Computing and Informatics, Vol. 40, 2021, 897–929, doi: 10.31577/cai_2021_4_897

VARIATIONAL ALGORITHMS FOR WORKFLOW SCHEDULING PROBLEM IN GATE-BASED QUANTUM DEVICES

Julia Plewa, Joanna Sieńko, Katarzyna Rycerz

AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: {juliadplewa, joannasienko1}@gmail.com, kzajac@agh.edu.pl

Abstract. In this paper we consider the combinatorial optimization problem known as workflow scheduling. We compare three encoding schemes of varying density: one-hot, binary, and domain wall, and test their performance against two well-known hybrid quantum-classical algorithms: Quantum Approximate Optimization Algorithm (QAOA) and Variational Quantum Eigensolver (VQE). In an attempt to obtain the best results possible, we investigate various parameters of the algorithms and test out other state-of-the-art improvements, such as dedicated QAOA mixers. Ultimately, we prove that, despite its popularity, one-hot encoding is not always the best, and using a denser encoding scheme, such as binary or domain wall, can allow for solving larger instances of workflow scheduling. Additionally, combining the above-mentioned encodings with dedicated QAOA mixers reduces the number of infeasible solutions, leading to better results.

Keywords: Hybrid quantum-classical algorithms, QAOA, VQE, optimization, workflow scheduling, one-hot encoding, binary encoding, domain wall encoding

Mathematics Subject Classification 2010: 68Q12

1 INTRODUCTION

With the growing popularity of quantum computers and algorithms, we are at an age when real-life applications of quantum computing are finally becoming feasible. Recent research in the field of quantum optimization has been quite promising, with achievements in fields such as biology [15, 18], resource distribution [8], machine learning [11], or finance [17]. In this paper, we focus on the popular combinatorial optimization problem that is workflow scheduling [4].

The goal of workflow scheduling is to assign a series of tasks to some available resources while meeting certain Quality of Service (QoS) requirements (such as a predefined time limit) and minimizing the overall cost. In our previous work, we formally defined workflow scheduling and solved it using the D-Wave quantum annealer, achieving quite promising preliminary results for small graphs of tasks fitting entirely on the computer architecture [28]. In this paper, we present the results for the same type of problem, but solved with alternative, variational algorithms designed for gate-based quantum devices. We focus on the two most popular algorithms: Variational Quantum Eigensolver (VQE) [20] and Quantum Approximate Optimization Algorithm (QAOA) [6]. In particular, the goal of this paper is to present and compare the possible improvements provided by variational algorithms. First, we investigate different problem encodings [10, 2] as more space-efficient methods of representing our problem. We also check the ability to limit the search space to the feasible subspace with QAOA mixing operators [12, 31]. We show that these methods allow us for encoding larger workflow problems and reduce the number of infeasible solutions.

The paper is organized as follows: in Section 2 we briefly describe the related work. Section 3 provides an overview of the variational algorithms used in this paper. The definition of the workflow scheduling problem can be found in Section 4. Next, in Section 5 we describe the selected encoding schemes and the appropriate QAOA mixing operators. Section 6 describes the experiment design and Section 7 presents the results. We conclude in Section 8.

2 RELATED WORK

Quantum optimization and scheduling are a recent and promising field of research, mainly focusing on the usage of the D-Wave quantum annealer. A possible method of solving the job shop scheduling problem using D-Wave is shown in [30], however, it has strong limitations in terms of scalability. A good example of a hybrid optimization method combining quantum and classical computers for this problem is presented in [13]. In our previous work we approached the solution of simple workflow scheduling problems using the D-Wave quantum annealer [29]. In this paper, we investigate an alternative variational approach for gate-based quantum devices that allows for experiments encoded in different ways, which can sometimes result in higher order cost functions. The architecture of gate-based devices, in contrary to the available quantum annealers, is not limited to quadratic problems. Our work was motivated by the recent results showing the trade-off between space efficiency and circuit depth [10, 7, 3]. Other approaches proposed in recent years include domain wall encoding [2, 3] and minimal encoding [27]. The introduction of dedicated QAOA mixing operators (also known as *mixers*) [12, 31] can be used to limit the configuration space to some feasible subspace in order to avoid configurations representing incorrect states. This approach can also be used in combination with different encoding schemes [2]. In our opinion, all presented solutions seem to be promising in solving optimization problems. To the best of our knowledge, the results of applying these methods to the workflow scheduling problem have never been discussed before. Therefore, in this paper, we present results of our research on that topic.

3 VARIATIONAL ALGORITHMS OVERVIEW

The Variatonal Quantum Eigensolver (VQE) algorithm [20] is a hybrid quantum-classical algorithm used to find the smallest eigenvalue of a given Hamiltonian and the corresponding eigenvector. Its main application is in solving large chemical problems, such as the problem of finding the ground state energy of molecules. VQE is an alternative to the QPE (Quantum Phase Estimation) algorithm [1], but with the advantage of smaller circuit depths, which is especially important for the current NISQ era of quantum computing. An overview of VQE is presented in Figure 1.

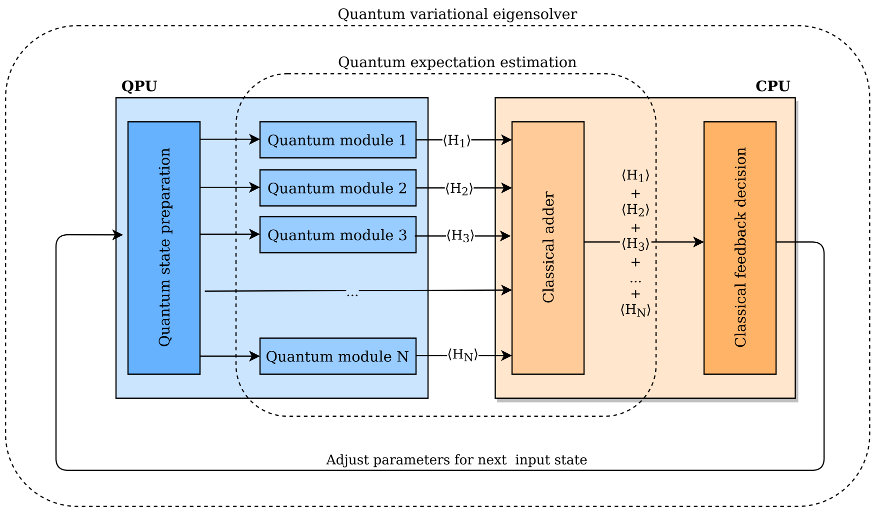

Quantum variational eigensolver
Quantum expectation estimation
QPU
CPU
Quantum state preparation
Quantum module 1
Quantum module 2
Quantum module 3
...
Quantum module N
⟨H1⟩
⟨H2⟩
⟨H3⟩
⟨HN⟩
Classical adder
⟨H1⟩ + ⟨H2⟩ + ⟨H3⟩ + ... + ⟨HN⟩
Classical feedback decision
Adjust parameters for next input state

Figure 1. An overview of VQE. The classical optimizer (marked with orange) adjusts the parameters of the quantum component (marked with blue) to minimize the energy of the system. The figure is based on [20] under the Creative Commons Attribution 4.0 license.

The main parts of the VQE algorithm are:

1. preparing the $|\psi\rangle$ state with the parametrized circuit (*ansatz*),
2. calculating the expectation value $\langle H_i \rangle$ of each Pauli term using separate quantum circuits (*quantum modules*) which consist of R_x and R_y rotations, allowing for measurements in the Z-basis,

3. adding all the expectation values on a classical computer,
4. optimizing the parameters of the ansatz using a classical routine, based on the expectation value sum.

The Quantum Approximate Optimization Algorithm (QAOA) [6] main application is to solve combinatorial optimization problems. QAOA relies on two operators: the cost Hamiltonian,

$$U(C,\gamma) = e^{-i\gamma C} = \prod_{\alpha=1}^{m} e^{-i\gamma C_\alpha}, \tag{1}$$

and the mixing Hamiltonian (or *mixer*),

$$U(B,\beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta \sigma_j^x} \tag{2}$$

where γ and β are angles, m is the number of constraints in the problem, and n is the number of qubits. The *initial state* $|s\rangle$ is the superposition over the computational basis states,

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_z |z\rangle. \tag{3}$$

Based on the above, a quantum state dependent on angles γ and β is defined as

$$|\gamma,\beta\rangle = U(B,\beta_p)U(C,\gamma_p)\ldots U(B,\beta_1)U(C,\gamma_1)\,|s\rangle \tag{4}$$

where p is a parameter describing the number of repetitions of the $U(B,\gamma_p)U(C,\beta_p)$ sequence, and thus the number of angles to be optimized. For a good approximation, the angles γ and β should be small and the algorithm should have a long running time, therefore a large p is expected. An overview of QAOA is shown in Figure 2.

To sum up, the main parts of the QAOA algorithm are:

1. preparing an initial state by applying the Hadamard gate on each qubit,
2. applying a sequence of $U(C,\gamma_i)U(B,\beta_i)$ gates p times,
3. measuring the circuit on a quantum computer in the computational Z-basis,
4. using a classical optimization subroutine to find the new angles $(\beta_1,\gamma_1,\ldots,\beta_p,$ $\gamma_p)$,
5. stopping the algorithm once the optimization objective is met.

4 DEFINITION OF WORKFLOW SCHEDULING PROBLEM

The groundwork for the formal definition of workflow scheduling used in this paper was laid in [29]. That definition focused specifically on one-hot encoding. In this section, we present a more general encoding-independent definition.

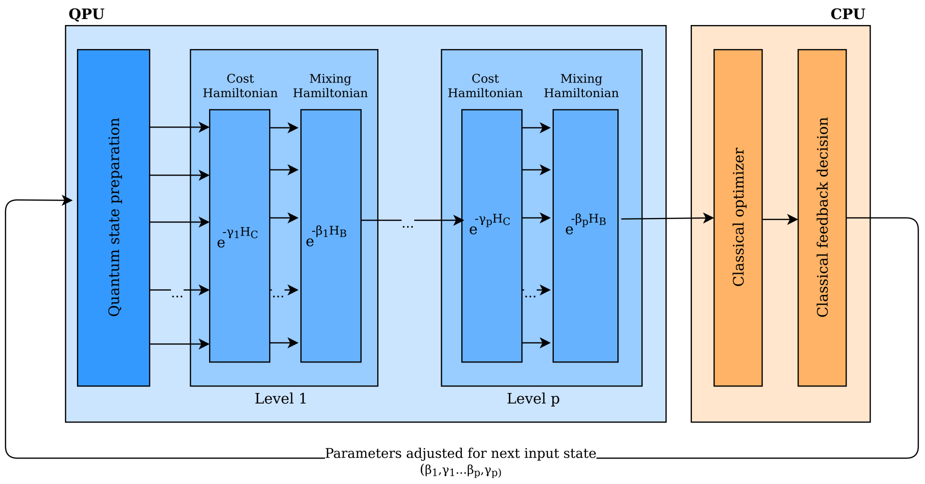


Figure 2. An overview of QAOA. The classical optimizer (marked with orange) adjusts the parameters of the quantum component (marked with blue) to minimize the energy of the system. The metaparameter p defines the length of the quantum circuit.

An instance of the workflow scheduling problem consists of N tasks and M types of machines (within each type, the number of machines is unlimited). The execution of a given task on a particular type of machine is associated with a specific cost and running time. The tasks have to be completed within a deadline d, while maintaining a specific order that is defined in the form of a directed acyclic graph (DAG). This graph needs to be decomposed into R paths that lead from the starting node to the final node, and each path has to be accounted for separately in the objective function.

Formally, the definition relies on the following constants:

- N – the number of tasks,
- M – the number of machines,
- R – the number of paths,
- d – the maximum number of time units allocated to the completion of all the tasks,

as well as the following matrices:

- $C(i,j)$ – the cost of executing task i on a machine of type j,
- $T(i,j)$ – the number of time units required to execute task i on a machine of type j,
- $S(k)$ – the number of slack variables for path k,

- $P(i,k)$ – a binary matrix of the form:

$$P(i,k) = \begin{cases} 1, & \text{if task } i \text{ is lies on path } k, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

The output of a quantum algorithm will typically assume the form of a one-dimensional binary vector. This vector can then be decoded and converted into two matrices:

- the solution matrix X, which has the form:

$$X(i,j) = \begin{cases} 1, & \text{if task } i \text{ is executed on a machine of type } j, \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

- the slack matrix Y, where $Y(k,l)$ denotes the value of the l^{th} slack on path k.

For simplicity, the elements of the above-mentioned matrices $C(i,j)$, $T(i,j)$, $S(k)$, $P(i,j)$, $X(i,j)$, and $Y(k,l)$ will be denoted as $c_{i,j}$, $t_{i,j}$, s_k, $p_{i,j}$, $x_{i,j}$, and $y_{k,l}$, respectively.

4.1 Objective Function

The objective of workflow scheduling is to minimize the cost of executing a series of tasks. We express it as[1]

$$O_{\text{cost}}(X) = \sum_i^N \sum_j^M c_{i,j} x_{i,j}. \tag{7}$$

4.2 Constraints

While our goal is to minimize the objective function, there are also certain constraints that have to be met, namely the time constraint and the encoding feasibility constraint.

4.2.1 Time Constraint

The time of execution does not have to minimized like the cost, but it does need to be kept below a deadline d, meaning we need to enforce the inequality

$$\sum_i^N \sum_j^M t_{i,j} x_{i,j} \leq d. \tag{8}$$

[1] The notation $\sum_i^N$ means a sum over the range $[0, N)$. Unless specified otherwise, it can be assumed that all sums in this paper exclude the upper bound.

The mechanism used to convert the inequality into an equality is based on *slack variables*. The slacks are additional non-negative variables in the solution vector that declare an offset added to the obtained value. Each path in the task ordering DAG has its own slack variables, since each path can complete in a different amount of time. For storing slack variables, we chose the binary encoding $\sum_l^{s_k} 2^l y_{k,l}$, where s_k is the number of bits needed to store the slack that corresponds to the k^{th} path. The equality takes the form

$$\sum_i^N \sum_j^M p_{i,k} t_{i,j} x_{i,j} + \sum_l^{s_k} 2^l y_{k,l} = d \quad \forall k \in R. \tag{9}$$

After converting it into a function, the constraint assumes the form

$$O_{\text{time}}(X,Y) = \sum_k^R \left(d - \left(\sum_i^N \sum_j^M p_{i,k} t_{i,j} x_{i,j} + \sum_l^{s_k} 2^l y_{k,l} \right) \right)^2 . \tag{10}$$

This function will assume the value of zero if the inequality from Equation (8) is satisfied, and otherwise the penalty grows quadratically.

4.2.2 Feasibility Constraint

Another constraint given in the workflow scheduling problem is the feasibility constraint, whose role is to make sure that each task is assigned to a valid machine type. This constraint is encoding-dependent, and it will be discussed in more detail in Section 5, while its specific implementations for each of the considered encodings can be found in [21].

4.3 Optimized Function

The full function takes the form

$$\begin{aligned} O(X,Y) = A \cdot \sum_i^N \sum_j^M c_{i,j} x_{i,j} + B \cdot \sum_k^R \left(d - \left(\sum_i^N \sum_j^M p_{i,k} t_{i,j} x_{i,j} + \sum_l^{s_k} 2^l y_{k,l} \right) \right)^2 \\ + C \cdot O_{\text{encoding}}(X) \end{aligned} \tag{11}$$

where A, B, and C are integer constants, to which we will later refer as *weights*. These weights correspond to execution cost, execution time constraint and feasibility constraint, respectively. The larger a weight for the given constraint is, the more we want our model to find solutions that satisfy it. Therefore, an important aspect of solving an optimization problem is to find such values for these weights that the optimal solution returns the smallest possible value of the optimized function.

5 ENCODING SCHEMES

Gate-model quantum machines are based on quantum bits (qubits), while many optimization problems involve discrete variables rather than binary. Such variables include integers but can include other problem representations as well, e.g. continuous variables or multiple mutually exclusive options. An example of a discrete variable in the workflow scheduling problem might be the index of a machine on which a certain task should be performed, as discussed in Section 4. In this section, we consider a specific example of such task-machine pairing, which is summarized in Table 1.

Task	Machine Type
0	0
1	3
2	2
3	1

Table 1. An example of task-machine pairing (note: the tasks and the machines are both indexed from 0)

Three different methods of encoding these discrete variables into qubits will be presented in this section. For each of those methods, we have to establish two functions:

- a binary function that takes an encoded bit string and validates if this string represents a specific category, e.g. "Does this bit string represent *Machine 2*?", by returning 1, if the answer is positive and 0, if it is not,
- a function that ensures a bit string contains a valid encoding – it should reach its minimum when faced with a valid encoding and return higher values otherwise.

Both of those functions are necessary for implementing the objective function and the constraints in most optimization problems. The specific objective and constraints needed for workflow scheduling were described in the previous section.

5.1 One-Hot Encoding

One-hot encoding is commonly used in different areas of research, including statistics, machine learning, or even digital circuits. It relies on the concept of *dummy variables*. A dummy variable can take the value of 0 or 1, indicating the absence or presence of some categorical quality. The example from Table 1 has been mapped to one-hot encoding in Table 2. Each row can contain only a single 1 throughout the five one-hot columns: *Run on M0*, *Run on M0*, *Run on M0*, *Run on M0*, and *Run on M0*. For consistency, let us collapse those columns back into one, as shown in Table 3. The placement of this single 1 indicates the category, i.e. the bit string 00001 can be translated to mean *Run on M4*. Any string that does not follow this pattern, such as 00000 or 01011, does not correspond to any valid state.

Task	Run on *M0*	Run on *M1*	Run on *M2*	Run on *M3*	Run on *M4*
0	1	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	1	0	0	0

Table 2. An example of one-hot-encoded task-machine pairing

Task	Machine Type
0	10000
1	00010
2	00100
3	01000

Table 3. An example of a one-hot encoded task-machine pairing, simplified

5.1.1 Bit String Interpretation

As mentioned at the beginning of this section, for every encoding we need to have a way of telling if a given state represents a specific category. The interpretation of one-hot-encoded strings is very straightforward. In order to evaluate if string s represents the i^{th} state, we simply have to look at the i^{th} bit in s, that is

$$f_{\text{one-hot}_i}(s) = s_i. \tag{12}$$

5.1.2 Bit String Feasibility

The second function needed for each encoding is used for penalizing invalid vectors. The function should reach its minimum, typically equal to 0, when applied to valid states, and otherwise it should return higher values. For one-hot encoding, this function is defined as

$$g_{\text{one-hot}}(s) = \left(1 - \sum_{i}^{n} s_i\right)^2 \tag{13}$$

where n is the number of states.

5.2 Binary Encoding

The idea behind binary encoding is very simple: each numerical category is encoded as a binary string, as shown in Table 4.

5.2.1 Bit String Interpretation

Although binary encoding seems straightforward at first, its interpretation is much harder than that of one-hot encoding. The function answering the question "Does

Task	Machine Type
0	$0_{10} = 000_2$
1	$3_{10} = 011_2$
2	$2_{10} = 010_2$
3	$1_{10} = 001_2$

Table 4. An example of a binary encoded task-machine mapping

this vector match the i^{th} machine type?" assumes a different form for each value of i. Its general form is defined as

$$f_{\text{binary}_i}(s) = \prod_{j}^{N} \left(1 - \left(s_j - b_j^{i,N}\right)^2\right) \tag{14}$$

where $N = \lceil \log_2 n \rceil$ is the number of bits in s and $b^{i,N}$ is the N-bit binary-encoded equivalent of i [10].

Considering our *Run on M3* example again, this function would assume the form

$$f_{\text{binary}_{3_{10}=011_2}}(s) = \left(1 - (s_0 - 0)^2\right)\left(1 - (s_1 - 1)^2\right)\left(1 - (s_2 - 1)^2\right), \tag{15}$$

which can be simplified to

$$\begin{aligned} f_{\text{binary}_{3_{10}=011_2}}(s) &= \left(1 - s_0^2\right)\left(1 - \left(s_1^2 + 2 - 2s_1\right)\right)\left(1 - \left(s_2^2 + 2 - 2s_2\right)\right) \\ &= \left(1 - s_0^2\right)\left(2s_1 - s_1^2\right)\left(2s_2 - {s_2}^2\right). \end{aligned} \tag{16}$$

Since s_0, s_1, and s_2 can only assume the values of 0 and 1, we can safely apply the identity $s_i = s_i^2$ and remove the squares, resulting in

$$f_{\text{binary}_{3_{10}=011_2}}(s) = (1 - s_0)s_1 s_2. \tag{17}$$

It is easy to notice that each part of this product assumes the value of 0 if the specific bit in vector s does not match the expected value, and otherwise it assumes the value of 1. Therefore, the result of this function is 1 if every bit matches the specific state and 0 if there is at least one bit that differs.

The functions for the remaining mappings from Table 4 are

$$\begin{aligned} f_{\text{binary}_{0_{10}=000_2}}(s) &= (1 - s_0)(1 - s_1)(1 - s_2), \\ f_{\text{binary}_{1_{10}=001_2}}(s) &= (1 - s_0)(1 - s_1)s_2, \\ f_{\text{binary}_{2_{10}=010_2}}(s) &= (1 - s_0)s_1(1 - s_2). \end{aligned} \tag{18}$$

5.2.2 Bit String Feasibility

A big advantage of binary encoding is that it is much *denser* than one-hot encoding. If the number of feasible states n is equal to 2^N, there are no states that are infeasible on the basis of encoding. Otherwise, if the number of states is not a power of two, we need to include a clause penalizing such invalid states.

In order to write this function, we need to compare the bit string in question to each of the infeasible states. In our example from Table 2, we established the five possible values for the *Machine type* column. The number of bits needed for this encoding is $N = \lceil \log_2 n \rceil$, which in our specific example is equal to 3. Since $2^3 = 8$, there are three states that are not made use of: 101, 110, and 111.

In order to penalize one of those states, we have to take all of the infeasible configurations specific to our problem, apply the formula established in Equation (14), and sum the results, which formally is defined as

$$g_{\text{binary}}(s) = \sum_{i \in W} f_{\text{binary}_i}(s) \tag{19}$$

where W is a set containing the infeasible states. This function will return a 1 for an infeasible state, and for any feasible state it will return a 0.

In our example, after applying the same simplifications as in Equation (17), this function would assume the form

$$\begin{aligned} g_{\text{binary}}(s) &= f_{\text{binary}_5}(s) + f_{\text{binary}_6}(s) + f_{\text{binary}_7}(s) \\ &= s_0(1-s_1)s_2 + s_0 s_1 (1-s_2) + s_0 s_1 s_2. \end{aligned} \tag{20}$$

5.3 Domain Wall Encoding

Unlike the previous two encodings, which are derived from classical computer science, domain wall encoding originates from physics [2]. *Domain walls* are an interface that occurs between neighboring magnetic domains, in which the magnetic moments of atoms gradually reorient themselves across a finite distance. In a domain wall encoded vector S, a wall occurs at index i if $s_{i-1} = 1$ and $s_i = 0$. Since this encoding derives meaning from pairs of neighboring bits rather than the bit values directly, it allows for the saving of one bit. Therefore, in order to encode n values, $n-1$ bits are needed. Additionally, we consider two virtual sentinel bits with fixed values: $s_{-1} = 1$ and $s_n = 0$.

The example from Table 1 is translated to domain wall encoding in Table 5. The two virtual sentinel bits are marked with a light gray color.

Task	Machine Type
0	100000
1	111100
2	111000
3	110000

Table 5. An example of domain wall encoded task-machine pairing, note: the far-left and far-right bits are constant

5.3.1 Bit String Interpretation

In order to determine if a given state represents a specific machine type, we need to calculate the difference of subsequent bits as defined in

$$f_{\text{domain-wall}_i}(s) = \begin{cases} 1 - s_0, & \text{if } i = 0, \\ s_i - s_{i+1}, & \text{if } 0 < i < N-2, \\ s_{N-1}, & \text{if } i = N-1. \end{cases} \tag{21}$$

Looking at the example from Table 5, for *Machine type 0*, we would look at the difference $1 - s_0$, whereas for *Machine type 1* we would look at the difference between s_1 and s_2. When this formula is used on a vector with only one domain wall, it returns either a 1, indicating that the represented machine type does indeed match the type in question, or a 0 if it does not. However, for a vector with more than one domain wall (an invalid domain wall vector), Equation (21) might also return the value of -1, which needs to be handled in a separate way, either by squaring the value of $f_{\text{domain-wall}_i}(s)$, or by excluding invalid vectors via another constraint or a QAOA mixer.

5.3.2 Bit String Feasibility

The number of walls in a domain wall encoded vector has to be odd. In a valid vector, there is exactly one wall. Therefore, to penalize invalid vectors, we can simply count the number of walls, as defined in

$$g_{\text{domain-wall}}(s) = (1 - s_0)^2 + \sum_{i}^{N-1} (s_i - s_{i+1})^2 + {s_{N-1}}^2. \tag{22}$$

For each encoding in Table 5, the above function would yield the value of 1 because there is only one domain wall, while for an infeasible solution, it would yield a greater value, e.g. for the state 1010, the number of walls is equal to 3.

5.4 Encoding and Workflow Scheduling

In order to implement the encoding-dependent objective function defined in Equation (11), two things need to be done. Firstly, the one-dimensional state returned by the quantum computer has to be translated into the X matrix from Equation (6). This is done via the f function defined for each encoding in Equations (12), (14), and (21) respectively. Similarly, the Y slack matrix also has to be *unflattened.* Secondly, as mentioned in Section 4.2.2, the encoding-dependent $O_{\text{encoding}}(X)$ clause is necessary to implement the full objective function. This formulation is directly related to the g penalty function defined for each encoding in Equations (13), (19), and (22) respectively – we simply need to apply this function to the encoding of each task's machine separately and sum the results. The specific implementations used in this paper are discussed in more detail in [21], along with the specific formulations of the QAOA mixers that are briefly described in the next subsection.

5.5 QAOA Mixers

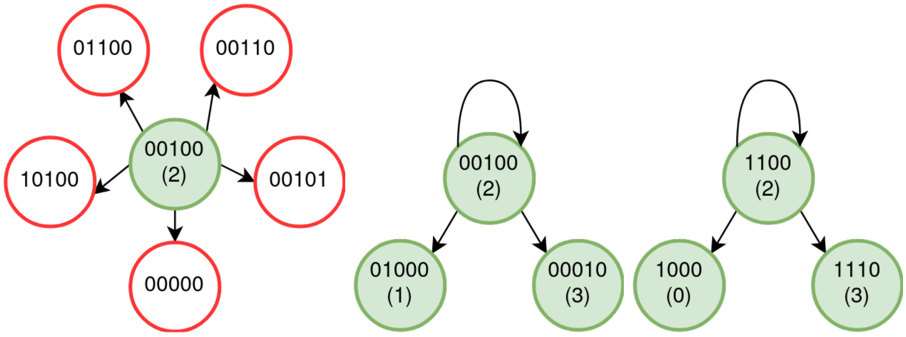
As described in Section 3, a QAOA circuit uses two operators: the cost operator defined in Equation (1), which depends on the optimization objective and the encoding, and the mixing operator defined in Equation (2). The idea behind the latter is to preserve the feasible subspace and to provide transitions between configurations that belong to this subspace.

The default mixing operator used in QAOA is the X mixer defined as

$$H_X = \sum_i^N X_i \tag{23}$$

where X_i is the Pauli X operator acting on the i^{th} qubit. This operator acts as a simple bit flip, allowing for transitions between any state and its neighbors [12].

Using the default QAOA mixer has its drawbacks, as this mixing strategy may cause the system to appear in an incorrect state. Figure 3 a) presents the state transitions performed by the X mixer originating from the valid state of 00100. None of the states produced by this mixer is valid. This could be handled via additional constrains or via post-processing, however, it can also be mitigated with a dedicated mixer.



a) One-hot encoding with X mixer
b) One-hot encoding with XY mixer
c) Domain wall encoding with dedicated mixer

Figure 3. State transitions depending on encoding and mixer, note: invalid states are marked with red

5.5.1 One-Hot Encoding

For one-hot encoding, the feasible subspace is quite limited. The only feasible states are the ones that include exactly a single 1. Instead of the default mixer, one-hot encoding is commonly used in combination with the dedicated XY mixer, defined as

$$H_{XY} = \sum_{i}^{N-1} (X_i X_{i+1} + Y_i Y_{i+1}) \tag{24}$$

where X_i and Y_i are Pauli operators acting on the i^{th} qubit [12, 31]. This operator works similarly to the SWAP gate – it turns the state $|01\rangle$ into $|10\rangle$ and vice versa, while the states $|00\rangle$ and $|11\rangle$ remain unchanged. The state transitions performed by this mixer can be seen in Figure 3 b). Unlike the graph in Figure 3 a), in this case all of the resulting states are valid.

5.5.2 Domain Wall Encoding

In a valid domain wall vector, there is exactly one wall. The dedicated mixer for domain wall encoding proposed by Chanellor [2] relies on this concept – given a valid state with exactly one wall, the mixer will only travel to states that are created by moving the wall by one bit in either direction. The mixer is defined as

$$H_{\text{mixer}_{\text{domain wall}}} = \sum_{i}^{N-1} (Z_{i-1}\ X_i - X_i\ Z_{i+1}). \tag{25}$$

This mixer flips the i^{th} qubit only when it is in direct neighborhood of a domain wall, i.e. if the values of the qubits at indices $i-1$ and $i+1$ are different, otherwise it does not introduce any change to the qubit. The mechanism is presented in Figure 3 c).

6 EXPERIMENT DESIGN

In this section, we discuss the design of our experiments. Firstly, we focus on the various experiment parameters, including the chosen algorithms and classical optimizers, the method used for initial point selection, and the selected QUBO (Quadratic Unconstrained Binary Optimization) parameters. We conclude the section by introducing the specific instances of workflow scheduling that will be used in our experiments.

6.1 Optimization Algorithms

In previous works focusing on quantum workflow scheduling, the main focus has been VQE [26]. We chose to also consider QAOA. The inclusion of QAOA opens up the door for including custom mixers in our computations. This lead us to arrive at the following combinations:

- QAOA with different mixers:
  - the default X mixer and encodings:

    * one-hot,
    * domain wall,
    * binary,
  - a custom encoding-dependent mixer and encodings:
    * one-hot (XY mixer),
    * domain wall (domain wall mixer),

- VQE with encodings:
  - one-hot,
  - domain wall,
  - binary.

6.2 Classical Optimizers

Although Qiskit offers numerous classical optimizers[2], previous research on workflow scheduling has focused on a single optimizer, specifically Spsa [26]. We chose to compare a few different optimizers to see whether any of them are preferable. The following optimizers were taken into account:

- Cobyla, gradient-free [22, 23, 24],
- Powell, gradient-free [25],
- Nelder-Mead, gradient-free [16],
- L-Bfgs-B, gradient-based [32].

Our choice of optimizers was backed by findings from literature, as well as personal experience. Cobyla and L-Bfgs-B have been found to be fast even in noisy environments [14]. Research also suggests that Spsa, Powell, and L-Bfgs-B are all effective even in noisy environments, while Cobyla and Nelder-Mead were found to handle noise worse [19]. Spsa was excluded from these tests due to its run time being very long.

6.3 Initial Point Selection

The importance of the initial point is an issue overlooked in literature and in previous research. We considered two approaches here: we could either look for a single point that renders the best result, or we could repeat the experiment enough times to be able to disregard the influence of randomization on our results. The first approach did not seem to be a good choice for two reasons. Firstly, finding a good initial point is a tedious and partly manual process. Secondly, reporting on a single best result is not fully honest and realistic. This point cannot be reused for other problems,

[2] `https://qiskit.org/documentation/apidoc/qiskit.aqua.components.optimizers.html`

and it does not reflect on the quality of the solution, since it is the result of reverse engineering.

6.3.1 Reusing Initial Points

While there is no indication that these initial points can be reused as the size of the problem grows, or that they are universally efficient regardless of the chosen classical optimizer, there is a way these values can be reused to improve one's results. The basic idea for both QAOA and VQE is that we can make more informed guesses for the initial point as the depth of the circuit grows. This approach is most often discussed for QAOA and its parameter p [10], however, it has also been suggested that it could be an efficient way to improve the results of VQE as the number of repetitions increases.

6.4 Initial State

The initial state is another parameter passed to QAOA. By default, the initial state is a superposition of all possible configurations, however, often it is preferable to replace that with a custom vector. Most notably, when custom QAOA mixers are used, it is necessary to pass the superposition of all the *feasible* states. This way, the mixer can remain in the feasible subspace and only consider valid vectors. In our case, this superposition contained all the correctly encoded machine combinations with every possible slack variable value.

6.5 QUBO Parameter Selection

The definition of workflow scheduling from Equation (11) includes three clauses. The first clause is the objective function, and the other two include penalties. Each clause has its respective weight, and by manipulating these weights we can alter the ordering of the energies of the possible configurations. This ordering has to follow two main principles:

- the optimal solution has to have the lowest energy,
- the energy of each correct solution has to be lower than that of an incorrect solution.

While these two principles generate a valid ordering, there are additional tweaks that can be implemented. Before we discuss them, let us define the following mutually exclusive types of solutions to the workflow scheduling problem:

- the optimal solution – a solution that represents a valid mapping which fits within the deadline and has the lowest possible cost,
- a correct solution – a solution that represents a valid mapping and fits within the deadline (**note**: this metric does not include the optimal solution),

- a semi-optimal solution – a solution identical to the optimal solution, but with an invalid slack configuration,
- a semi-correct solution – a solution identical to some correct solution, but with an invalid slack configuration,
- an incorrect solution – a solution that is not optimal, correct, semi-optimal, or semi-correct, meaning it either corresponds to an invalid configuration or it exceeds the deadline.

These metrics were specifically designed to add up to the number of total solutions (100 %) returned from the experiment. The reason for highlighting these *semi-* solutions is that they cannot really be considered to be correct, since their energies includes a penalty for a seemingly exceeded deadline. The deadline is not actually exceeded, as it is just the slacks that give this impression, but the algorithm has no way of distinguishing between them. While the *semi-* solutions are not really desired, they are still preferable to incorrect solutions – a *semi-* solution corresponds to an actual correct configuration, so from a practical perspective it is quite sensible.

In a perfect setting, we would want the optimal and semi-optimal solutions to have the lowest energies, followed by correct and semi-correct solutions. The energies of incorrect solutions should be higher, preferably with feasible configurations having energies lower than infeasible solutions. This ordering is presented in Figure 4.

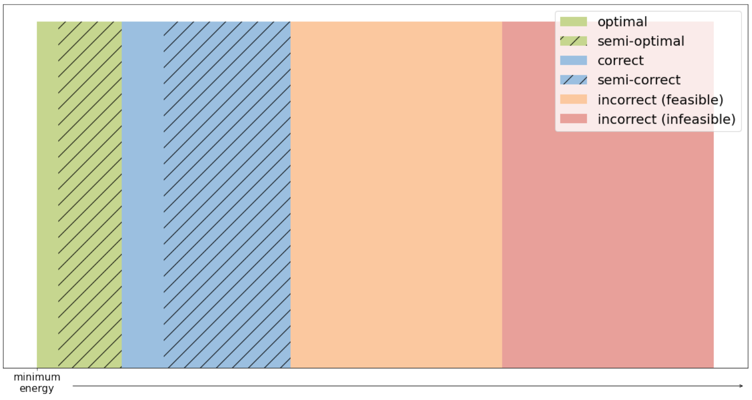


Figure 4. A one-dimensional visualization of the ideal ordering of solution energies

However, this ideal ordering is not possible due to the penalties affecting the *semi-* solutions, which make it impossible to distinguish between solutions that exceed the deadline and solutions that are semi-correct or semi-optimal. Therefore, we initially settled on an approach that shall be referred to as the *naive ordering*, in which we do guarantee that the energies of optimal and correct solutions are the lowest, however, all other solutions remain mixed with each other. This ordering is shown in Figure 5.

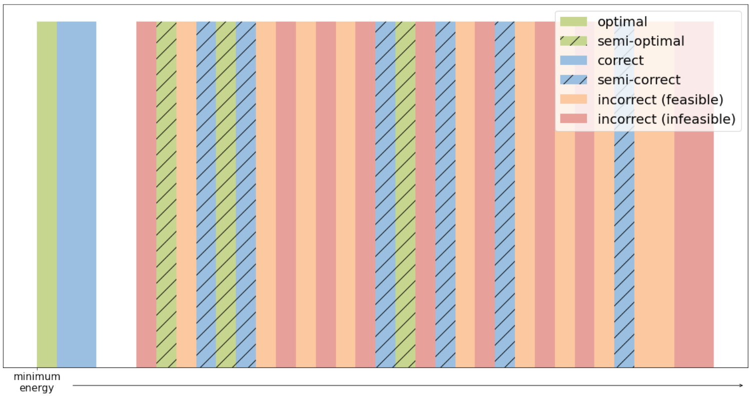


Figure 5. A one-dimensional visualization of the *naive ordering* of solution energies

Eventually, we tried further manipulating the A, B, C weights from Equation (11) to increase the gap between the correct solutions and the rest, but this was not particularly effective. Finally, we settled on an ordering, which will be referred to as the *feasibility-jump ordering*. This ordering guarantees the lowest energies for optimal and correct solutions and the highest energies for incorrect and infeasible solutions. In the middle, we have a mix of *semi-* solutions as well as incorrect feasible solutions. This ordering is illustrated in Figure 6.

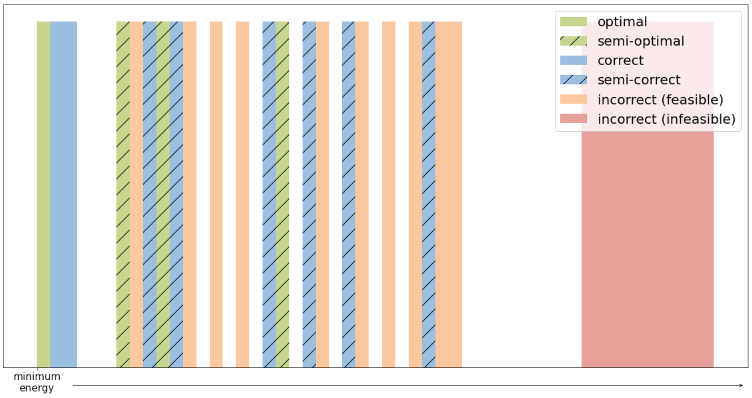


Figure 6. A one-dimensional visualization of the *feasibility jump ordering* of solution energies

This ordering was possible due to the feasibility constraint from Equation (11) having a separate weight, C. By sending those infeasible configurations away from the rest, we found some improvement in the results. In Section 7 we will discuss the results for one-hot encoding for two sets of weights implementing the naive ordering and the feasibility-jump ordering.

6.6 Result Evaluation Metrics

Different metrics can be used for evaluating the obtained results. A good metric is necessary not only for assessing the quality of a solution, but also when using the p-reusing trick described in Section 6.3. In this method, one needs to select the best solution for a specific p and then the optimal point of this solution is used as the initial point for $p + 1$.

The following metrics were taken into consideration:

- the number of correct/optimal/feasible solutions,
- the average energy between all solutions,
- the energy of the most frequent solution,
- the highest energy from among the found solutions,
- the lowest energy from among the found solutions.

Finally, we settled on the "average energy between all solution" approach, although this metric is certainly not perfect, as the standard deviation also appears to affect the results tremendously.

6.7 Considered Workflows

Three different workflows were considered: a single small problem to be tested against each encoding, and two large problems designed to test the capabilities of the two denser encodings.

The small problem consists of three machines and three tasks. Its DAG is shown in Figure 7 a). This problem required 13 qubits for one-hot encoding and 10 qubits for binary and domain wall encoding.

Two larger problems were used for tests of binary encoding and domain wall encoding. Both problems consisted of 4 tasks and both used the same DAG (shown in Figure 7 b)), however, each used a different number of machines and different cost and time matrices. The problem used to test binary encoding consisted of 4 tasks and 4 machines. It required 7 slack variables and 15 qubits overall. The problem used for domain encoding was slightly smaller due to the encoding being less dense – it contained 3 machines instead of 4. The number of slack variables needed for this problem instance was 6 and the total number of qubits was 14.

The full definitions of all three problem instances can be found in [21].

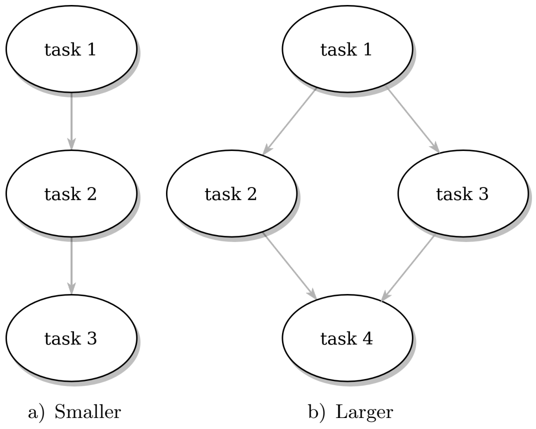


Figure 7. Solved problems

6.8 Implementation and Hardware

In our implementation, we used the open-source SDK for working with quantum computers, Qiskit[3]. The most important components used in this work were the implementations of VQE and QAOA, as well as those of the classical optimizers described in Section 6.2. To run the code, we used Qiskit's quantum simulation feature and in some cases we also relied on their quantum computer noise models.

Our implementation involved modelling the QUBO formulation of the considered problem in each of the three encodings. The Qiskit implementations of VQE and QAOA both accept the Ising Hamiltonian form of the optimized problem, therefore it was necessary for us to transform the QUBO formulation into an Ising Hamiltonian. Qiskit provides such a functionality, but only for functions of order that is at most quadratic, and since our implementation for binary and domain wall encoding required objective functions of higher orders (as can be seen in Section 5), we implemented our own small library for transforming QUBO's into Ising Hamiltonians. Furthermore, the QAOA mixing operators for one-hot and domain wall encoding were also implemented on our own.

The experiments were conducted on the Prometheus supercomputer[4] (the HP Apollo 8000 platform) at the Academic Computer Centre Cyfronet AGH with the usage of the PLGrid infrastructure.

[3] `https://qiskit.org`
[4] `https://www.cyfronet.pl/en/computers/15226,artykul,prometheus.html`

7 EVALUATION OF THE RESULTS

In this section, we describe the results obtained from the experiments described previously. For each of the three encodings, the QAOA and VQE algorithms were both tested. In addition, for the encodings with a dedicated mixer available, QAOA was tested with a custom and a default mixer separately. In our case, only binary encoding did not have a custom mixer, which means that we tested eight combinations of encodings and algorithms. Then, for each such pair, we tested four different optimizers, and for each *encoding-algorithm-optimizer* triple, we repeated the experiment with the *p*/*reps* parameters ranging from 1 to 3.

Therefore, for the smaller problem instance we ran $8 \cdot 4 \cdot 3 = 96$ experiments, while for the larger problem, since one-hot encoding was omitted, we ran $5 \cdot 4 \cdot 3 = 60$ experiments. Each experiment was repeated 1000 times to get the most reliable averages. Additionally, it is worth mentioning that every repetition measures the quantum circuit 1024 times, which is defined via the *shots* [5] parameter.

7.1 Smaller Problem

The smaller problem instance can be seen in Figure 7 a).

7.1.1 QAOA

In order to make the encoding comparison plots easier to read, the QAOA algorithm is compared in separate plots (Figure 8), regardless of the mixer used, and the VQE algorithm is compared in separate plots (Figure 9). The results obtained using the same optimizer are labelled with one color. For each optimizer used with a given encoding, there is only a single box plot drawn, as the results for all *p*/*reps* parameters are merged together. Sample outliers were also removed to make the plots more readable.

The performance of the QAOA algorithm for various encodings is shown in Figure 8. The percentage of optimal solutions was not used here, as it was not satisfactory, and it is easier to draw conclusions from the other solution metrics. The analysis of correct solution percentages was also not presented, as it did not provide any additional valuable information.

When comparing both the incorrect and the feasible solution percentages for QAOA with a default mixer, it can be concluded that **one-hot encoding – which is the most popular encoding used in literature for solving optimization problems – performed the worst out of the three possible encodings**. The newly proposed domain wall encoding and the well-known, yet less popular, binary encoding performed significantly better, e.g. for the COBYLA optimizer, the percentage of feasible solutions increased around 10 times when compared to one-hot encoding. The difference between the performance of domain wall and binary encoding used with a default mixer was not as significant.

[5] `https://qiskit.org/documentation/apidoc/execute.html`

Applying a custom mixer to the QAOA algorithm resulted in significantly decreasing the percentage of incorrect solutions. In this variant, the performance of one-hot encoding and domain wall encoding is comparable, so it can be concluded that one-hot encoding should be always used with a custom mixer, as it uses the largest number of qubits out of the three compared encodings, and therefore it has many infeasible states. What is worth noting, is that, **when used with custom mixers, both one-hot and domain wall encoding outperformed binary encoding**.

The most important conclusion that can be drawn from analyzing the percentage of feasible solutions is that the custom mixer implemented in the problem actually worked and therefore, when used, all the results are feasible.

7.1.2 VQE

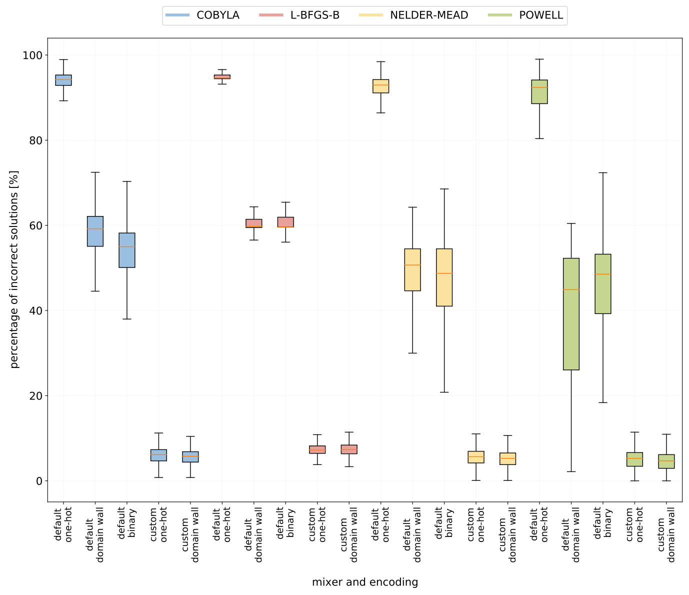
The incorrect solution percentages and correct solution percentages, as obtained by the VQE algorithm using different encodings are presented in Figure 9. When considering the performance of VQE, it can be again seen that one-hot encoding performed the worst, while domain wall and binary encoding both performed better. The biggest difference between the two can be seen for the Nelder-Mead optimizer, for which the median of incorrect solution percentages decreased from around 95 % to around 50 % for domain wall encoding, and even to around 30 % for binary encoding.

An incorrect solution percentage analysis can lead to a conclusion that domain wall and binary encoding performed similarly, however, when comparing the correct solution percentages, it turns out that binary encoding performed better. For each of the four optimizers, the median of correct solution percentages for binary encoding was higher than for domain wall encoding. The biggest difference can be seen for the Powell optimizer, for which the percentage of correct solutions raised from around 7 % to 12 %.

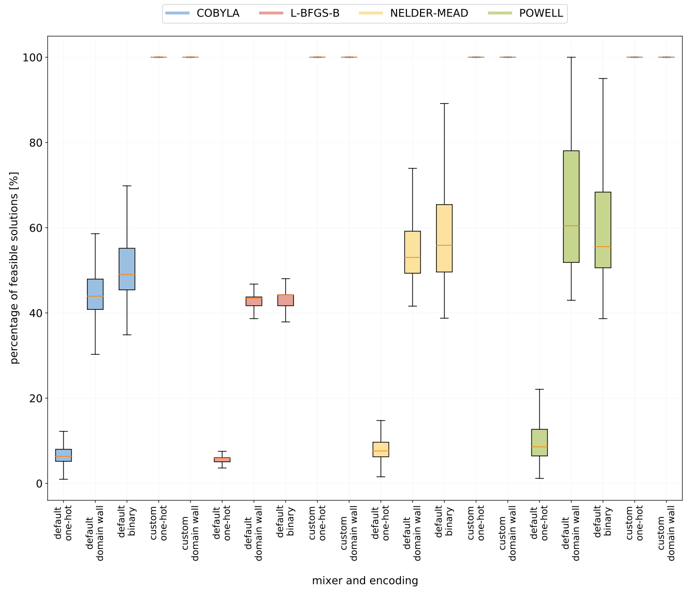
To sum up the results for the smaller problem instance, for QAOA it is domain wall encoding with a custom mixer and the Powell optimizer that performed the best – with a median of incorrect solutions at around 5 % and a median of correct solutions at around 7 %. Notably, one-hot encoding with a custom mixer performed only slightly worse. For VQE, it was definitely binary encoding with the Powell optimizer that performed the best – with the median of incorrect solutions at around 0 % and the median of correct solutions at around 12 %. Thus, it can be observed that the **VQE algorithm performed better than QAOA** and that **applying a custom mixer for a specific encoding in QAOA yields better results**.

7.2 Larger Problem

The larger problem instance can be seen in Figure 7 b). It was designed to fit on the publicly available 15-qubit Ibmq_16_Melbourne machine. For one-hot encoding,

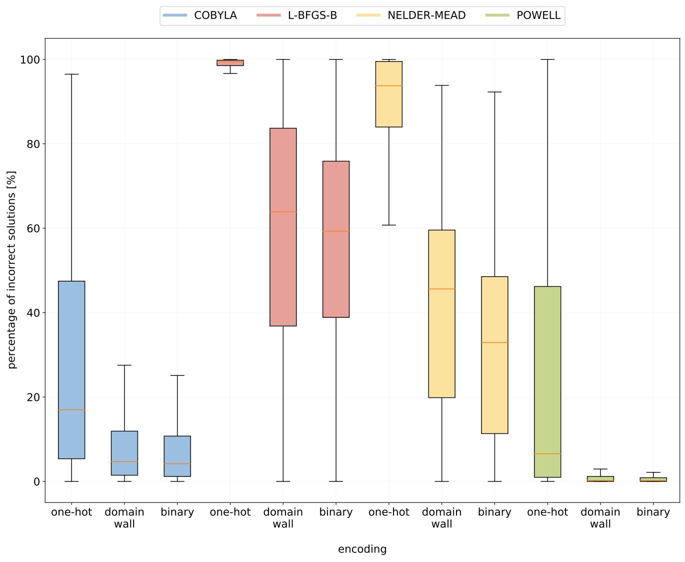


a) Incorrect solutions

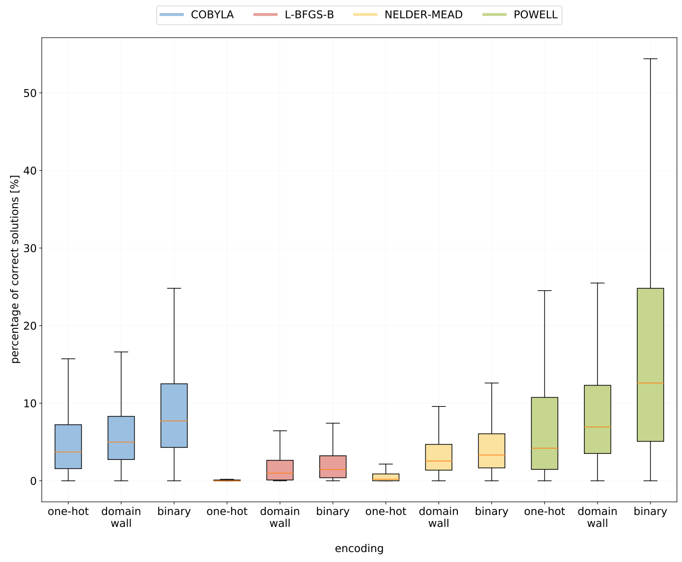


b) Feasible solutions

Figure 8. Smaller problem instance: An incorrect and feasible solution comparison between the three encodings with QAOA



a) Incorrect solutions



b) Correct solutions

Figure 9. Smaller problem instance: An incorrect and correct solution comparison between the three encodings with VQE

it was impossible to design a larger problem instance, since the small problem already occupied 13 qubits.

7.2.1 QAOA

A comparison of incorrect solution percentages for QAOA with both the default and the custom mixer with binary and domain wall encoding is presented in Figure 10 a). The first thing to notice is that for QAOA with a default mixer in every case it was binary encoding that performed better, e.g. for L-Bfgs-B the median percentage of incorrect solutions dropped by about 10 %.

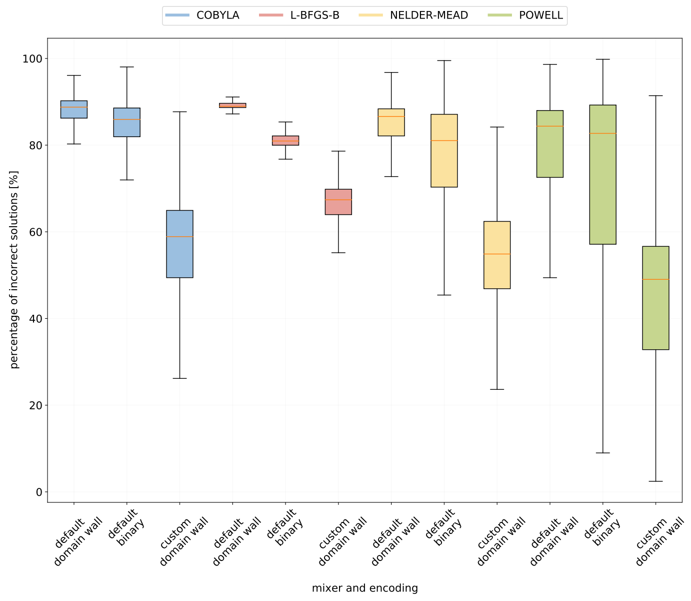
Next, it can be seen that **using a custom mixer for QAOA with domain wall encoding significantly improved the results**. For example, for the Powell optimizer, the median of incorrect solutions decreased by about 35 % when compared to binary encoding. The same relationship can be observed for all the optimizers used. Compared to the smaller problem, the larger problem produced worse results, which is because the problem itself uses a large number of qubits and is therefore more difficult to solve.

7.2.2 VQE

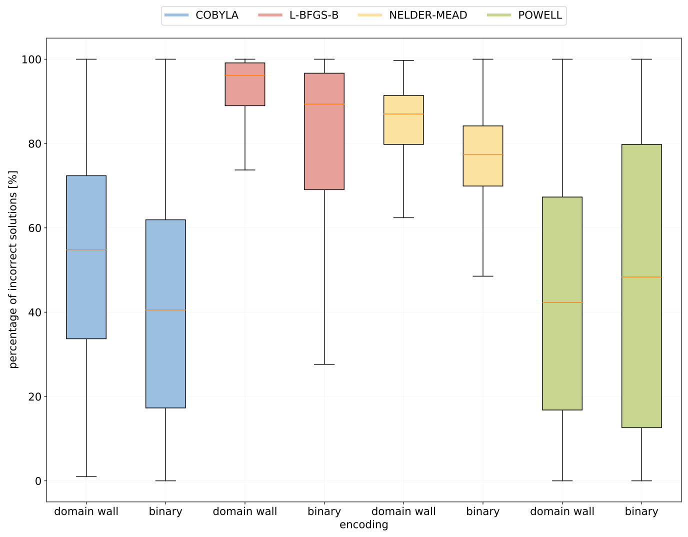
The results obtained by VQE are presented in Figure 10 b). Most importantly, VQE resulted in fewer incorrect solutions than QAOA (regardless of the mixer used) for the Cobyla and Powell optimizers. This confirms a similar conclusion made in the previous section that **VQE is strongly optimizer-dependent, as the L-Bfgs-B and Nelder-Mead optimizers performed worse than in QAOA, which clearly implies that they are not sufficient for VQE**.

Comparing the performance of binary and domain wall encoding with the VQE algorithm, we find that for the Cobyla, L-Bfgs-B, and Nelder-Mead optimizers, binary encoding produced fewer incorrect solutions (see Figure 10 b)), while the opposite is true for the Powell optimizer. It is noteworthy that for Cobyla and Powell, the results span across the 0 %–100 % range.

To conclude, for the larger problem instance, the VQE algorithm with binary encoding and Cobyla optimizer obtained the highest number of correct solutions (median of 0.3 %) and at the same time the lowest number of incorrect solutions (median of 40 %). An analysis of correct solution percentages – the corresponding graphs of which can be found in [21] – leads us to believe that QAOA with a custom mixer obtained a similar number of correct solutions for domain wall encoding, however, VQE performed better in terms of the number of incorrect solutions. Nevertheless, this conclusion could indicate that **QAOA may have more potential in larger, more real-world problems**.



a) QAOA



b) VQE

Figure 10. Larger problem instance: An incorrect solution comparison between the two encodings and two algorithms

7.3 Objective Function Weight Selection

As mentioned in previous section, we considered two variants of energy ordering: the *naive ordering* and the *feasibility jump ordering*. In the previous parts of this section, we presented the results for the latter, as we found that this ordering typically produced better results. This difference was most prominent when considering VQE.

A comparison of incorrect solution percentages between the two orderings for one-hot encoding can be seen in Figure 11. **It can be observed that the *feasibility jump ordering* resulted in a significant decrease of incorrect results, especially when considering COBYLA and POWELL.**

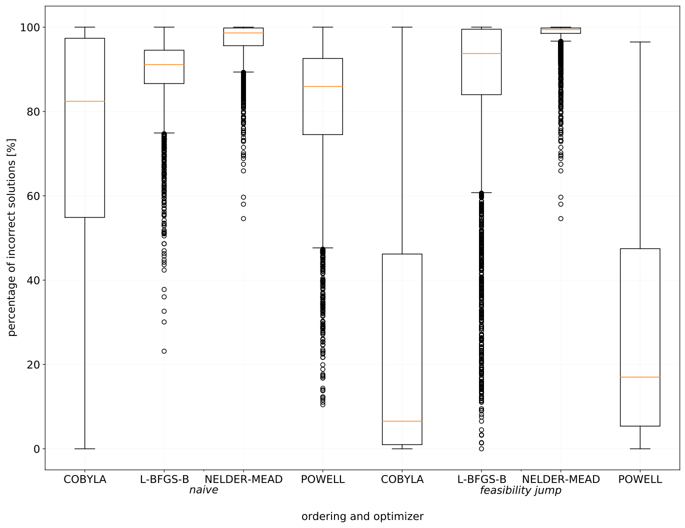


Figure 11. The difference in incorrect solution percentages for different orderings with one-hot encoding and VQE algorithm for the smaller problem instance

8 CONCLUSIONS AND FUTURE WORK

In today's quantum computing, we are limited to noisy near-term quantum devices with a relatively small number of qubits. Although the true potential of quantum computing has not yet been reached, many researchers focus on trying to improve the existing algorithms and methods, in order to prepare for the hardware of the future.

8.1 Achieved Goals and Observations

The goal of this paper was to research various methods of improving the existing solutions of workflow scheduling. Additionally, we also wanted to compare different problem representations with the hope that they would allow us to solve larger problems.

8.1.1 General Findings

While working on this paper and designing our experiments, we made many general observations. They are as follows:

- Using encoding methods alternative to one-hot encoding allows for encoding larger problems.
- The inclusion of slack variables severely complicates the objective function and leads to the introduction of *semi-correct* and *semi-optimal* states. Those states are very hard for the optimizer to handle, because they are penalized in the same way as states that break the time constraint. Additionally, the number of possible configurations increases majorly when slack variables are introduced, meaning for each valid state there are several versions of it that are penalized. With such numerous configurations, it becomes harder and harder to get to the optimal solution.
- Denser encodings, such as binary encoding, limit the number of infeasible configurations, which in some cases can lead to better results. In some specific situations, binary encoding can even limit the number of infeasible configurations to zero.
- Selecting good weights for the objective function and constraints is crucial. Introducing energy jumps between feasible and infeasible states leads to an improvement in the results.
- The inclusion of QAOA mixers allows the algorithm to disregard infeasible configurations, which in turn leads to an improvement of the simulation results. However, in our experiments, the noisy simulation results were not as satisfactory, which might suggest that the additional noise generated by the inclusion of a custom mixer might counteract any real benefits of the mixer [21].

8.1.2 Problem-Specific Findings

In Section 7, we presented our results in detail and discussed their importance. Those observations can be summarized as follows:

- For the problems solved in the paper, it was the VQE algorithm that performed better, but for the bigger problem instance the difference between VQE and QAOA is not as significant. This may suggest that QAOA has more potential for larger problem instances.
- Increasing the p parameter used by QAOA usually improved the results only when it was increased to 2. When increased to 3, the results deteriorated. For VQE, any increase in the *reps* parameter resulted in worse results. This trend was observed for both smaller and larger instances of the problem. Although it is

not presented in this paper, we also made attempts to increase these parameters further, however, we were not able to improve our results.

- The VQE algorithm is highly dependent on the chosen classical subroutine, and the best results were obtained when the COBYLA or POWELL optimizers were used. For QAOA there are no such discrepancies between optimizers.
- The use of custom mixers allowed us to reduce the complexity of the objective function (see Section 5.5). This performed well as a perfect simulation, however, with the inclusion of noise, there were no real benefits from the mixer.

8.2 Future Work

In this work, we focused on exploring several aspects of quantum optimization, but there are still many other approaches that could be tested.

Firstly, we believe that the inclusion of slack variables is worth reexamining, since those variables increase the complexity of the circuit and the number of qubits. Other methods of encoding inequality constrains should be considered, such as the Alternating Direction Method of Multipliers, based on augmented Lagrangians [9].

Another interesting direction for further research would be in finding better ways of determining the weights used for the objective and the constraints. As shown in Section 7, modifying those weights influences the results visibly, therefore some specific paradigm for choosing *good weights* should be established.

Thirdly, since we did not observe any noticeable improvement in the results of QAOA after increasing the p parameter, we believe that this method should be explored further. In this paper, we only included the results for low values of p, however, we initially also conducted experiments for much larger values and those experiments did not return promising results. Other researchers report noticeable improvements for growing values of p [10], which leads us to believe that our methodology might be worth reevaluating. A good starting point might be to reconsider the metric used for selecting the best angles from a number of runs. Some ideas for different metrics were discussed in Section 6.6. In this paper, we chose to focus on the lowest average energy metric, however, it is likely that other metrics would perform better. Similarly, we experimented with modifying the *reps* parameter passed to VQE, and we did not observe any clear correlation between this value and the quality of the results.

While our research focused mainly on improving the results obtained on a simulator, in the future it will be necessary to guarantee reasonable results on a real quantum computer as well. The first step in this exploration is the inclusion of artificial noise models which serve as an indication of how a specific implementation might perform under decoherence. Some of the techniques tackled in this paper seemed to perform well as a perfect noiseless simulation, however, the inclusion of noise severely affected the results. It would be particularly valuable to reexamine the performance of QAOA mixers to evaluate if their performance outweighs the potential implementation cost. Ultimately, in order to minimize the influence of decoherence on a real quantum computer, it would be a good idea to explore some error mitigation techniques [5].

Acknowledgements

We would like to thank Dr. Hiroshi Horii from IBM for his willingness to support us through technical difficulties, as well as Dr. Adam Glos from IITiS for his eagerness to share his work and discuss ideas. We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team. This research was supported in part by PL-Grid Infrastructure. The authors are grateful for support from the subvention of Polish Ministry of Education and Science assigned to AGH University of Science and Technology (Faculty of Computer Science, Electronics and Telecommunications).

REFERENCES

[1] Aspuru-Guzik, A.—Dutoi, A. D.—Love, P. J.—Head-Gordon, M.: Simulated Quantum Computation of Molecular Energies. Science, Vol. 309, 2005, No. 5741, pp. 1704–1707, doi: 10.1126/science.1113479.

[2] Chancellor, N.: Domain Wall Encoding of Discrete Variables for Quantum Annealing and QAOA. Quantum Science and Technology, Vol. 4, 2019, No. 4, Art. No. 045004, doi: 10.1088/2058-9565/ab33c2.

[3] Chen, J.—Stollenwerk, T.—Chancellor, N.: Performance of Domain-Wall Encoding for Quantum Annealing. IEEE Transactions on Quantum Engineering, Vol. 2, 2021, pp. 1–14, doi: 10.1109/tqe.2021.3094280.

[4] Deelman, E.—Gannon, D.—Shields, M.—Taylor, I.: Workflows and e-Science: An Overview of Workflow System Features and Capabilities. Future Generation Computer Systems, Vol. 25, 2009, No. 5, pp. 528–540, doi: 10.1016/j.future.2008.06.012.

[5] Endo, S.—Benjamin, S. C.—Li, Y.: Practical Quantum Error Mitigation for Near-Future Applications. Physical Review X, Vol. 8, 2018, No. 3, Art. No. 031027, doi: 10.1103/PhysRevX.8.031027.

[6] Farhi, E.—Goldstone, J.—Gutmann, S.: A Quantum Approximate Optimization Algorithm. 2014, arXiv: 1411.4028.

[7] Fuchs, F. G.—Kolden, H. Ø.—Aase, N. H.—Sartor, G.: Efficient Encoding of the Weighted MAX k-CUT on a Quantum Computer Using QAOA. SN Computer Science, Vol. 2, 2021, Art. No. 89, doi: 10.1007/s42979-020-00437-z.

[8] Gaily, S. E.—Imre, S.: Derivation of Parameters of Quantum Optimization in Resource Distribution Management. 2019 42$^{\mathrm{nd}}$ International Conference on Telecommunications and Signal Processing (TSP), 2019, pp. 58–61, doi: 10.1109/TSP.2019.8769092.

[9] Gambella, C.—Simonetto, A.: Multiblock ADMM Heuristics for Mixed-Binary Optimization on Classical and Quantum Computers. IEEE Transactions on Quantum Engineering, Vol. 1, 2020, doi: 10.1109/TQE.2020.3033139.

[10] Glos, A.—Krawiec, A.—Zimborás, Z.: Space-Efficient Binary Optimization for Variational Computing. 2020, arXiv: 2009.07309.

[11] Gomes, J.—McKiernan, K. A.—Eastman, P.—Pande, V. S.: Classical Quantum Optimization with Neural Network Quantum States. 2019, arXiv: 1910.10675.

[12] Hadfield, S.—Wang, Z.—O'Gorman, B.—Rieffel, E. G.—Venturelli, D.—Biswas, R.: From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz. Algorithms, Vol. 12, 2019, No. 2, Art. No. 34, doi: 10.3390/a12020034.

[13] Kurowski, K.—Węglarz, J.—Subocz, M.—Różycki, R.—Waligóra, G.: Hybrid Quantum Annealing Heuristic Method for Solving Job Shop Scheduling Problem. In: Krzhizhanovskaya, V. et al. (Eds.): Computational Science – ICCS 2020. Springer, Cham, Lecture Notes in Computer Science, Vol. 12142, 2020, pp. 502–515, doi: 10.1007/978-3-030-50433-5_39.

[14] Lavrijsen, W.—Tudor, A.—Müller, J.—Iancu, C.—de Jong, W.: Classical Optimizers for Noisy Intermediate-Scale Quantum Devices. 2020 IEEE International Conference on Quantum Computing and Engineering (QCE), 2020, pp. 267–277, doi: 10.1109/qce49297.2020.00041.

[15] Li, R. Y.—Di Felice, R.—Rohs, R.—Lidar, D. A.: Quantum Annealing Versus Classical Machine Learning Applied to a Simplified Computational Biology Problem. npj Quantum Information, Vol. 4, 2018, Art. No. 14, doi: 10.1038/s41534-018-0060-8.

[16] Nelder, J. A.—Mead, R.: A Simplex Method for Function Minimization. The Computer Journal, Vol. 7, 1965, No. 4, pp. 308–313, doi: 10.1093/comjnl/7.4.308.

[17] Orús, R.—Mugel, S.—Lizaso, E.: Quantum Computing for Finance: Overview and Prospects. Reviews in Physics, Vol. 4, 2019, Art. No. 100028, doi: 10.1016/j.revip.2019.100028.

[18] Outeiral, C.—Strahm, M.—Shi, J.—Morris, G. M.—Benjamin, S. C.—Deane, C. M.: The Prospects of Quantum Computing in Computational Molecular Biology. WIREs Computational Molecular Science, Vol. 11, 2021, No. 1, Art. No. e1481, doi: 10.1002/wcms.1481.

[19] Pellow-Jarman, A.—Sinayskiy, I.—Pillay, A.—Petruccione, F.: A Comparison of Various Classical Optimizers for a Variational Quantum Linear Solver. Quantum Information Processing, Vol. 20, 2021, No. 6, Art. No. 202, doi: 10.1007/s11128-021-03140-x.

[20] Peruzzo, A.—McClean, J.—Shadbolt, P.—Yung, M. H.—Zhou, X. Q.—Love, P. J.—Aspuru-Guzik, A.—O'Brien, J. L.: A Variational Eigenvalue Solver on a Photonic Quantum Processor. Nature Communications, Vol. 5, 2014, No. 1, Art. No. 4213, doi: 10.1038/ncomms5213.

[21] Plewa, J.—Sieńko, J.: Hybrid Algorithms for Workflow Scheduling Problem in Quantum Devices Based on Gate Model. Master's Thesis, AGH University of Science and Technology, Kraków, 2021, `http://dice.cyfronet.pl/publications/source/MSc_theses/JPlewa_JSienko_msc.pdf`.

[22] Powell, M. J. D.: Direct Search Algorithms for Optimization Calculations. In: Acta Numerica, Vol. 7, 1998, pp. 287–336, doi: 10.1017/S0962492900002841.

[23] Powell, M. J.: A View of Algorithms for Optimization Without Derivatives. Mathematics Today – Bulletin of the Institute of Mathematics and Its Applications, Vol. 43, 2007, No. 5, pp. 170–174.

[24] Powell, M. J. D.: A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. In: Gomez, S., Hennart, J. P. (Eds.): Advances in Optimization and Numerical Analysis. Springer, Dordrecht, Mathematics and Its Applications, Vol. 275, 1994, pp. 51–67, doi: 10.1007/978-94-015-8330-5_4.

[25] Powell, M. J. D.: An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives. The Computer Journal, Vol. 7, 1964, No. 2, pp. 155–162, doi: 10.1093/comjnl/7.2.155.

[26] Stachoń, M. A.: Solving Optimization Problems Using Qiskit Aqua. Master's Thesis, AGH University of Science and Technology, Kraków, 2020, `http://dice.cyfronet.pl/publications/source/MSc_theses/Malgorzata_Stachon_msc.pdf`.

[27] Tan, B.—Lemonde, M.-A.—Thanasilp, S.—Tangpanitanon, J.—Angelakis, D. G.: Qubit-Efficient Encoding Schemes for Binary Optimisation Problems. Quantum, Vol. 5, 2021, Art. No. 454, doi: 10.22331/q-2021-05-04-454.

[28] Tomasiewicz, D.: Analysis of D'Wave 2000Q Applicability for Job Scheduling Problems. Master's Paper, AGH University of Science and Technology, Kraków, 2020, `http://dice.cyfronet.pl/publications/source/MSc_theses/Dawid_Tomasiewicz_msc.pdf`.

[29] Tomasiewicz, D.—Pawlik, M.—Malawski, M.—Rycerz, K.: Foundations for Workflow Application Scheduling on D-Wave System. In: Krzhizhanovskaya, V. V. et al. (Eds.): Computational Science – ICCS 2020. Springer, Cham, Lecture Notes in Computer Science, Vol. 12142, 2020, pp. 516–530, doi: 10.1007/978-3-030-50433-5_40.

[30] Venturelli, D.—Marchand, D. J. J.—Rojo, G.: Quantum Annealing Implementation of Job-Shop Scheduling. 2016, arXiv: 1506.08479.

[31] Wang, Z.—Rubin, N. C.—Dominy, J. M.—Rieffel, E. G.: XY Mixers: Analytical and Numerical Results for the Quantum Alternating Operator Ansatz. Physical Review A, Vol. 101, 2020, No. 1, Art. No. 012320, doi: 10.1103/PhysRevA.101.012320.

[32] Zhu, C.—Byrd, R. H.—Lu, P.—Nocedal, J.: Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization. ACM Transactions on Mathematical Software, Vol. 23, 1997, No. 4, pp. 550–560, doi: 10.1145/279232.279236.

Julia Plewa received her M.Sc. degree in computer science in 2021 from the AGH University of Science and Technology in Kraków, Poland. Her research interests have included driver behavior modeling and quantum computing. She currently works as a quantum computing researcher at BNP Paribas.

Joanna Sieńko received her M.Sc. degree in computer science in 2021 from the AGH University of Science and Technology in Kraków, Poland. Her interests include object-oriented programming, microservices, and cloud computing. She currently works as a software engineer at an e-commerce company.

Katarzyna Rycerz works as Assistant Professor at the Institute of Computer Science at AGH in Kraków, Poland. She is the co-author of over 50 international publications in the area of distributed computing, environments for multiscale simulations, quantum computing simulation, and support for scientific applications. She was involved in the EU ICT projects: CrossGrid (the Architecture Team member), CoreGRID, and MAPPER (WP leader).