Reducing the Compilation Time of Quantum Circuits Using Pre-Compilation on the Gate Level

Nils Quetschlich* Lukas Burgholzer* Robert Wille*†
*Chair for Design Automation, Technical University of Munich, Germany
†Software Competence Center Hagenberg GmbH (SCCH), Austria
nils.quetschlich@tum.de lukas.burgholzer@tum.de robert.wille@tum.de
https://www.cda.cit.tum.de/research/quantum

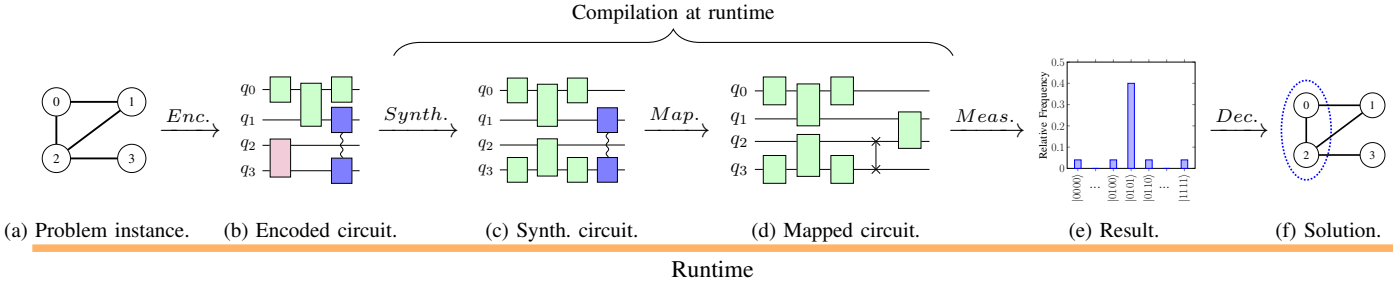
***Abstract*—In order to implement a quantum computing application, problem instances must be encoded into a quantum circuit and then compiled for a specific platform. The lengthy compilation process is a key bottleneck in this workflow, especially for problems that arise repeatedly with a similar yet distinct structure (each of which requires a new compilation run thus far). In this paper, we aim to overcome this bottleneck by proposing a comprehensive pre-compilation technique that tries to minimize the time spent for compiling recurring problems while preserving the solution quality as much as possible. The following concepts underpin the proposed approach: Beginning with a problem class and a corresponding quantum algorithm, a *predictive encoding* scheme is applied to encode a representative problem instance into a general-purpose quantum circuit for that problem class. Once the real problem instance is known, the previously constructed circuit only needs to be adjusted—with (nearly) no compilation necessary. Experimental evaluations on QAOA for the MaxCut problem as well as a case study involving a satellite mission planning problem show that the proposed approach significantly reduces the compilation time by several orders of magnitude compared to Qiskit's compilation schemes while maintaining comparable compiled circuit quality. All implementations are available on GitHub (https://github.com/cda-tum/mqt-problemsolver) as part of the *Munich Quantum Toolkit* (MQT).**

I. Introduction

Quantum computing is an emerging technology that is constantly improving both in software and hardware—sparking interest in academia and industry. As a result, many approaches have emerged that try to use quantum computing to solve problems from various application domains such as finance (e.g., [1]), machine learning (e.g., [2]), optimization (e.g., [3]), and chemistry (e.g., [4]). Dedicated *workflows* to describe the necessary steps from an initial problem to a solution using quantum computing are starting to emerge (e.g., [5]–[8]). These workflows usually include several steps: A suitable quantum algorithm has to be chosen for the respective problem and a quantum circuit needs to be constructed that encodes it. Then, this circuit must be compiled and executed on a targeted quantum device before the solution can be decoded from the results of the execution. Many *Software Development Kits* (SDKs) have been developed for conducting these steps (or at least some of them). Prominent examples are IBM's Qiskit [9], Quantinuum's TKET [10], Google's Cirq [11], and BQSKit [12]. Additionally, there are quantum computing platform providers that offer access to various quantum devices such as AWS Braket [13], and Microsoft's Azure Quantum [14] which implicitly take care of the compilation when executing quantum circuits.

Compilation, i.e., making sure that a particular quantum circuit can actually be executed on a targeted device, is an essential part of these workflows that involves many computationally hard problems such as gate synthesis [15]–[20] or qubit routing/mapping [21]–[28]. It is crucial to perform this task as efficiently as possible, since the possible overhead introduced by compilation directly correlates with the quality of the compiled circuit. However, this takes time. Time that in existing workflows is spent almost exclusively at "runtime", i.e., at the moment the entire problem instance is known. Furthermore, every single instance of a particular problem class is compiled from scratch—although its instances have a similar yet distinct structure and are frequently recurring, such as problems in scheduling [29] and finance [30].

In this work, we propose a *pre-compilation* technique that aims to drastically reduce the time spent on compilation at runtime while maintaining comparable quality of the compiled circuits. The proposed approach is based on the following ideas: Starting with a problem class (e.g., the MaxCut problem) and a quantum algorithm to solve the problem (e.g., the *Quantum Approximate Optimization Algorithm*, QAOA [31]), a *predictive encoding* scheme is applied that encodes the classes' structure (e.g., a QAOA circuit for fully connected graphs) into a general-purpose quantum circuit for the whole class of problems. This circuit is then (pre-) compiled to yield a generic circuit for solving problems in the respective class that can be executed on a certain device. At runtime, when the actual problem instance is known, the resulting circuit only needs to be adjusted for the particular instance (e.g., by removing gates corresponding to edges not present in the graph instance)—without requiring any compilation. As the general-purpose circuits have to subsume a wide variety of instances, their compilation is likely to induce a higher overhead compared to just compiling one particular problem instance. To counteract this potential loss in quality, some lightweight optimizations are performed at runtime in addition to adjusting the circuit.



(a) Problem instance. (b) Encoded circuit. (c) Synth. circuit. (d) Mapped circuit. (e) Result. (f) Solution.

Fig. 1: Established quantum computing workflow from a (MaxCut) problem to its solution.

Experimental evaluations on QAOA for the MaxCut problem as well as a real-world satellite mission planning use-case demonstrate the benefits of the proposed approach. For instances ranging from 5 to 100 qubits, the compilation time at runtime is reduced by several orders of magnitudes with a similar compiled circuit quality compared to the established workflow with all implementations available on GitHub (https://github.com/cda-tum/mqt-problemsolver) as part of the *Munich Quantum Toolkit* (MQT).

The remainder of this work is structured as follows: The currently established quantum computing workflow from a problem to a solution is reviewed in Section II. Subsequently, Section III motivates the proposed pre-compilation scheme and reviews related work. Based on that, Section IV describes the proposed compilation scheme in detail before it is evaluated in Section V. Afterwards, the proposed scheme is applied to a real-world industry use-case from satellite mission planning in Section VI and discussed in Section VII. Finally, Section VIII concludes this work.

II. The Quantum Computing Workflow

Quantum computing can be used to solve classical problems from various domains as discussed in Section I. Although the problems to be solved obviously vary, the general workflow from the problem to a solution is commonly structured as shown in Fig. 1.

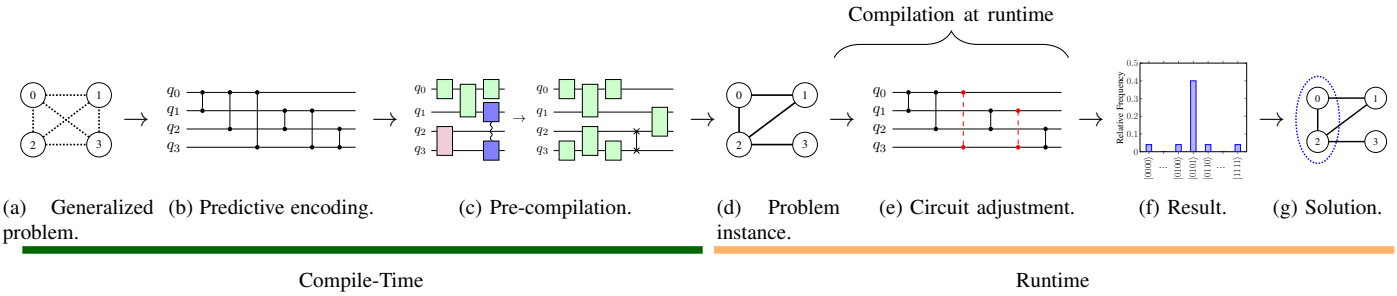
Given a classical problem instance of a certain problem class, the first step to solve the problem with quantum computing is to *encode* the problem into a quantum circuit that shall afterwards be executed on a quantum computer. This encoding step is highly non-trivial and constitutes a whole research area on its own (e.g., [32]–[34]). It requires the problem description to be transformed in a way that is suitable as an input for a certain quantum algorithm—with many degrees of freedom in the choice of an encoding and a particular algorithm.

Example 1. *Consider the graph shown in Fig. 1a and assume that we want to solve the well-known MaxCut problem on this graph, i.e., the goal is to find a partition of the graph's nodes so that the number of edges between these partitions is maximal. The* Quantum Approximate Optimization Algorithm *(QAOA, [31]) has been proposed as a candidate to tackle this problem on a quantum computer. For this, each node is encoded as a qubit and each edge between two nodes is encoded as a certain interaction between the respective qubits—as sketched in Fig. 1b.*

Once encoded as a quantum circuit, the circuit must be *compiled* to be executable on a particular quantum device. This is similar to classical programming, where a high-level program (e.g., written in C++) needs to be compiled into low-level machine instructions that are supported by the CPU on which the program should run. Quantum computers generally offer a small but universal set of gates that is natively supported by the device. In the following, this is referred to as *native gate-set*. Every quantum computation that shall be executed on a certain device must first be broken down into sequences of native gates—a process frequently referred to as *synthesis* with many respective methods proposed, e.g., in [15]–[20].

Example 2. *Consider again the circuit from Fig. 1b and assume that one of its five gates (the one marked in red) is not a native gate on the targeted quantum device. Then, Fig. 1c sketches what a synthesized circuit might look like, where that gate has been decomposed into three (native) gates (now marked in green).*

Now that the circuit consists only of native gates, the circuit's (logical) qubits need to be *mapped* to the device's (physical) qubits—a process frequently referred to as *qubit placement*, *qubit allocation*, or *qubit layout*. Many quantum computers, such as those based on superconducting qubits or neutral atoms, have a limited connectivity between their qubits, i.e., multi-qubit gates may only be applied to qubits that are connected on the device. As a result, the qubit mapping generally has to be adapted dynamically throughout the circuit. This is typically achieved by inserting *SWAP* operations in a process referred to as *qubit routing*. The whole process of determining an initial qubit placement and then routing the circuit is commonly referred to as *mapping* with methods proposed, e.g., in [21]–[28].



(a) Generalized problem. (b) Predictive encoding. (c) Pre-compilation. (d) Problem instance. (e) Circuit adjustment. (f) Result. (g) Solution.

Fig. 2: Proposed workflow including pre-compilation.

Example 3. *Consider the synthesized circuit shown in Fig. 1c and assume that each of the qubits is connected to its neighbors on the targeted device. Then, all but one gate (the one marked in blue) already adhere to the connectivity constraints. To resolve the remaining conflict, a single* SWAP *gate is inserted in front of the last gate as shown in Fig. 1d.*

The circuit is now executable on the targeted device—marking the end of the compilation step considered in this work[1]. It is essential to keep the overhead introduced by compiling the original circuit as low as possible, since each added gate decreases the fidelity of the overall result. The complexity of the underlying problems (e.g., mapping being NP-complete [38]) makes compilation a time- and resource-intensive task that is, nevertheless, crucial for reliably executing quantum circuits.

For the actual execution, the circuit is usually sent to a quantum device provider over the cloud and the result is sent back in the form of a histogram of measurement outcomes. These measurement outcomes are then post-processed (*decoded*) to reveal the solution to the problem.

Example 4. *An execution of the quantum circuit sketched in Fig. 1d might yield a distribution of measurement outcomes as shown in Fig. 1e. There, the measurement result ($|0101\rangle$) occurred with a significantly higher frequency than all other results—marking it as the solution. Since each qubit was chosen to encode one node of the graph, the value of each qubit indicates to which partition a particular node belongs. In this case, the solution groups $\{0, 2\}$ and $\{1, 3\}$ as shown in Fig. 1f, which, indeed, are a solution to the MaxCut problem on this graph.*

III. General Idea

The established workflow described in Section II and illustrated in Fig. 1 comes with a major bottleneck: The crucial and expensive compilation step is conducted *from scratch* during *runtime* for *every* single problem instance—leading to long runtimes for determining a solution for a particular problem instance. In the following, we propose a *pre-compilation* technique that aims to drastically reduce the time spent on compilation at runtime while maintaining comparable solution quality. This is illustrated in Fig. 2 and described next.

A. Motivation

For a particular problem class and a selected quantum algorithm, the process of taking a classical problem description and encoding it as a quantum circuit comes with an inherent structure that is independent of the specific instance to be solved. This can be turned into a *predictive encoding* that captures the structure of the whole problem class at once in the form of a general-purpose quantum circuit.

Example 5. *In case of the MaxCut problem to be solved with QAOA (as considered throughout Section II), the problem can be generalized to a problem class by making some kind of assumption on the properties of the graphs that are expected as input. The simplest case, essentially assuming that there are no restrictions, is to consider the* complete *graph as shown in Fig. 2a. Any MaxCut problem on four nodes can be derived from this generalized description. The predictive encoding is generated by applying the same encoding process to the generalized description as would have been applied to a specific instance. More precisely, a qubit is allocated for each of the nodes and a dedicated gate is added to the circuit for each connection between nodes. Fig. 2b figuratively sketches what such a all-to-all circuit might look like.*

Since the predictive encoding is, per definition, independent of the particular problem instance, it can be *pre-compiled*, i.e., determined at *compile-time* as opposed to at runtime once the particular instance is known. This creates a general-purpose, executable circuit for the problem class considered.

Example 6. *Consider again the predictive encoding circuit shown in Fig. 2b. Using pre-compilation, the entire compilation process is conducted and yields a pre-compiled and ready-to-execute circuit at compile-time as sketched in Fig. 2c.*

At runtime, once the problem instance is known, the previously compiled circuit needs to be *adjusted* to reflect the actual problem instance before being sent to the device provider for execution. This is significantly cheaper than a full compilation pass as the necessary adjustments frequently boil down to simple gate removals.

[1]Before a circuit can really be executed on a particular device, several more low-level steps such as pulse-level compilation and scheduling need to be conducted (using SDKs such as proposed in, e.g., [35]–[37]). These steps will not be considered in further detail in this work and are assumed to occur once the circuit is sent to the device provider for execution.

Example 7. *Assume that, at runtime, the same problem instance previously considered as the starting point in Fig. 1a should be solved. The respective graph (shown again in Fig. 2d) misses two edges compared to the complete graph that was used for the predictive encoding (shown in Fig. 2a). Therefore, the pre-compiled quantum circuit must be traversed and all compiled quantum gates involved in the anticipated but now non-present edges must be removed as shown in Fig. 2e (insinuated in red). Afterwards, the altered compiled quantum circuit can be sent for execution and the solution can be decoded as before—leading to the same histogram and solution—again visualized in Fig. 2f and Fig. 2g, respectively.*

Following this approach, much of the heavy burden of compilation can be shifted from runtime to compile-time. Furthermore, heavy and compute-intensive optimizations can be applied at compile-time, where it is not so critical to be fast as this has to be done only once.

However, there is no free lunch: The predictive encoding has to subsume a variety of instances which is likely to increase the compilation overhead as compared to compiling a single problem instance. As demonstrated by experimental evaluations (summarized later in Section V), this can be mitigated to some extent by performing lightweight optimizations after the circuit adjustment at runtime.

B. Related Work

So far, pre-compilation has hardly been explored—presumably, because compiling everything at runtime has been "good enough" for the scale of problems considered today. Existing works focus on *Variational Quantum Algorithms* (VQAs) and mostly rely on the concept of lookup tables that map uncompiled (sequences of) gates to their compiled equivalent.

In [39], already a decade ago, an approach to pre-compile certain rotation angles of single-qubit gates was proposed. Whenever a single-qubit with an arbitrary rotation angle occurs, it is compiled by concatenating a sequence of the pre-compiled rotation angles.

After years of no active research in this domain, further works have examined *pre-compilation* on the *pulse* level—describing how each quantum gate is translated into a sequence of electro-magnetic pulses when executed on the actual quantum device. In [40], an approach has been proposed targeting VQAs by exploiting their general circuit structure consisting of parameterized and non-parameterized gates. The authors pre-compile the non-parameterized gates to the pulse level and optimize them extensively. At runtime, only the parameterized gates need to be compiled and stitched together with the pre-compiled pulses.

Similarly, the authors of [41] focus on the pulse level and propose a *static/dynamic* hybrid workflow. This approach is based on storing pre-compiled pulses for certain groups of quantum gates in a database. Whenever a new quantum circuit is compiled, it is decomposed into groups and during the static phase, the database of pre-compiled pulses is searched for suitable pulses. If there are no pre-compiled pulses for all groups, the respective pulses are generated in the dynamic phase, before they are concatenated with the pulses of the static phase.

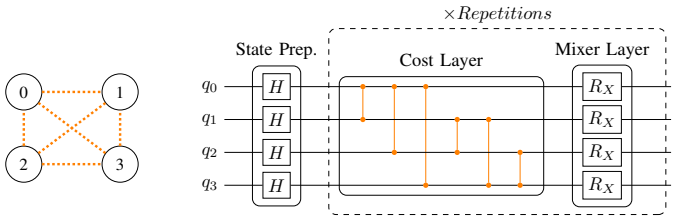


Fig. 3: Predictive Encoding for the MaxCut problem using QAOA assuming that all interactions are present.

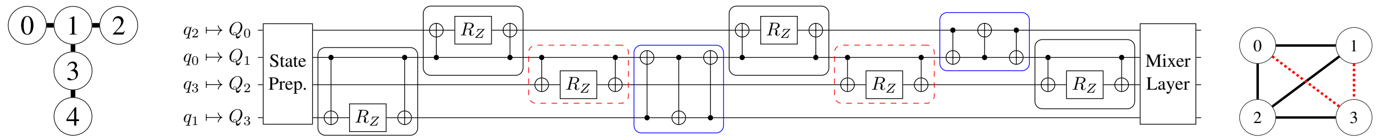
IV. Proposed Compilation Scheme

In this section, the proposed compilation scheme is explained in more detail—again using MaxCut as a running example. More specifically, the proposed compile-time and circuit adjustment steps are described.

A. Compile-time Steps

The compile-time steps comprise the (predictive) encoding and the (pre-) compilation, which in turn entails native gate-set synthesis and mapping as reviewed in Section II. So far, these steps have mostly been conducted at runtime after the problem instance is known. In contrast, a different approach has been chosen in the proposed scheme: instead of encoding the actual problem instance, a specific problem instance is anticipated—exploiting the inherent structure of the problem class and the selected quantum algorithm.

Example 8. *To illustrate the proposed predictive encoding, assume a MaxCut problem with four nodes, and therefore, six possible edges shall be solved using QAOA as already mentioned in Example 5. The general structure of the algorithm (shown in Fig. 3) consists of a sequence of three different building blocks: the state preparation, the cost layer, and the mixer layer—with the last two blocks being repeated a certain number of times. To this end, only the cost layer depends on the actual problem instance. It models the graph by representing each node as a qubit and applying* RZZ *gates between connected nodes, e.g., an edge between nodes* 0 *and* 1 *translates to a* RZZ *gate between qubits* q_0 *and* q_1*. By anticipating that all possible edges are present, the cost layer can be fully encoded.*



(a) *imbq_quito*.

(b) Mapped quantum circuit. The state preparation and mixer layer are problem-independent and their compiled gate sequences (as shown in Example 9) are encapsulated in respective blocks. Each SWAP gate (compiled to three CNOT gates) is denoted in a blue box, each anticipated and also actually present edge is denoted in a black box while the to-be-removed edges are denoted in red boxes.

(c) MaxCut problem instance.

Fig. 4: Mapping.

After the predictive encoding, all further compilation steps can be conducted in the same fashion as before—just with the difference that the resulting compiled quantum circuit does not represent the actual problem instance (which is not known yet) but an anticipated one. Using the predictive encoding scheme, pre-compilation is no longer restricted to the problem-independent parts of a general quantum algorithm (as in the existing techniques reviewed in Section III-B), but can be applied to the entire quantum circuit.

Example 9. *Assume that the encoded quantum circuit from Example 8 shall be executed on the five-qubit* ibmq_quito *device that offers R_Z, $\sqrt{X}$,* X, *and* CX *gates as its native gates and whose layout is shown in Fig. 4a. Then, each of the QAOA building blocks—state preparation (consisting of* H *gates), cost* (RZZ *gates), and mixer layers (R_X gates)—is first compiled to native gates using the following circuit identities:*

$$H = R_Z(\pi/2)\; X\; R_Z(\pi/2)$$

$$RZZ(\alpha) = CX \; (I \otimes R_Z(\alpha)) \; CX$$

$$R_X(\alpha) = R_Z(\pi/2)\; \sqrt{X}\; R_Z(\alpha+\pi)\; \sqrt{X}$$

After that, the circuit's qubits can be assigned to the device qubits as shown on the left-hand side of Fig. 4b. Consider a single repetition of the cost and mixer layer. Then, two SWAP *gates (which are, in turn, decomposed into sequences of three* CX *gates) are necessary to satisfy the connectivity constraints imposed by the architecture throughout the entire circuit. Eventually, this results in the mapped circuit shown in Fig. 4b with the compiled* SWAP *gates denoted in the blue boxes.*

After these steps have been executed, a fully executable general-purpose quantum circuit has been created—while the actual problem instance is not known yet.

B. Circuit Adjustment Step

At runtime, the problem instance is known and the goal is to make sure that the fully encoded, compiled, and mapped circuit is altered such that it actually solves the problem instance. For that, the difference between the encoded and the actual problem instance must be determined. Subsequently, the already compiled quantum circuit must be adjusted based on the determined difference.

Example 10. *Consider the same MaxCut instance as in Example 7. It comprises four of the six anticipated edges as illustrated in Fig. 4c (missing edges are denoted as dashed red lines). Consequently, the pre-compiled quantum circuit (shown in Fig. 4b) needs to be adjusted and the (compiled) quantum gates representing the two missing edges must be removed—as highlighted by red boxes each representing one of the to-be-removed edges (while the black boxes represent the actual present edges). The easiest way to accomplish this—requiring only a linear traversal of the circuit—is by simply setting the angles of all R_Z gates corresponding to the to-be-removed edges to zero. This adjustment effectively makes the corresponding R_Z gates no-ops and hence leads to the applications of the remaining CX gates to cancel each other out. The resulting circuit is executable and solves the underlying problem—with barely any modifications to the pre-compiled circuit necessary at runtime.*

All these benefits do not come for free. As stated previously, not only is it important for a circuit to be executable and solve the problem, but the overhead introduced by compilation should be as low as possible. For most systems, this overhead can be quantified by the number of two-qubit gates in the resulting circuit. Since predictive encoding anticipates many different problem instances, it is highly likely that its compilation induces a larger overhead compared to compiling a single problem instance. As such, the proposed pre-compilation scheme offers a trade-off between the time spent during compilation at runtime and the quality of the resulting circuit. Applying optimizations at runtime allows making this trade-off almost continuous. The heavier optimizations are applied at runtime, the closer the performance will typically get to the established workflow—at the cost of longer runtimes.

Example 11. *Consider again the circuit from Example 10. Instead of only setting the angles of the R_Z gates to zero during circuit adjustment, a lightweight optimization removes these gates entirely from the circuit and tries to cancel subsequent CX gates that act on the same qubits. Considering Fig. 4b, this completely eliminates the gates in the red boxes.*

Compared to directly compiling the circuit for the problem instance (as in the established workflow shown in Fig. 1), the proposed scheme results in a circuit containing one more SWAP *gate. Note that some parts of the introduced* SWAP *gates can be further canceled with parts of some* RZZ *gates via the lightweight gate optimization—bringing the overhead down to a bare minimum.*

As shown by experimental evaluations, which are summarized next, this can reduce the compilation time at runtime by several orders of magnitude while maintaining comparable compiled circuit quality.

V. Experimental Evaluation

This section evaluates the proposed compilation scheme for varying problem instances and compares both the compilation time at runtime and the quality of the compiled quantum circuits. All implementations are available on GitHub (https://github.com/cda-tum/mqt-problemsolver) as part of the *Munich Quantum Toolkit* (MQT).

A. Setup

For the evaluation, the MaxCut problem described throughout all examples is implemented based on Qiskit (v0.41.1) in *Python* with various parameters:

- Number of nodes considered (and therefore qubits): 5 to 100 with a step size of 5.
- Chosen algorithm: QAOA with 3 repetitions.
- Targeted devices: ibmq_quito, ibmq_montreal, and ibmq_washington with 5, 27, and 127 qubits, respectively. Each problem instance is compiled to the smallest but sufficiently large device.
- Predictive encoding assumption: Edges are possible between all qubit pairs ($e = all$) (as shown in the example in Example 11) or each node can have at most one possible edge to its immediate successor ($e = 1$).
- Problem instance creation: With sample probability $p \in \{0.3, 0.7\}$ a possible edge is actually present. The higher p, the more anticipated interactions will be present after the problem instance is revealed.

All problem instances evaluated are assessed by two criteria:

1) Compilation time at runtime and
2) Compiled circuit quality—determined by the number of present two-qubit gates representing the compilation overhead induced by the proposed approach with lower being better.

For comparison, IBM's Qiskit compiler is used with *optimization levels* $O0$ to $O3$ following the quantum workflow (mentioned in Fig. 1) while the compile-time steps of the proposed approach have been conducted with Qiskit's highest optimization level $O3$. Additionally, it is spot-checked (using MQT QCEC [42]) that the compiled and adjusted quantum circuits created by the proposed approach are equivalent to the created circuits of the baselines.

B. Results

The findings of this study are summarized in Fig. 5. For each combination of e and p values—four combinations in total—the compilation time at runtime and the resulting compiled circuit quality have been evaluated and denoted side-by-side, e.g., in Fig. 5a and Fig. 5b for the combination $e = 1$, $p = 0.3$.

The results show that the proposed compilation scheme is faster than all Qiskit compilations for all the combinations evaluated of e and p as indicated in the left figures of Fig. 5. When comparing the time reduction depending on the sample probability p, it is noteworthy that the improvement is higher for both $p = 0.7$ cases compared to $p = 0.3$ cases. The higher p is, the closer the predicted encoding is to the actual problem instances, and consequently, less compilation overhead is induced.

The respective qualities of the compiled circuits are indicated on the right of Fig. 5. The proposed approach, again, outperforms Qiskit's $O0$ and is on par or only slightly worse than the most optimized Qiskit compilation $O3$ in most cases. When again comparing the $p = 0.7$ cases to the $p = 0.3$ cases, $p = 0.7$ leads to a smaller overhead in terms of the number of two-qubit gates and even outperforms Qiskit's $O3$ for many evaluated problem instances for $e = all$ (as shown in Fig. 5h).

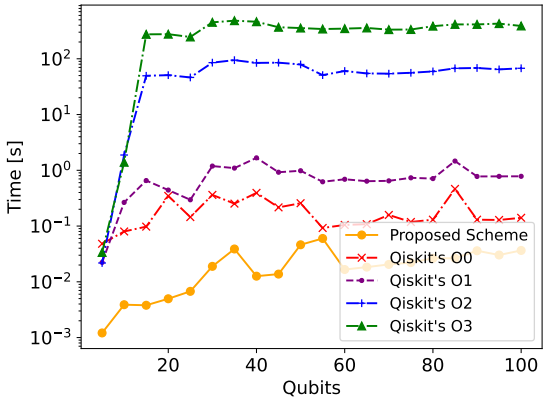
The evaluations demonstrate that the proposed compilation scheme achieves a promising trade-off between compilation time and compiled circuit quality. For scenarios where the compilation time at runtime is of prime importance, Qiskit's fastest compilations are the baselines to beat—which the proposed approach is strictly doing for all problem instances. On top of that, the compiled circuit quality is always better than Qiskit's $O0$ and in the vast majority of cases comparable to Qiskit's most optimized $O3$ compilation—which comes with an inferior compilation time by several orders of magnitude and, by that, is often not a viable option in those scenarios.

VI. Application: Satellite Mission Planning Problem

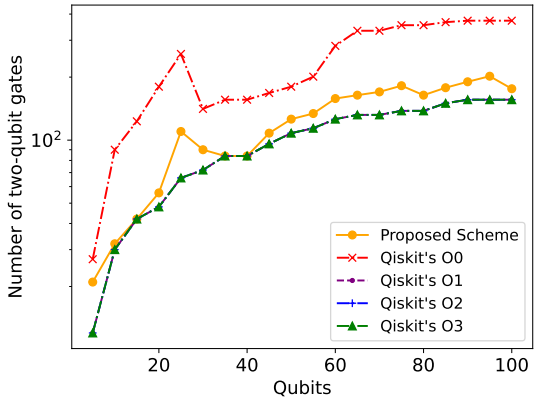
So far, the benefits of the proposed compilation scheme have been shown and evaluated based on the rather academic MaxCut problem. In this section, an application from the space industry (taken from [43]) is introduced and the proposed compilation scheme is applied to highlight the benefit it offers for a real-world use-case.

A. Motivation

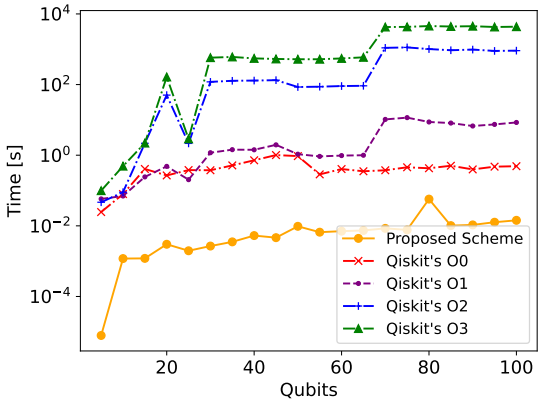
Currently, there are hundreds of satellites orbiting the Earth with the goal of photographing certain locations. However, each satellite usually has only a very narrow field-of-view and, therefore, needs to physically rotate its optics between capturing different locations while it is moving on its orbit with constant speed. As a consequence, it may not be possible to take images of all to-be-captured locations. Additionally, the time to determine which locations to select is critical and must stay within a fixed time budget—otherwise the result is worthless, since the satellite may already passed the first location.



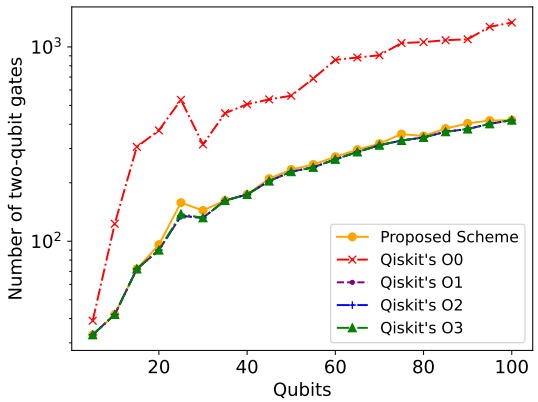
(a) Compilation time at runtime: $e = 1$ and $p = 0.3$.



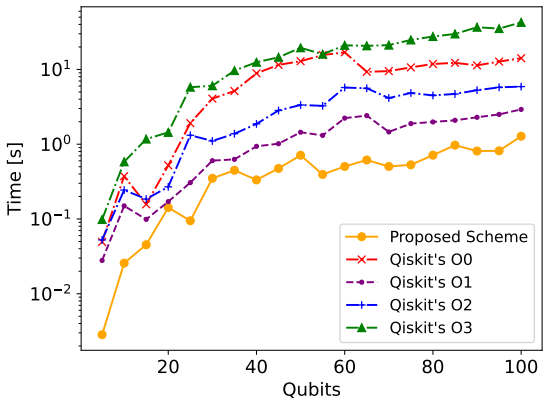
(b) Compiled circuit quality: $e = 1$ and $p = 0.3$.



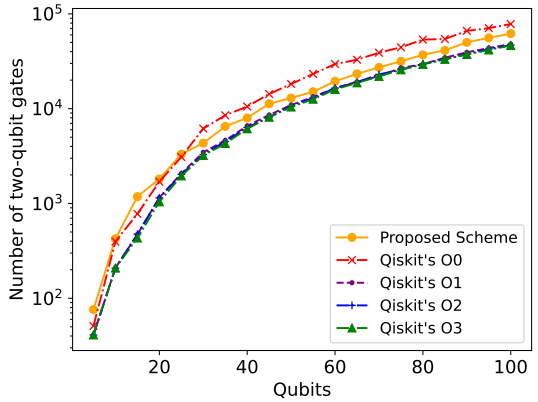
(c) Compilation time at runtime: $e = 1$ and $p = 0.7$.



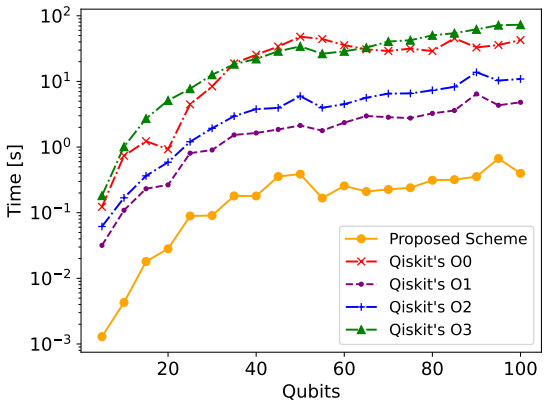
(d) Compiled circuit quality: $e = 1$ and $p = 0.7$.



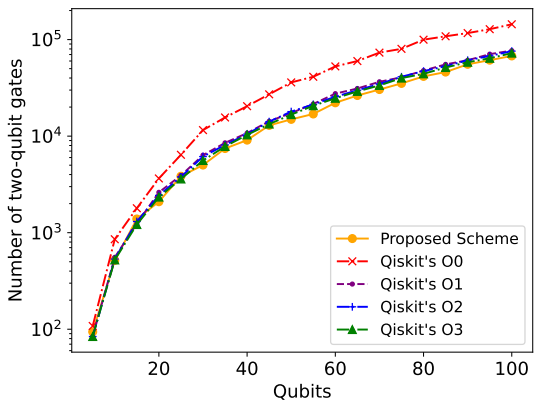
(e) Compilation time at runtime: $e = all$ and $p = 0.3$.



(f) Compiled circuit quality: $e = all$ and $p = 0.3$.



(g) Compilation time at runtime: $e = all$ and $p = 0.7$.



(h) Compiled circuit quality: $e = all$ and $p = 0.7$.

Fig. 5: Experimental evaluation of the proposed approach for various parameter choices.

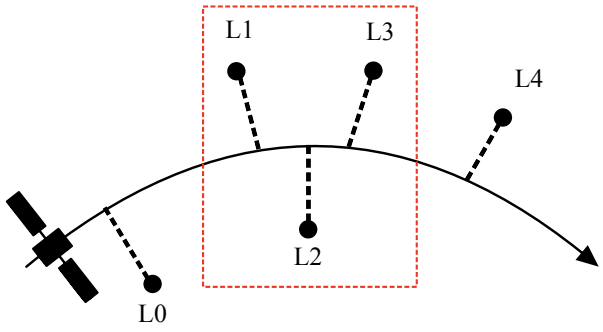


Fig. 6: Satellite mission planning problem with five to be imaged locations ($L0$ to $L4$) with two infeasible location pairs: $L1$/$L2$ and $L2$/$L3$.

Example 12. *Consider a satellite that is tasked to take photos of five locations ($L0$ to $L4$) as shown in Fig. 6. While the locations $L0$ and $L1$ are distant and provide enough time for the physical rotation of the optics, there is a selection problem for locations $L1$ to $L3$: If the location $L1$ is selected, it is infeasible to also select the location $L2$. Furthermore, if location $L2$ is selected, location $L3$ becomes infeasible. Therefore, the maximum number of feasible locations is achieved by selecting all except $L2$—resulting in four captured locations.*

B. Quantum Computing Approach

In [43], this problem has already been approached using a quantum annealing ansatz. Therefore, a problem formulation as a *Quadratic Unconstrained Binary Optimization* (QUBO) problem was proposed.

In addition to the quantum annealing approach, this problem can also be solved using gate-based quantum computing, e.g., using QAOA. For that, a simplified problem formulation compared to [43] has been chosen where each location can be captured from one position on the orbit. By that, each location is encoded as one qubit representing if that location has been selected for capturing or not—with the goal of capturing as many locations as possible while still being physically feasible.

Since QAOA is used—similar as in Fig. 3—also the circuit structure for the satellite mission planning problem is similar: Whenever two locations cannot be feasibly selected, there is a *RZZ* gate (similar to the edges between nodes in the MaxCut problem). The main difference to the MaxCut QAOA example is a weight factor for every qubit (encoded by a R_Z gate) to represent the dependencies between selecting varies locations as shown in Fig. 7 in its uncompiled form for the problem instance depicted in Fig. 6. Additionally, all *RZZ* gates in the problem layer are multiplied by a factor γ determined by the specific problem instance and its translation to an *Ising Hamiltonian*.

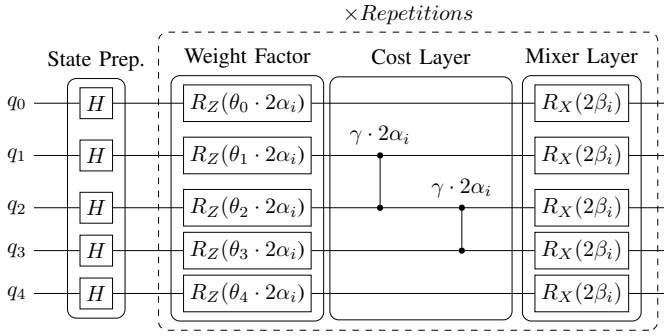


Fig. 7: Satellite problem encoded using QAOA.

C. Experimental Evaluation

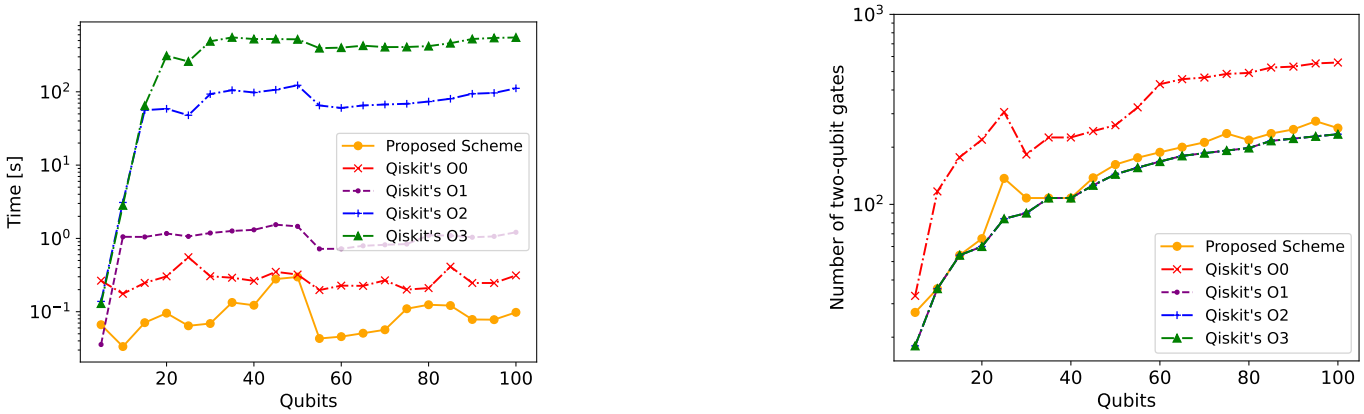
The satellite mission planning problem is evaluated with similar parameters as the MaxCut problem:

- Number of locations considered (and, therefore, qubits): 5 to 100 with a step size of 5.
- Chosen algorithm: QAOA with 3 repetitions.
- Targeted devices: ibmq_quito, ibmq_montreal, and ibmq_washington with 5, 27, and 127 qubits, respectively. Each problem instance is compiled to the smallest but sufficiently large device.
- Predictive encoding assumption: Each pair of subsequent locations is either feasible or infeasible ($e = 1$).
- Problem instance creation: With sample probability $p = 0.4$ a pair of to-be-captured locations is infeasible.

Using the compilation scheme proposed in Section IV results in a significantly reduced compilation time at runtime compared to Qiskit's compilation schemes (as depicted in Fig. 8a) while maintaining comparable compiled circuit quality (as depicted in Fig. 8b)—similar to the results of the experimental evaluations described in Section V. The proposed approach is almost always faster than all baselines while the difference becomes even more distinct with an increasing number of qubits. Regarding the compiled circuit quality, the proposed compilation scheme always outperforms Qiskit's $O0$ and is only slightly worse than Qiskit's most optimized $O3$ compilation—which is inferior by several orders of magnitude in compilation time.

VII. Discussion

All those mentioned benefits do not come for free. For the predictive encoding, a suitable quantum algorithm and a respective encoding of the problem class must be determined and applied. This alone is challenging and requires expert knowledge in both quantum computing and the problem domain—constituting a whole research domain on its own. The closer the anticipated problem instance comes to the actual problem instance, the better will be the compilation quality.



(a) Compilation time at runtime: $p = 0.4$.

(b) Compiled circuit quality: $p = 0.4$.

Fig. 8: Experimental evaluation for the satellite use-case.

Additionally, the encoding scheme must be chosen so that the circuit adjustment step—which adjusts the already fully executable pre-compiled quantum circuit to represent the actual problem instance and not the anticipated one—can be conducted efficiently. Therefore, it is recommended to only *delete* quantum gates from the pre-compiled circuit. Inserting new gates could, in the worst case, destroy the mapping so that this compilation step needs to be re-done—impairing the desired compilation time at runtime improvement. Hence, it heavily influences the trade-off between reduced compilation time and compiled circuit quality—if it is either slow or ineffective, the benefits of the proposed compilation scheme dissolves.

To adapt the proposed scheme for arbitrary problems, some one-time manual effort is required to setup the predictive encoding based on the assumptions made. The degree of required work needed depends very much on the selected quantum algorithm. For some algorithms such as the *Variational Quantum Eigensolver* (VQE, [4]) it is simple since the whole quantum circuit is problem-independent and can be pre-compiled already by the approaches mentioned in Section III-B. For other algorithms such as *Grover* [44] or *Quantum Phase Estimation* (QPE, [45]) it is less straight-forward to find efficient assumptive problem instances, since it is harder to find an overarching anticipated problem instance that can be encoded as a general-purpose circuit where all other possible problem instances can be derived from.

VIII. Conclusions

Compilation is an essential—but time-consuming—part of quantum workflows solving problems from any kind of application domain and is currently almost exclusively conducted at runtime. Additionally, every single problem instance is compiled from scratch, even though they share a similar structure. This becomes even more severe for frequently recurring problems.

In this paper, we proposed a comprehensive pre-compilation scheme to reduce the compilation time at runtime based on predictive encoding at compile-time and a respective circuit adjustment at runtime. For that, a general-purpose quantum circuit is created—subsuming a wide variety of problem instances—and pre-compiled. As soon as the actual problem instance becomes known, the pre-compiled circuit is adjusted to represent the actual problem and not the anticipated one.

Experimental evaluations on QAOA for the MaxCut problem as well as a case study involving a satellite mission planning problem show that this reduces the compilation time at runtime by several orders of magnitude compared to Qiskit's compilation schemes while maintaining comparable compiled circuit quality. All implementations are available on GitHub (https://github.com/cda-tum/mqt-problemsolver) as part of the *Munich Quantum Toolkit* (MQT).

However, implementing this proposed pre-compilation scheme comes with a considerable one-time effort, since a suitable predictive encoding scheme to create the general-purpose circuit is highly dependent on the problem class and the selected quantum algorithm to solve it. In addition, the encoding scheme must be chosen so that the circuit adjustment step can be conducted efficiently. Therefore, this work constitutes a first step towards a general comprehensive pre-compilation scheme, but further steps are necessary.

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 BMWK on the basis of a decision by the German Bundestag through project QuaST, as well as by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

[1] N. Stamatopoulos *et al.*, "Option pricing using quantum computers," *Quantum*, 2020.

[2] C. Zoufal, A. Lucchi, and S. Woerner, "Quantum Generative Adversarial Networks for learning and loading random distributions," *npj Quantum Information*, 2019.

[3] S. Harwood, C. Gambella, D. Trenev, A. Simonetto, D. Bernal Neira, and D. Greenberg, "Formulating and Solving Routing Problems on Quantum Computers," *IEEE Transactions on Quantum Engineering*, 2021.

[4] A. Peruzzo *et al.*, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, 2014.

[5] N. Quetschlich, L. Burgholzer, and R. Wille, "Towards an Automated Framework for Realizing Quantum Computing Solutions," in *Int'l Symp. on Multi-Valued Logic*, 2023.

[6] B. Poggel, N. Quetschlich, L. Burgholzer, R. Wille, and J. M. Lorenz, "Recommending Solution Paths for Solving Optimization Problems with Quantum Computing," in *Int'l Conf. on Quantum Software*, 2023.

[7] N. Quetschlich, L. Burgholzer, and R. Wille, "Predicting Good Quantum Circuit Compilation Options," in *Int'l Conf. on Quantum Software*, 2023.

[8] N. Quetschlich, L. Burgholzer, and R. Wille, "Compiler Optimization for Quantum Computing Using Reinforcement Learning," in *Design Automation Conf.*, 2023.

[9] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023. [Online]. Available: https://doi.org/10.5281/zenodo.2573505.

[10] S. Sivirajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "TKET: A Retargetable Compiler for NISQ devices," *Quantum Science and Technology*, 2020.

[11] C. Developers, *Cirq*, version v1.2.0, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8161252.

[12] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, E. Smith, and USDOE, "Berkeley quantum synthesis toolkit (bqskit) v1," 2021. [Online]. Available: https://www.osti.gov/biblio/1785933.

[13] *Amazon Braket Python SDK*, Amazon Web Services, 2022. [Online]. Available: https://github.com/aws/amazon-braket-sdk-python.

[14] *Azure Quantum documentation*, Microsoft, 2022. [Online]. Available: https://learn.microsoft.com/en-us/azure/quantum/.

[15] B. Giles and P. Selinger, "Exact synthesis of multiqubit Clifford+T circuits," *Physical Review A*, 2013.

[16] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2013.

[17] 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.

[18] A. Zulehner and R. Wille, "One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2018.

[19] P. Niemann, R. Wille, and R. Drechsler, "Advanced exact synthesis of Clifford+T circuits," *Quantum Information Processing*, 2020.

[20] T. Peham, N. Brandl, R. Kueng, R. Wille, and L. Burgholzer, "Depth-optimal synthesis of Clifford circuits with SAT solvers," in *Int'l Conf. on Quantum Computing and Engineering*, 2023.

[21] M. Y. Siraichi, V. F. dos Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Int'l Symp. on Code Generation and Optimization*, 2018.

[22] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2019.

[23] A. Matsuo and S. Yamashita, "An efficient method for quantum circuit placement problem on a 2-D grid," in *Int'l Conf. of Reversible Computation*, 2019.

[24] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Design Automation Conf.*, 2019.

[25] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.

[26] T. Peham, L. Burgholzer, and R. Wille, "On Optimal Subarchitectures for Quantum Circuit Mapping," *ACM Transactions on Quantum Computing*, 2023.

[27] S. Hillmich, A. Zulehner, and R. Wille, "Exploiting Quantum Teleportation in Quantum Circuit Mapping," in *Asia and South Pacific Design Automation Conf.*, 2021.

[28] A. Zulehner and R. Wille, "Compiling SU(4) quantum circuits to IBM QX architectures," in *Asia and South Pacific Design Automation Conf.*, 2019.

[29] H. Mohammadbagherpoor *et al.*, "Exploring airline gate-scheduling optimization using quantum computers," 2021. arXiv: 2111.09472.

[30] R. Orús, S. Mugel, and E. Lizaso, "Quantum computing for finance: Overview and prospects," *Reviews in Physics*, 2019.

[31] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014. arXiv: 1411.4028.

[32] F. Dominguez, J. Unger, M. Traube, B. Mant, C. Ertler, and W. Lechner, "Encoding-independent optimization

problem formulation for quantum computing," 2023. arXiv: 2302.03711.

[33] M. Weigold, J. Barzen, F. Leymann, and M. Salm, "Encoding patterns for quantum algorithms," *IET Quantum Communication*, 2021.

[34] M. Schuld and F. Petruccione, "Machine learning with quantum computers." Springer, 2021.

[35] T. Alexander *et al.*, "Qiskit pulse: Programming quantum computers through the cloud with pulses," *Quantum Science and Technology*, 2020.

[36] B. Li *et al.*, "Pulse-level noisy quantum circuits with QuTiP," *Quantum*, 2022.

[37] H. Silvério *et al.*, "Pulser: An open-source package for the design of pulse sequences in programmable neutral-atom arrays," *Quantum*, 2022.

[38] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Symposium on Combinatorial Search*, 2018.

[39] D. Kudrow *et al.*, "Quantum rotations: A case study in static and dynamic machine-code generation for quantum computers," *SIGARCH Comput. Archit. News*, 2013.

[40] P. Gokhale *et al.*, "Partial compilation of variational algorithms for noisy intermediate-scale quantum machines," in *Int'l Symposium on Microarchitecture*, 2019.

[41] J. Cheng, H. Deng, and X. Qian, "Accqoc: Accelerating quantum optimal control based pulse generation," 2020. arXiv: 2003.00376.

[42] T. Peham, L. Burgholzer, and R. Wille, "Equivalence checking of parameterized quantum circuits: Verifying the compilation of variational quantum algorithms," in *Asia and South Pacific Design Automation Conf.*, 2023.

[43] T. Stollenwerk, V. Michaud, E. Lobe, M. Picard, A. Basermann, and T. Botter, "Image acquisition planning for earth observation satellites with a quantum annealer," 2020. arXiv: 2006.09724.

[44] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proc. of the ACM*, 1996.

[45] A. Y. Kitaev, "Quantum measurements and the Abelian Stabilizer Problem," 1995. arXiv: arXiv : quant - ph / 9511026.