

# Quantum Hardware-Efficient Selection of Auxiliary Variables for QUBO Formulations

Damian Rovara\*

Lukas Burgholzer\*†

Robert Wille\*‡†

\*Chair for Design Automation, Technical University of Munich, Germany

†Munich Quantum Software Company GmbH, Garching near Munich, Germany

‡Software Competence Center Hagenberg GmbH (SCCH), Austria

damian.rovara@tum.de

lukas.burgholzer@tum.de

robert.wille@tum.de

<https://www.cda.cit.tum.de/research/quantum>

**Abstract**—The *Quantum Approximate Optimization Algorithm* (QAOA) requires considered optimization problems to be translated into a compatible format. A popular transformation step in this pipeline involves the quadratization of higher-order binary optimization problems, translating them into *Quadratic Unconstrained Binary Optimization* (QUBO) formulations through the introduction of auxiliary variables. Conventional algorithms for the selection of auxiliary variables often aim to minimize the total number of required variables without taking the constraints of the underlying quantum computer—in particular, the connectivity of its qubits—into consideration. This quickly results in interaction graphs that are incompatible with the target device, resulting in a substantial compilation overhead even with highly optimized compilers. To address this issue, this work presents a novel approach for the selection of auxiliary variables tailored for architectures with limited connectivity. By specifically constructing an interaction graph with a regular structure and a limited maximal degree of vertices, we find a way to construct QAOA circuits that can be mapped efficiently to a variety of architectures. We show that, compared to circuits constructed from a QUBO formulation using conventional auxiliary selection methods, the proposed approach reduces the circuit depth by almost 40%. An implementation of all proposed methods is publicly available at <https://github.com/munich-quantum-toolkit/problemsolver>.

## I. INTRODUCTION

While large-scale quantum algorithms (such as Shor’s [1] or Grover’s [2] algorithm) still cannot be feasibly executed on state-of-the-art quantum computers, many experts in the field seek more tangible advantages through utility-scale algorithms compatible with *Near-Term Intermediate Scale Quantum* (NISQ) devices. Such algorithms are typically used to approximate solutions to optimization problems and focus on low-depth circuits that are employed in conjunction with classical optimizers [3]–[7]. A popular example for these utility-scale algorithms is the *Quantum Approximate Optimization Algorithm* (QAOA) [5]. This algorithm takes advantage of a hybrid quantum-classical approach to find the ground state of a Hamiltonian operator representing the desired optimization problem, which, in turn, leads to its optimal solution.

To solve an optimization problem using QAOA, the problem must first be reformulated into a compatible format, such as the *Quadratic Unconstrained Binary Optimization* (QUBO) formalism [8]–[10]. This involves the construction of a binary cost function  $Q(\mathbf{x})$  with  $\mathbf{x} \in \{0, 1\}^N$  consisting only of *linear terms* of the form  $c_i x_i$  or *quadratic terms* of the form  $c_{ij} x_i x_j$  for some real coefficients  $c_i$  and  $c_{ij}$ . In cases where the original cost function requires additional constraints or higher-order terms, additional variables must typically be introduced to remain within the rules of the QUBO formalism [11]–[13]. Once such a QUBO cost function has been constructed, it can be translated into a QAOA circuit in a straightforward manner that can then be executed on a quantum computer.

However, one key concern of NISQ devices lies in the compilation problem: As quantum computers typically only support a subset of quantum gates [14]–[16] and do not allow

arbitrary connectivity among all qubits [17]–[20], a quantum circuit defined in a device-agnostic way must first be compiled to the target architecture before it can be executed. This compilation procedure introduces overhead to the circuit that often makes it perform substantially worse than the ideal device-agnostic circuit would suggest. Therefore, the task of efficiently compiling quantum circuits is an important area of research in the field of quantum computing design automation [21], [22].

For the specific use case of compiling QAOA circuits for specific architectures, a myriad of previous work has already shown the potential improvements that can be achieved by considering the compilation problem for such a specialized scenario [23]–[26]. Similarly, previous work on the task of constructing QUBO formulations was able to automate the generation of QUBO cost functions from a variety of input specifications in an efficient manner [27]–[31]. This work, however, attempts to combine these two approaches by taking device considerations already into account at the time of constructing the QUBO formulation. This is achieved by selecting auxiliary variables in such a way that the structure of interactions between the variables remains regular in a way that can be mapped efficiently to the target machine. While the number of qubits required to implement QAOA circuits from these hardware-efficient QUBO cost functions rises, the depth of these circuits is lower than conventionally created QAOA circuits, as overheads introduced by compilation are reduced. Our evaluations show a rising improvement as the considered problem size increases, with an average depth reduction of 39.2% at problems with 16 variables.

The remainder of this work is structured as follows: Section II provides an overview on the construction of QUBO formulations and how QAOA circuits are typically generated from them. Section III then highlights the issues of this hardware-agnostic approach and instead motivates a solution for the hardware-efficient construction of QUBO formulations. The algorithms proposed to construct such QUBO formulations and compile them to hardware-compatible QAOA circuits in the ideal case are then proposed in Section IV, whereas Section V illustrates how special cases can be handled efficiently as well. Finally, the results obtained during the evaluation of the proposed methods are summarized in Section VI before Section VII concludes this work.

## II. BACKGROUND

This section briefly reviews the underlying concepts of QUBO problems and QAOA.

### A. QUBO Formulations

To optimize a binary cost function using QAOA, the cost function is commonly first translated to the QUBO formalism [8]–[10]. In particular, this requires the reduction of all higher-order terms to quadratic terms by introducing

auxiliary variables, a procedure also known as quadratization [25], [32]–[36]. Each auxiliary variable encodes a product of two variables  $y_i = x_{i_1} x_{i_2}$ . By substituting all occurrences of this product by the corresponding auxiliary variable, the order of each updated term is reduced by 1. The introduction of this auxiliary variable also requires an additional penalty term  $P_i(x_a, x_b) = c_{P_i}(x_a x_b - 2x_a y_i - 2x_b y_i + 3y_i)$  to be added to the cost function. This penalty term ensures that the total cost cannot be reduced by violating the equality constraint  $y_i = x_a x_b$ . To incentivize its minimization, each penalty term is multiplied by a constant cost factor  $c_{P_i}$  that has to be chosen sufficiently large, depending on the full QUBO cost function. Repeatedly applying these substitutions allows any higher-order unconstrained binary optimization problem to be reduced to a QUBO cost function.

**Example 1.** Consider the following binary optimization function to be approximated using QAOA:

$$C(\mathbf{x}) = x_1 x_2 x_3 x_4$$

This cost function, together with its general extension to  $N$  variables  $C_N(\mathbf{x}) = \prod_i^N x_i$ , will serve as running examples for this work. To translate this binary optimization problem into the QUBO formalism, we substitute (sub-)products of two variables with newly introduced auxiliary variables to reduce the order of the expression. An efficient way to achieve this is to select commonly occurring products first with the goal of minimizing the number of additionally introduced variables. For the considered expression, we may choose  $y_1 = x_1 x_2$  and  $y_2 = x_3 x_4$ , resulting in the new cost function:

$$Q(\mathbf{x}, \mathbf{y}) = y_1 y_2 + P_1(x_1, x_2) + P_2(x_3, x_4)$$

with two additional variables  $y_i$  and two penalty terms  $P_i$  defined as above, that account for the substitution of the auxiliary variables. The corresponding penalty factors were both chosen as  $c_{P_i} = 1$ .

The process of transforming general optimization problems into the QUBO format is a popular area of research. Existing solutions provide approaches to find space-efficient QUBO formulations for higher-order optimization problems as well as for problems that include additional constraints [27]–[30].

### B. Optimizing QUBO Problems with QAOA

Once a QUBO cost function is generated, algorithms such as QAOA [5], [37] can be used to find its optimal variable assignment. As a *variational quantum algorithm* (VQA) [3], QAOA consists of two components: a *parametrized quantum circuit* for the preparation of a state as well as a *classical optimizer* that updates the circuit parameters to find the optimal variable assignment [38], [39].

QAOA requires the QUBO cost function to be translated to a *cost Hamiltonian*  $H_C$ . The goal of the circuit is then to find the ground state of this Hamiltonian which corresponds to the assignment resulting in the minimal cost. In practice, this is achieved by applying the operator  $e^{-i\gamma H_C}$  to the uniform superposition  $H^{\otimes n} |0\rangle$  for some parameter  $\gamma$ . This is then followed by the application of  $e^{-i\beta H_M}$  with some other parameter  $\beta$ , representing an evolution towards the *mixer Hamiltonian*  $H_M$  [40], [41], which typically consists of a set of  $X$  rotations on each qubit. These cost and mixer layers can be applied to the circuit repeatedly, utilizing new parameters  $\gamma_i, \beta_i$  for each repetition layer. While, often, a single application of the layers is already enough to yield results, it has been shown that each repetition further increases the

accuracy of the algorithm [5]. The general process of starting with a QUBO cost function and constructing the QAOA cost layer from it is illustrated in the following example.

**Example 2.** Returning to the QUBO cost function introduced in Example 1, the corresponding QAOA circuit requires 6 qubits: 4 qubits represent the initial problem variables  $x_i$ , while the remaining two qubits represent the auxiliary variables  $y_j$ . All qubits are initially set to a uniform superposition by applying an  $H$  gate to each qubit. We then map the cost function to the cost layer in the QAOA circuit construction by adding an  $rzz(2c_{ij}\gamma)$  gate for each quadratic term where  $c_{ij}$  is the term's coefficient in the cost function. Furthermore, each individual qubit also requires a  $rz(2b_i\gamma)$  gate, where  $b_i = -2c_i - \sum_j c_{ij}$  is introduced by the translation from a binary cost function to a Hamiltonian. Here,  $c_i$  represents the coefficient of each variable's linear term in the QUBO function. The resulting circuit is illustrated in Fig. 1.

The resulting QAOA circuit can then be used to find the expectation value of  $H_C$  by repeatedly running the circuit and measuring the qubits. A classical optimizer can use the measurement results to tune the parameters  $\gamma_i, \beta_i$  until an assignment is found that yields an adequately low cost value. This assignment can then be translated back directly to the assignment that minimizes the original cost function  $C(\mathbf{x})$ .

## III. MOTIVATION AND GENERAL IDEA

This section describes the typical workflow of mapping QAOA circuits to quantum computers. In the process, it highlights common challenges and limitations of the approach, and motivates how an architecture-aware selection of auxiliary variables during the construction of QUBO formulations can mitigate and resolve them.

### A. Considered Problem

Due to the limitations of state-of-the-art quantum computers, the depth of quantum circuits is a crucial criterion for their usefulness. Not only do disproportionately deep circuits require more time for their execution, consuming valuable limited resources, but each additional gate has a negative impact on the circuit's probability of success due to gate errors and noise. Unfortunately, the connectivity between hardware qubits is limited on many currently explored quantum computer architectures, such as superconducting systems. Thus, additional gates (such as SWAPs) need to be inserted into the original circuit so that it conforms to the device's topology. This typically leads to substantial increases in circuit depth.

**Example 3.** Fig. 2 shows the coupling map of the 133-qubit superconducting quantum computer `ibm_torino` provided by the IBM Quantum Cloud platform [42]. It employs a heavy-hex topology, which is characterized by a grid of 12-qubit blocks that are arranged in a brick wall pattern. In this topology, the maximum degree of nodes in the coupling map is 3, with a majority of qubits only being connected to 2 neighbors. However, considering once again the binary cost function introduced in Example 1 and its corresponding QAOA circuit constructed in Example 2, it is easy to see that the circuit is not directly compatible with this coupling map: As the circuit in Fig. 1 requires two directly connected qubits with a degree of 3, it must be modified to fit the device topology. Specifically, this circuit requires at least 2 SWAP gates to be compatible

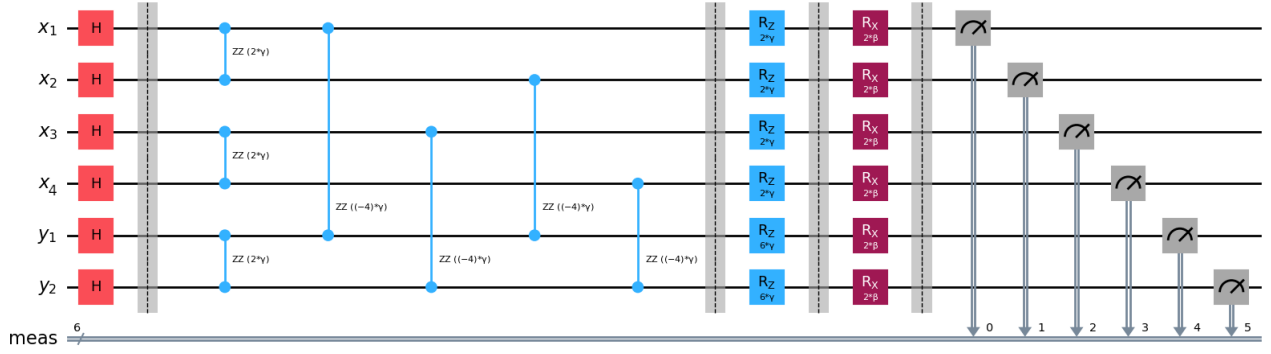


Figure 1. The QAOA circuit for the optimization problem introduced in Example 1 using one repetition layer.

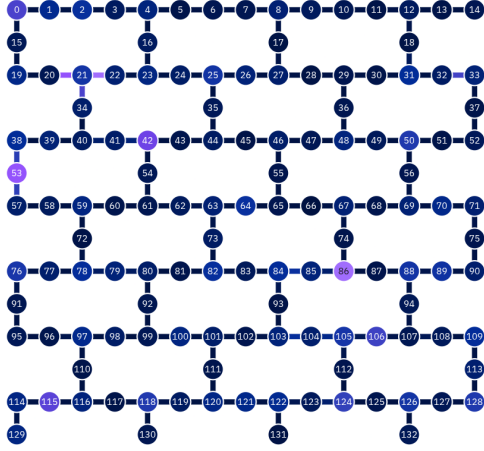


Figure 2. The coupling map of the 133-qubit `ibm_torino` superconducting quantum computer based on IBM's *Heron* processor type.

with `ibm_torino`, that each have to be further decomposed into the native gates supported by the device<sup>1</sup>.

While finding the optimal circuit mapping to minimize the total depth is a computationally hard problem [43], various optimization techniques have already been proposed for this task [18], [19], [44]–[48]. Nonetheless, the impact of the increased depth caused by this procedure may greatly reduce the efficiency of the entire QAOA algorithm, as even the addition of a low number of SWAP gates may already increase the final circuit size considerably. This effect only gets worse with larger circuits and more complex cost functions.

While several approaches have been proposed to tackle this compilation overhead for QAOA circuits [23], [24], [26], [49], none of the existing approaches can really overcome the underlying problem that any created QUBO cost function might already be a bad fit for the architecture. However, the process of constructing QUBO formulations from general binary optimization problems offers some degree of freedom in choosing auxiliary variables. Crucially, this decision should already take into account the hardware's constraints in order to minimize the subsequent compilation overhead.

### B. General Idea

Due to the commutative property of  $rzz$  and  $rz$  gates, the order of individual gates within a cost layer in the resulting circuit does not matter. Therefore, we can focus all optimization

<sup>1</sup>IBM's latest generation of quantum computers natively supports `id`, `x`, `sx`, `rz`, `rx`, `cz`, and `rzz` gates. Based on that gateset, a SWAP can be decomposed to a set of 3 `cz` gates and 6 `sx` gates for a total depth of 6.

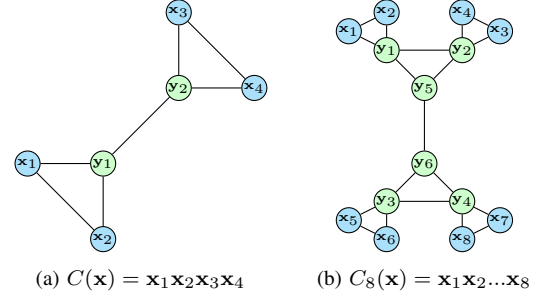


Figure 3. The interaction graph constructed from the QUBO formulation proposed in Example 1, as well as its extension to 8 variables, following the strategy of repeatedly replacing all sub-products of two variables by a new auxiliary variable until only a quadratic cost function is left.

efforts only on the interactions between variables and, in extension, the qubits that represent them. To this end, we illustrate QUBO formulations as undirected graphs, where each vertex represents a variable in the expression and each edge between two vertices indicates that the product of the corresponding variables is a term in the full cost function. Blue vertices represent the original problem variables and are labeled with  $x_i$ , while green vertices represent auxiliary variables and are labeled with  $y_i$ .

**Example 4.** Fig. 3a illustrates the interaction graph constructed for the QUBO cost function introduced in Example 1. It is clearly visible how each edge in the graph directly translates to an  $rzz$  gate in the circuit in Fig. 1. Comparing this interaction graph with the coupling map shown in Fig. 2 demonstrates that it clearly cannot be mapped directly to the `ibm_torino` device. However, the interaction graph complexity rises even further when including additional variables in the initial cost function. Fig. 3b shows the interaction graph obtained when using the same approach of selecting auxiliary variables on a product of 8 binary variables. Generally, as the number of variables grows, we similarly expect the interaction graph to become more complex. Such graphs may contain vertices with drastically increasing degrees, leading to a similar rise in the required additional gates introduced by the compilation procedure.

By further investigating the nature of the interaction graph, we see that each auxiliary variable  $y_k$  introduced to replace the product  $x_{k_1}x_{k_2}$  leads to a triangle connecting the three variables in the interaction graph. This shows how the selection of auxiliary variables can be leveraged to influence the required connections between qubits. In particular, by selecting each new auxiliary variable  $y_{i+1}$  in such a way that the previously

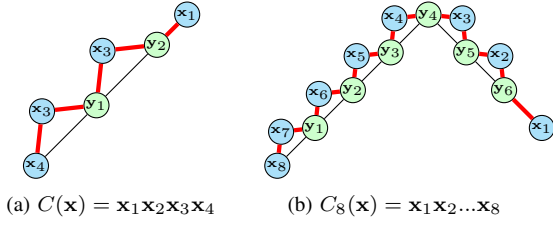


Figure 4. The interaction graph constructed by selecting auxiliary variables to form chains of triangles. The red lines indicate the ordering in which the variables should be mapped to a path of qubits on the target device.

added auxiliary variable  $y_i$  is a factor of the replaced term, the resulting interaction graph will always be a chain of such triangles. This way, even if the number of variables increases, the same chain-of-triangles structure will be retained consistently, allowing qubit mapping strategies to be proposed for that regular structure, rather than the less predictable graphs shown in Fig. 3.

**Example 5.** Returning to the initial cost function defined in Example 1, we propose an alternative selection of auxiliary variables. We begin by substituting the product of any pair of variables, such as  $y_1 = x_3 x_4$ . Then, we introduce the second auxiliary variable  $y_2$  to substitute a product that includes  $y_1$ , such as  $y_2 = x_2 y_1$ . This results in a new QUBO formulation that can be translated to the interaction graph illustrated in Fig. 4a. Clearly, this graph includes two triangles ( $\overline{y_1 x_3 x_4}$ , and  $\overline{y_2 x_2 y_1}$ ), corresponding to the two auxiliary variables. Furthermore, by once again extending the approach to 8 variables in Fig. 4b, we see how the structure consistently expands to a chain of 6 triangles for the 6 required auxiliary variables. Any additional variables added to the product cost function  $\prod_i^N x_i$  will result in an additional triangle in the interaction graph in the same consistent and regular manner.

While the resulting circuits still typically cannot be directly executed on devices with a limited connectivity, the regular structure of the circuits allows to derive application-specific placement and routing strategies. By mapping the variables onto a line of qubits that support nearest neighbor connections according to the path highlighted in red in Fig. 4, any interaction graph following this structure can be implemented by a circuit with constant depth. In the following, we further investigate how such a circuit can be constructed in detail and what changes have to be applied to also support cost functions that cannot be represented by a single chain of triangles.

#### IV. HARDWARE-EFFICIENT QUBO GENERATION

Continuing from the general idea proposed above, this section illustrates in more detail how auxiliary variables can be selected in a hardware-efficient manner even in more complex cases. It then shows how the resulting structure of auxiliary variables can be leveraged to efficiently compile QAOA circuits for specific architectures.

##### A. Hardware-Efficient Auxiliary Variable Selection

To construct a QUBO formulation from higher-order binary cost functions, all terms of order 3 or higher must be reduced by adding auxiliary variables to the cost function. Each auxiliary variable  $y_i$  may substitute a product of two binary variables taken either from the original problem variables  $x$  or the previously added auxiliary variables  $y$ .

To construct an interaction graph that consists of a maximal chain of triangles, each newly selected auxiliary variable  $y_{i+1}$  must replace the previously introduced auxiliary variable  $y_i$  as

well as a problem variable  $x_j$  that has not yet been substituted by an auxiliary variable. To this end, we propose a greedy algorithm that chooses the most commonly occurring  $x_j$  to increase the likelihood of being able to further extend the chain. Using the resulting substitution, it transforms the original cost function  $C(x)$  into the QUBO cost function  $Q(x, y)$ .

We begin by selecting the sub-product that appears most frequently among all terms in  $C(x)$  to be substituted by the first auxiliary variable. All occurrences of the sub-product in  $C(x)$  are then replaced by the newly introduced auxiliary variable and the penalty term required by it is added to the cost function. We then iterate through all terms in the remaining cost function with order greater than 2, searching for the problem variable  $x_i$  that most frequently appears in a product together with the newly introduced auxiliary variable. This product is substituted by a new auxiliary variable and the corresponding penalty term is added once again. This process is then repeated, iteratively reducing the order of the cost function by substituting a previous auxiliary variable  $y_i$  with a new variable  $y_{i+1}$ . If the procedure can be repeated until no more higher-order terms remain in the cost function, the resulting interaction graph contains a single chain of triangles, as shown in Fig. 4.

**Example 6.** Consider the following initial cost function:

$$C(x) = x_1 x_2 x_3 x_4 x_5 + x_1 x_2 x_3 x_4 + x_2 x_3 x_4$$

The most frequently appearing sub-products are  $x_3 x_4$ ,  $x_2 x_4$ , and  $x_2 x_3$ . We first substitute  $x_3 x_4$  by  $y_1$ . This changes the cost function to

$$C(x, y) = x_1 x_2 x_5 y_1 + x_1 x_2 y_1 + x_2 y_1 + P_1(x_3, x_4)$$

Now, the most frequent problem variable appearing in a product with  $y_1$  is  $x_2$ . Therefore,  $y_2 = x_2 y_1$ , changing the cost function to

$$C(x, y) = x_1 x_5 y_2 + x_1 y_2 + y_2 + P_1(x_3, x_4) + P_2(y_1, x_2)$$

Finally, only the first term of the cost function has an order higher than two, thus, all contained variables appear with the same frequency. For this example, we select  $y_3 = y_2 x_5$ , resulting in the final QUBO cost function

$$Q(x, y) = x_1 y_3 + x_1 y_2 + y_2 + P_1(x_3, x_4) + P_2(y_1, x_2) + P_3(y_2, x_5).$$

This QUBO formulation results in an interaction graph with a chain of triangles structure consisting of 3 triangles, each originating from one of the penalty functions  $P_i$ .

The proposed algorithm constructs a hardware-efficient QUBO formulation in a variety of cases. While it does not succeed in finding a single chain of triangles in all cases, these special cases can also be handled in an efficient manner, as discussed later in Section V.

##### B. Compiling QAOA Circuits from Interaction Graphs

Once a binary cost function is translated to an interaction graph based on the rules formulated in Section IV-A, it can be translated to a QAOA circuit using a set of simple rules.

We start by following a path of vertices on the interaction graph, starting with an edge between the first two problem variables that were substituted by  $y_1$ . We then continue the path, iteratively moving to each subsequent auxiliary variable  $y_{i+1}$  connected by edges through the problem variable that has an edge to both  $y_i$  and  $y_{i+1}$ . Examples for such paths are highlighted in Fig. 4.

We then find a path of qubits on the coupling map with the same size. While finding such a path is a hard problem on general graphs [50], the regular structure of many superconducting device topologies often allows such a path to be found efficiently. For the heavy hex topology, for instance, as shown in Fig. 2, a path can be found by starting at the lowest-index qubit that is only connected to one neighbor (14 in the example) and continuing to traverse the coupling map, always selecting the qubit with the closest index. For `ibm_torino` with a total number of 133 qubits, this allows for a path with a maximum length of 112.

Each variable of the path on the interaction graph is then mapped to the qubit at its corresponding index in the path through the coupling map. A  $H$  gate is applied to each qubit to prepare a superposition for the QAOA circuit. To apply the cost Hamiltonian, we then use the following algorithm:

- 1) *Even direct interactions*: Apply an `rz` gate to each  $q_{2i}, q_{2i+1}$  for each  $i \in [0, \lfloor q/2 \rfloor]$ . As the selection of qubits supports nearest neighbor interactions and none of the required gates intersect each other, all gates can be performed in parallel in a single layer.
- 2) *Odd direct interactions*: Apply an `rz` gate to each  $q_{2i-1}, q_{2i}$  for each  $i \in [1, \lfloor q/2 \rfloor]$ . Once again, all of these gates can be applied in a single layer.
- 3) *Even indirect interactions*: Apply an `rz` gate to each  $q_{4i}, q_{4i+2}$  for each  $i \in [0, \lfloor q/4 \rfloor]$ . As there is a gap of size 1 between each pair of targets, each `rz` gate requires a `SWAP` gate to move the qubits together and a second `SWAP` gate to move them back. Therefore, this step only requires a constant depth.
- 4) *Odd indirect interactions*: Apply an `rz` gate to each  $q_{4i-2}, q_{4i}$  for each  $i \in [1, \lfloor q/4 \rfloor]$ . Due to the gaps between the targeted qubits, the circuit once again requires `SWAP` gates to move them together. In this case, however, no second `SWAP` gate is required to move the qubits back, as the cost function is now complete and requires no further 2-qubit gates<sup>2</sup>.
- 5) *Linear terms*: Finally, for all linear terms in the cost function, apply an `rz` gate to the corresponding qubit, keeping in mind possible changes to the qubit layout introduced during the previous step.

For each applied rotation gate, the corresponding angle has to be computed as  $2c\gamma$ , where  $c$  is the coefficient of the corresponding term in the cost function and  $\gamma$  is the QAOA parameter of the current repetition layer. In total, this process results in a QAOA cost Hamiltonian with a constant depth. On `ibm_torino`, the decomposition of `SWAP` gates mentioned in Section III results in a total depth of 23 for one application of the cost Hamiltonian. As all individual steps of the above algorithm can be performed in parallel, the circuit depth will not grow, even if the interaction graph increases in size.

## V. HANDLING SPECIAL CASES

The approach proposed above provides a general strategy for the efficient selection of auxiliary variables. However, two additional scenarios must be handled depending on the initial cost function. While a single, regular chain of triangles can be mapped efficiently to a variety of different topologies, these anomalies require additional adaptations that may increase the circuit depth. In the following, we will discuss how these special cases can be handled and how hardware-efficient QAOA circuits can be constructed from them.

<sup>2</sup>If further repetition layers are required, odd-numbered iterations should instead apply step 4 first in reverse, returning the qubits to the original layout, and then continue with the remaining steps from 1 to 5.

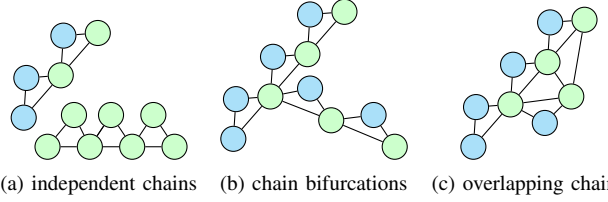


Figure 5. Potential types of anomalies in the interaction graph if the variable selection algorithm is executed multiple times.

### A. Extraneous Interactions

Pre-existing quadratic terms in the cost function are not considered during the choice of auxiliary variables. While some of them may still be substituted by auxiliary variables that consider the same products, most of these pre-existing quadratic terms are not expected to be influenced by the auxiliary variable selection. Therefore, the corresponding interactions will remain part of the resulting interaction graph.

Each of these *extraneous interactions* requires a corresponding `rz` gate to be added to the end of the cost layer of the circuit. As these qubits will typically not be connected in the coupling map, additional `SWAP` gates must be added accordingly. For this process, we employ conventional `SWAP` insertion methods [18], [19]. These additional `SWAP` gates will further increase the circuit depth, depending on the amount of extraneous interactions.

### B. Additional Chains

Furthermore, if at some point during the execution, no new  $y_{i+1}$  can be found that substitutes a product with the factor  $y_i$ , the algorithm stops early. In this case, the resulting cost function still contains higher order terms. To resolve this, the algorithm can be executed again, starting with the cost function  $Q(\mathbf{x}')$ , where  $\mathbf{x}' = \mathbf{x} \cup y$ . This repeated execution leads to *additional chains* in the interaction graph which may have different structures, depending on what variables are selected in the subsequent execution of the algorithm. Examples for each of these altered kinds of structures are illustrated in Fig. 5.

- *Independent Chains* (Fig. 5a): If no variable selected for substitution by the second execution was an auxiliary variable or a problem variable selected during the first execution of the variable selection algorithm, the second execution results in two individual chains of triangles in the final interaction graph.
- *Chain Bifurcations* (Fig. 5b): If a variable selected for substitution by the second execution was an auxiliary variable introduced during the first execution, the resulting interaction graph will split into two chains at the corresponding vertex.
- *Overlapping Chains* (Fig. 5c): If several variables selected for substitution by the second execution were also selected as problem variables or introduced as auxiliary variables in the first execution, the resulting chains of triangles overlap at the corresponding vertices, resulting in a less structured interaction graph.

Depending on the type of additional chain, the mapping process must be adapted. In the case of *independent chains*, each chain can be mapped to the target architecture individually. As no qubits are reused among both chains, all corresponding gates between the two chains can be applied in parallel, leading to no increase in circuit depth. For *chain bifurcations* or *overlapping chains*, the initial structure must first be modified to transform them into *independent chains*. This can be achieved by noting that in a binary cost function, the equality  $x = x'$

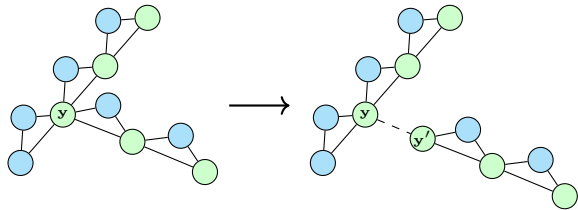


Figure 6. The process of splitting a bifurcation into two individual chains by introducing a new variable  $y'$ .

can be enforced by the penalty term  $(x - x')^2$ . Therefore, for each variable  $x_i$  shared between two chains, we introduce a new variable  $x'_i$ . We then assign  $x_i$  to the first chain and  $x'_i$  to the second chain, adding a single new edge between the two variables due to the penalty term as an *extraneous interaction*. This process is further illustrated by Fig. 6. This once again allows the chains to be handled in parallel in the resulting QAOA circuit. The total depth is only increased by the cost of inserting SWAP gates for the single newly introduced interaction, while the circuit width is increased depending on the number of shared variables.

## VI. EVALUATION AND DISCUSSION

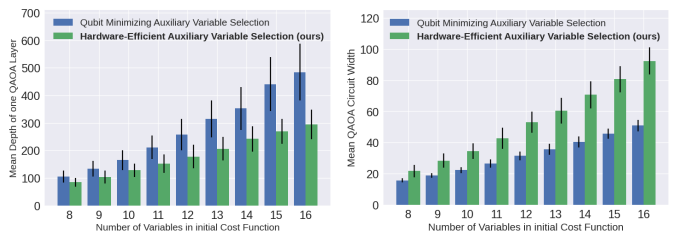
To evaluate the proposed methodology, we implemented the core methods with all code publicly available at <https://github.com/munich-quantum-toolkit/problemsolver> as part of the Munich Quantum Toolkit (MQT, [51]). We then conducted evaluations, targeting IBM’s `ibm_torino` superconducting device with 133 qubits for all procedures.

This evaluation focuses on randomly generated cost functions of the form  $\sum_i^N c_i \prod \bar{x}_i$ , where each  $\bar{x}_i$  is the product of a randomly chosen subset of all  $N$  encoding variables  $\mathbf{x}$  and each  $c_i$  is a random coefficient with  $-10 \leq c_i \leq 10$ . This results in an adequately random selection of cost functions that are not biased towards the proposed solution. 100 inputs were generated for each input size  $8 \leq N \leq 16$ .

Circuit depth gives valuable insight into the performance of a circuit on NISQ devices. It provides an estimate on both the expected accuracy of the results as well as the expected runtime of the circuit.

Each generated cost function was then compiled to the target architecture using the hardware-efficient auxiliary variable selection proposed in this work. Simultaneously, using a conventional heuristic to select auxiliary variables in order to minimize the total number of variables, an alternative QUBO formulation was constructed for each input and its corresponding QAOA circuit was mapped to the target device using *Qiskit*’s [52] `transpile` method with optimization level 3. Fig. 7a shows a comprehensive comparison of the resulting circuit depths for both approaches, where a higher value corresponds to a comparatively worse circuit. It indicates a clear reduction in circuit depth, especially as the size of inputs rises, resulting in an average improvement of 39.2% at the maximal sizes possible for the 133 qubit device. Fig. 7b further compares the circuit widths of the approaches. Here, higher values indicate that the circuit requires a larger number of qubit and is, therefore, more difficult to execute. As expected, the proposed method for auxiliary variable selection sacrifices width for depth. For the largest investigated inputs, the conventional auxiliary variable selection method requires an average of 45.0% fewer qubits than the proposed approach.

Furthermore, Fig. 8 shows a detailed comparison between both approaches for individual input functions. The proposed method consistently performs better in almost all instances and the green trend line through the origin clearly suggests a general improvement as the circuit size increases.



(a) depth

(b) width

Figure 7. The average depth and width of one QAOA repetition layer on `ibm_torino`, averaged over 100 samples per size, using the approach proposed in this work compared to a hardware-agnostic strategy that aims to minimize the total number of qubits.

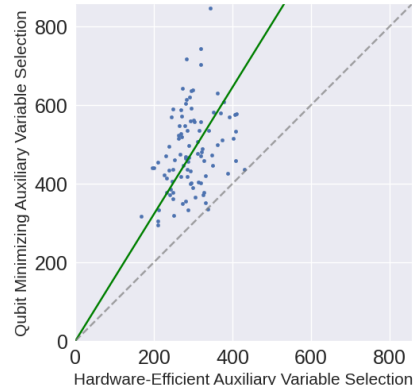


Figure 8. The depth of a single QAOA repetition layer on `ibm_torino` using the approach proposed in this work compared to a hardware-agnostic strategy that aims to minimize the number of qubits for individual test inputs.

## VII. CONCLUSION

In this work, we have proposed a novel approach for the selection of auxiliary variables to construct hardware-efficient QUBO formulations. A greedy algorithm for the selection of substitutions leads to a regular interaction graph structure even as the total number of variables grows. A secondary algorithm was proposed to map this interaction graph to a QAOA circuit that achieves constant depth in the ideal case. In cases where the ideal outcome cannot be obtained, we further provided strategies to minimize the resulting limitations. Evaluations of the proposed method have shown an increase in the advantage gained by this method in terms of circuit depth as the number of variables grows at the expense of circuit width. Notably, the proposed auxiliary variable selection method is complementary with other compilation techniques. Even without using the proposed method for QAOA circuit generation, improvements can already be achieved by using conventional compilers with the generated QUBO cost function. Future work may investigate combining the proposed approach with existing QAOA-optimized compilers to achieve even greater depth reductions. Furthermore, new strategies, such as qubit reuse, may also be leveraged to reduce the number of required qubits, mitigating the circuit width cost of the proposed methodology. All proposed methods are made available as part of an open-source implementation at <https://github.com/munich-quantum-toolkit/problemsolver>.

## ACKNOWLEDGMENTS

This work received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 101001318), was part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus, and has been supported by the BMK and BMDW. 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.

## REFERENCES

- [1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," *Foundations of Computer Science*, 1994.
- [2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Theory of computing*, 1996.
- [3] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, et al., "Variational quantum algorithms," *Nature Reviews Physics*, 2021.
- [4] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvale solver on a photonic quantum processor," *Nature Communications*, 2014.
- [5] E. Farhi, J. Goldstone, and S. Gutmann, *A quantum approximate optimization algorithm*, 2014. arXiv: 1411.4028 [quant-ph].
- [6] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, 2016.
- [7] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *nature*, 2017.
- [8] G. A. Kochenberger, F. Glover, B. Alidaee, and C. Rego, "A unified modeling and solution framework for combinatorial optimization problems," *OR Spectrum*, 2004.
- [9] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, "The unconstrained binary quadratic programming problem: A survey," *Journal of Combinatorial Optimization*, 2014.
- [10] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, 2014.
- [11] F. Glover, G. Kochenberger, and Y. Du, *A Tutorial on Formulating and Using QUBO Models*, 2019. arXiv: 1811.11538 [cs].
- [12] M. Ayodele, "Penalty Weights in QUBO Formulations: Permutation Problems," in *Evolutionary Computation in Combinatorial Optimization*, Cham: Springer International Publishing, 2022.
- [13] A. Verma and M. Lewis, "Penalty and partitioning techniques to improve performance of QUBO solvers," *Discrete Optimization*, 2022.
- [14] D. M. Miller, R. Wille, and Z. Sasanian, "Elementary quantum gate realizations for multiple-control Toffoli gates," in *Int'l Symp. on Multi-Valued Logic*, 2011.
- [15] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical Review A*, 1995.
- [16] P. V. Sriluckshmy, V. Pina-Canelles, M. Ponce, M. G. Algaba, F. Š. IV, and M. Leib, "Optimal, hardware native decomposition of parameterized multi-qubit Pauli gates," *Quantum Science and Technology*, 2023.
- [17] A. Holmes, S. Johri, G. G. Guerreschi, J. S. Clarke, and A. Y. Matsuura, "Impact of qubit connectivity on quantum algorithm performance," *Quantum Science and Technology*, 2020.
- [18] A. Zulehner, A. Paler, and R. Wille, "An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.
- [19] R. Wille and L. Burgholzer, "MQT QMAP: Efficient quantum circuit mapping," in *Int'l Symp. on Physical Design*, 2023.
- [20] P. Hopf, L. Burgholzer, and R. Wille, "Quantum Circuit Compilation for Superconducting Bus-Resonator Architectures," in *Design, Automation and Test in Europe*, 2026.
- [21] L. Burgholzer and R. Wille, *Design Automation Tools and Software for Quantum Computing: Inside the Munich Quantum Toolkit*. Springer, 2026.
- [22] R. Wille, L. Burgholzer, S. Hillmich, T. Grurl, A. Ploier, and T. Peham, "The Basis of Design Tools for Quantum Computing: Arrays, Decision Diagrams, Tensor Networks, and ZX-Calculus," in *Design Automation Conf.*, 2022.
- [23] M. Alam, A. Ash-Saki, and S. Ghosh, "Circuit Compilation Methodologies for Quantum Approximate Optimization Algorithm," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.
- [24] Y. Zhu, Y. Zhou, J. Cheng, Y. Jin, B. Li, S. Niu, and Z. Liang, "Coqa: Blazing Fast Compiler Optimizations for QAOA," in *2024 ACM/IEEE International Conference on Computer-Aided Design*, 2024.
- [25] L. Schmidbauer and W. Mauerer, *SAT Strikes Back: Parameter and Path Relations in Quantum Toolchains*, 2025. arXiv: 2505.22060 [quant-ph].
- [26] Y. Ji, X. Chen, I. Polian, and Y. Ban, "Algorithm-oriented qubit mapping for variational quantum algorithms," *Physical Review Applied*, 2025.
- [27] D. Volpe, N. Quetschlich, M. Graziano, G. Turvani, and R. Wille, "Towards an Automatic Framework for Solving Optimization Problems with Quantum Computers," in *IEEE International Conference on Quantum Software (QSW)*, 2024.
- [28] D. Rovara, N. Quetschlich, and R. Wille, *A Framework to Formulate Pathfinding Problems for Quantum Computing*, 2024. arXiv: 2404.10820 [quant-ph].
- [29] J. T. Iosue, "Qubovert." (2020), [Online]. Available: <https://qubovert.readthedocs.io> (visited on 01/07/2025).
- [30] IBM, "IBM Decision Optimization CPLEX Modeling for Python." (2023), [Online]. Available: <https://ibmdecisionoptimization.github.io/docplex-doc/> (visited on 01/07/2025).
- [31] B. Nash, V. Gheorghiu, and M. Mosca, "Quantum circuit optimizations for NISQ architectures," *Quantum Science and Technology*, 2020.
- [32] I. Rosenberg, "Reduction of Bivalent Maximization to the Quadratic Case.," *Cahiers du Centre d'Etudes de Recherche Operationnelle* 17, 1975.
- [33] A. Verma and M. Lewis, "Optimal quadratic reformulations of fourth degree Pseudo-Boolean functions," *Optimization Letters*, 2020.
- [34] M. Anthony, E. Boros, Y. Crama, and A. Gruber, "Quadratic reformulations of nonlinear binary optimization problems," *Mathematical Programming*, 2017.
- [35] L. Schmidbauer, E. Lobe, I. Schaefer, and M. Wolfgang, *It's Quick to be Square: Fast Quadratisation for Quantum Toolchains*, 2024. arXiv: 2411.19934 [quant-ph].
- [36] L. Schmidbauer, K. Wintersperger, E. Lobe, and W. Mauerer, *Polynomial Reduction Methods and their Impact on QAOA Circuits*, 2024. arXiv: 2406.08889 [quant-ph].
- [37] K. Blekos, D. Brand, A. Ceschini, C.-H. Chou, R.-H. Li, K. Pandya, and A. Summer, "A review on Quantum Approximate Optimization Algorithm and its variants," *Physics Reports*, 2024.
- [38] X. Bonet-Monroig, H. Wang, D. Vermetten, B. Senjean, C. Moussa, T. Bäck, V. Dunjko, and T. E. O'Brien, "Performance comparison of optimization methods on variational quantum algorithms," *Physical Review A*, 2023.
- [39] M. J. D. Powell, "Direct search algorithms for optimization calculations," *Acta Numerica*, 1998.
- [40] F. G. Fuchs, K. O. Lye, H. Møll Nilsen, A. J. Stasik, and G. Sartor, "Constraint Preserving Mixers for the Quantum Approximate Optimization Algorithm," *Algorithms*, 2022.
- [41] L. C. G. Govia, C. Poole, M. Saffman, and H. K. Krovi, "Freedom of the mixer rotation axis improves performance in the quantum approximate optimization algorithm," *Physical Review A*, 2021.
- [42] IBM Quantum, <https://quantum-computing.ibm.com/>. (2025), (visited on 01/07/2025).
- [43] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Int'l Symp Comb. Search*, 2018.
- [44] M. Y. Siraichi, V. F. dos Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation," in *International Symposium on Code Generation and Optimization*, 2018.
- [45] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, *Quantum circuit optimization with deep reinforcement learning*, 2021. arXiv: 2103.07585 [quant-ph].
- [46] F. Wagner, A. Bärmann, F. Liers, and M. Weissenböck, "Improving Quantum Computation by Optimized Qubit Routing," *Journal of Optimization Theory and Applications*, 2023.
- [47] G. Li, Y. Ding, and Y. Xie, "Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.
- [48] Y. Guo and S. Yang, "Efficient quantum circuit compilation for near-term quantum advantage," *EPJ Quantum Technology*, 2025.
- [49] J. A. Montanez-Barrera, Y. Ji, M. R. von Spakovsky, D. E. B. Neira, and K. Michielsen, *Optimizing QAOA circuit transpilation with parity twine and SWAP network encodings*, 2025. arXiv: 2505.17944 [quant-ph].
- [50] D. Karger, R. Motwani, and G. D. S. Ramkumar, "On approximating the longest path in a graph," *Algorithmica*, 1997.
- [51] R. Wille, L. Berent, T. Forster, J. Kunasaikaran, K. Mato, T. Peham, et al., "The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing," in *Int'l Conf. on Quantum Software*, 2024. arXiv: 2405.17543, A live version of this document is available at <https://mqt.readthedocs.io>.
- [52] Qiskit contributors, *Qiskit: An open-source framework for quantum computing*, 2023.