protocol.cx(qr[10], qr[15])
protocol.cx(qr[11], qr[15])
protocol.ccx(qr[10], qr[11], qr[15])

# OR 3: q[12] OR q[13] -> q[16]
protocol.cx(qr[12], qr[16])
protocol.cx(qr[13], qr[16])
protocol.ccx(qr[12], qr[13], qr[16])

# OR 4: q[14] OR q[15] -> q[17]
protocol.cx(qr[14], qr[17])
protocol.cx(qr[15], qr[17])
protocol.ccx(qr[14], qr[15], qr[17])

# OR 5: q[17] OR q[16] -> q[18] (final output)
protocol.cx(qr[17], qr[18])
```

```
protocol.cx(qr[16], qr[18])
protocol.ccx(qr[17], qr[16], qr[18])

# Measure input bits
for i in range(8):
    protocol.measure(qr[i], cr[i])

# Measure final OR result
protocol.measure(qr[18], cr[8])
```

Listing 1: Permutation validation implementation

```
from qiskit import QuantumCircuit, QuantumRegister,
    ClassicalRegister, transpile
from qiskit.circuit import Parameter
from qiskit.visualization import plot_histogram
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# Backend and shots
shots = 10000
backend = backend

# Create QAOA circuit template
gamma0 = Parameter('γ0')
beta0 = Parameter('β0')
qr = QuantumRegister(3, 'q')
cr = ClassicalRegister(3, 'c')
qc_template = QuantumCircuit(qr, cr)

def apply_layer(qc, qr, gamma, beta):
    # Constraint computation: q2 = 1 iff q0 == q1
    qc.cx(qr[0], qr[2])
    qc.cx(qr[1], qr[2])
    qc.x(qr[2])
    # Cost unitary
    qc.rz(-gamma * 1000, qr[2])
    qc.rz(-gamma * 64, qr[0])
    qc.rz(-gamma * 75, qr[1])
    qc.cx(qr[0], qr[1])
    qc.rz(gamma * 139, qr[1])
    qc.cx(qr[0], qr[1])
    # Mixer unitary
    qc.rx(2 * beta, qr[0])
    qc.rx(2 * beta, qr[1])

# Initial state (superposition)
qc_template.h(qr[0])
qc_template.h(qr[1])

# Add QAOA layer
apply_layer(qc_template, qr, gamma0, beta0)

# Measurement
qc_template.measure(qr, cr)

# Cost function from bitstring
def bitstring_to_cost(bitstring):
    q0 = int(bitstring[2]) # qr[0]
    q1 = int(bitstring[1]) # qr[1]
    q2 = int(bitstring[0]) # qr[2]
    return 64 * q0 + 75 * q1 - 139 * q0 * q1 + 1000
    * q2

# Evaluate QAOA expectation value for given
    parameters
def evaluate_qaoa(gamma0_val, beta0_val):
    bound_qc = qc_template.assign_parameters({
        gamma0: gamma0_val, beta0: beta0_val
    })
    transpiled = transpile(bound_qc, backend)
    job = backend.run(transpiled, shots=shots)
    result = job.result()
    counts = result.get_counts()
    total_cost = 0
    for bitstring, count in counts.items():
        cost = bitstring_to_cost(bitstring)
        total_cost += cost * count
    avg_cost = total_cost / shots
    return avg_cost

# Objective function for optimizer
def qaoa_objective(params):
    gamma0_val, beta0_val = params
    return evaluate_qaoa(gamma0_val, beta0_val)

# Optimize gamma and beta
initial_guess = [0.5, 0.5]
res = minimize(qaoa_objective, x0=initial_guess,
    method='COBYLA', options={'maxiter': 60})
optimal_gamma0, optimal_beta0 = res.x

print("Optimal γ0:", optimal_gamma0)
print("Optimal β0:", optimal_beta0)

# Final evaluation with optimal parameters
final_qc = qc_template.assign_parameters({
    gamma0: optimal_gamma0, beta0: optimal_beta0
})
final_transpiled = transpile(final_qc, backend)
job = backend.run(final_transpiled, shots=shots)
result = job.result()
counts = result.get_counts()

# Display results
print("Final measurement counts:")
for bit, c in counts.items():
    print(f"{bit}: {c}")
plot_histogram(counts)
```

Listing 2: COBYLA optimization code

REFERENCES

[1] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014. [Online]. Available: https://arxiv.org/abs/1411.4028

[2] A. Glos, A. Krawiec, and Z. Zimborás, "Space-efficient binary optimization for variational quantum computing," *npj Quantum Inf.*, vol. 8, p. 39, 2022, doi: 10.1038/s41534-022-00546-y.

[3] A. Ciacco, F. Guerriero, and G. Macrina, "Review of quantum algorithms for medicine, finance and logistics," *Soft Comput.*, vol. 29, no. 4, pp. 2129–2170, 2025, doi: 10.1007/s00500-025-10540-z.

[4] N. Chancellor, "Domain wall encoding of discrete variables for quantum annealing and qaoa," *Quantum Sci. Technol.*, vol. 4, no. 4, p. 045004, 2019, doi: 10.1088/2058-9565/ab33c2.

[5] D. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[6] K. Helsgaun, "An effective implementation of the lin–kernighan traveling salesman heuristic," *Eur. J. Oper. Res.*, vol. 126, no. 1, pp. 106–130, 2000, doi: 10.1016/S0377-2217(99)00284-2.

[7] A. Lucas, "Ising formulations of many np problems," *Front. Phys.*, vol. 2, p. 5, 2014, doi: 10.3389/fphy.2014.00005.

[8] C. Papalitsas, T. Andronikos, K. Giannakis, G. Theocharopoulou, and S. Fanarioti, "A qubo model for the traveling salesman problem with time windows," *Algorithms*, vol. 12, no. 11, p. 224, 2019, doi: 10.3390/a12110224.

[9] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H.G. Katzgraber, "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer," *Front. Phys.*, vol. 7, p. 48, 2019, doi: 10.3389/fphy.2019.00048.

[10] S. Chander, "Comparative analysis of quantum approximate optimization algorithms for solving the travelling salesperson problem – revised with data and further analysis," *preprint Research Gate*, 2025, doi: 10.13140/RG.2.2.25958.54080.

[11] Y. Ruan, S. Marsh, X. Xue, Z. Liu, and J. Wang, "The quantum approximate algorithm for solving traveling salesman problem," *Comput. Mater. Continua*, vol. 63, no. 3, p. 1237–1247, 2020, doi: 10.32604/cmc.2020.010001.

[12] W. Qian, R.A.M. Basili, M.M. Eshaghian-Wilner, A. Khokhar, G. Luecke, and J.P. Vary, "Comparative study of variations in quantum approximate optimization algorithms for the traveling salesman problem," *Entropy*, vol. 25, no. 8, p. 1238, 2023, doi: 10.3390/e25081238.