17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

Evolving Spacecraft Quantum On-Call Scheduling

Sven Prüfer*, Wanja Sajko†, Nikolas Pomplun‡, Andreas Spörl§
Deutsches Zentrum für Luft- und Raumfahrt e. V., German Aerospace Center
Münchener Straße 20, 82234 Weßling, Germany

In this paper we will show recent progress in the development of Spacecraft Quantum On-Call Scheduling (SQOS) which uses Grover's algorithm on a quantum computer to solve a simplified scheduling problem for on-call shifts of operators at the German Space Operations Center (GSOC). In particular, we reduce gate costs of the longest paths of the resulting quantum circuits by up to 90%.

The problem itself is an instance of a combinatorial optimization problem with constraints, which is currently solved in operations using a heuristic approach with some random variation. While the problem itself is not specific to spacecraft operations, there is a multitude of other applications that might be solved using similar techniques, such as ground station or earth observation planning.

Besides an overview of the on-call scheduling problem and a broad introduction to quantum computing, we describe the following optimizations of SQOS: Improving constraint implementations using difference-based comparisons, counting constraint violations in a Quantum Fourier Transform (QFT) basis, implementing inequality constraints on sliding windows using a quantum register buffer, local quantum circuit optimizations such as using phase-wrong gates, and implementing Grover's algorithm without knowing the number of solutions beforehand.

While some of these points are rather specific to the SQOS problem, they give a rough idea of techniques that one may employ to implement a planning problem on a quantum computer more efficiently. This paper may be seen as a source of potential ideas to optimize quantum circuits implementing concrete scheduling algorithms.

Keywords: Scheduling, Planning, Quantum Computing, Algorithms, Optimization

Acronyms/Abbreviations

Acronym/Abbreviation	Meaning
DLR	Deutsches Zentrum für Luft- und Raumfahrt e.V.
GSOC	German Space Operations Center
NISQ	Noisy Intermediate Scale Quantum
QAOA	Quantum Approximate Optimization Algorithm
QCI	Quantum Computing Initiative
QFT	Quantum Fourier Transformation
SQOS	Spacecraft Quantum On-Call Scheduling
VQE	Variational Quantum Eigensolver

I. Introduction

On-Call operator scheduling at GSOC is an instance of the nurse scheduling problem, see e. g. [1–3] for a general overview and quantum approaches. In this paper we report on improvements to a Grover algorithm solution that can be

*Mission Planning System Engineer, Mission Technology department, German Space Operations Center, Sven.Pruefer@dlr.de
†Mission Operations Technology Working Student, Mission Technology department, German Space Operations Center, Wanja.Sajko@dlr.de
‡Mission Operations Technology Engineer, Mission Technology department, German Space Operations Center, Nikolas.Pomplun@dlr.de
§Deputy Head of Mission Operations Technology Team, Mission Technology department, German Space Operations Center, Andreas.Spoerl@dlr.de

Table 1 Constraints for a valid on-call schedule. This table was first published in [5]

	Description
I	Operators can only work on some given positions.
II	Per tuple of day and position at least one operator needs to be assigned.
III	An operator can be assigned to at most one position a day.
IV	Operators can specify days in advance when they are unavailable.
V	A partial on-call schedule may be supplied and needs to be obeyed.
VI	Operators can work at most two out of any three consecutive weeks.
VII	Operators can work at most 35 days out of any 105 consecutive days.
VIII	Operators shall work preferably whole weeks.
IX	All operators shall work a similar amount of days

run on so-called NISQ (noisy intermediate scale quantum) devices.

This section introduces briefly the on-call scheduling problem at GSOC in Section I.A and gives a rough overview about quantum computing Section I.B, in particular a high-level description of Grover's algorithm. For a more thorough introduction and prior work on SQOS, see e. g. [4] and [5].

This work was partially supported by the Quantum Computing Initiative [6].

A. On-Call Scheduling

On-Call Scheduling is a shift scheduling problem that is common in many industries such as healthcare, transportation, and logistics. Here, the objective is to assign a set of *operators* to a set of satellite sub systems (called *positions*) for every required *time slot* while satisfying a set of constraints (e.g. availabilities, capabilities, and various legal rules). There are various aspects of a schedule one may optimize for, one example is to minimize the variation of the number of scheduled shifts per operator. While SQOS currently does not optimize anything, it incorporates most relevant constraints for practical applications at GSOC and the recent past was spent on reducing the number of quantum gates needed to implement those. The main reason for this is to be able to run sample instances of an interesting size on real NISQ hardware.

In Table 1 you can find a list of relevant constraints within SQOS. Depending on the problem input, these constraints define appropriate quantum gates that are added by SQOS to a quantum circuit, see Section I.B for the general idea. SQOS then runs this circuit on a simulator or an actually quantum computer and returns the most probable result to the caller. This result then defines a valid on-call schedule according to the above constraints.

B. Quantum Computing and Grover's algorithm

Quantum computing is a branch of computer science that exploits quantum effects to allow for new algorithms that may potentially solve certain problems asymptotically faster than current classical solutions. Currently, we see a lot of progress in various implementations of quantum computers, such as superconducting circuits [7], trapped ions [8], neutral atoms [9] or photonic quantum computers [10]. As there are still lots of different engineering challenges remaining, it is not clear which technology might be the eventual winner. In particular, DLR is now commissioning industry developing different quantum hardware approaches in Germany on behalf of the German Federal Ministry For Economic Affairs and Climate Action, see [6, 11].

One well-known algorithm in quantum computing is Grover's algorithm [12], which allows finding a marked item in an unsorted list with a quadratic speed-up compared to classical algorithms. The search is space is encoded in the Hilbert space of the quantum computer such that the marked item corresponds to some identifiable state. It then repeatedly amplifies the amplitude of the target state to increase the probability to measure it. This is done by two steps: First, one marks the target state by flipping its phase. This is done via a *Grover oracle* which depends on the problem at hand. Second, the state is reflected at a certain average state of the search space. This reflection is problem-independent and referred to as a *Grover diffuser*. Overall, this has the effect that the amplitude of the target state is increased compared to all the other states, so by repeating this procedure one can achieve high probabilities of measuring the target state.

See Figure 1 for a visual description of the circuit, Figure 2 for a basic step-by-step illustration how Grover amplifies amplitudes of marked states and [13] for a more detailed introduction as well as the relevant computations.

To apply Grover's algorithm to a constraint satisfaction problem, one can proceed as follows:

1) Encode the problem definition space on the Hilbert space of the quantum computer
2) Add a counter quantum register, that is, a few qubits that will be used to count the number of constraint violations of a state.*
3) For every constraint implement a small circuit that adds 1 to the counter register in case a state violates the constraint.
4) After going through all constraints, flip the phase of the state in case the counter register is different from zero.
5) After the phase-flip revert the additions such that the counter register is zero again for the next Grover iteration.

If the number of valid states is known, e. g. the percentage of valid states of the whole state space is p, then the ideal number of Grover iterations asymptotically maximizing the probability of measuring a valid state is given by

$$N = \frac{\pi}{4}\sqrt{\frac{1}{p}}, \tag{1}$$

so for problems with very few solutions this becomes very large, causing problems for today's qubits and quantum gates due to their error-rate. Usually, however, this number is not known or computable, so for real applications one needs to devise a scheme how to gradually increase the number of iterations to find a valid state as fast as possible. This aspect is analyzed in Section II.C.1.

Regarding the problem encoding, we use the original one from [4], that means every time slot–position tuple has a quantum register consisting of a few qubits that encode the operator in a binary fashion that works the position at the time slot. So, for example, in case of 4 operators, 2 positions and 3 days the basis state $|110111001000\rangle$ means that operator 3 (i. e. 11 in binary) works on day 0 and 1 position 0, whereas operator 0 (i. e. 00 in binary) works on day 1 and 2 in position 1, see Figure 3 for an overview.

II. Improvements

In this section, we present several improvements to our implementation of Grover's algorithm for on-call operator scheduling for satellites. They can be loosely grouped in the following three categories: In Section II.A we describe two improvements regarding the implementation of constraint checks, whereas Section II.B contains techniques that improve all parts of the quantum circuit by making the individual building blocks more efficient. Last, in II.C we look at two more general methods that can be used to build a Grover quantum circuit.

A. Constraint Implementation

The constraints in the on-call operator scheduling problem represent the requirements that must be satisfied in a valid schedule. Implementing these constraints efficiently is crucial for the performance of the algorithm, as it determines how many qubits and gates are needed.

*If the register is too small, there might be an overflow causing the algorithm to return invalid schedules

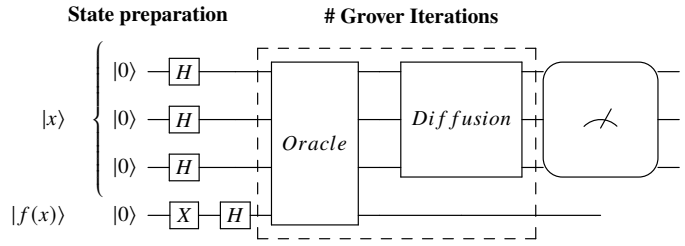

Fig. 1 Major steps of Grover's algorithm. The oracle depends on the concrete problem and, in case of SQOS, is built such that we count constraint violations in a register and if that register is zero, we flip the phase, thus marking the searched state. Notice that the phase-flip qubit at the bottom of the circuit can in fact be left out. This diagram was first shown in [5]

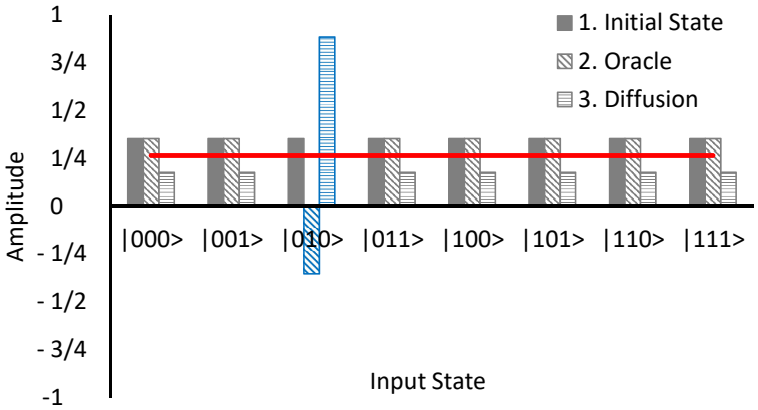

Fig. 2 Illustration of Grover Diffusion with the example of $n = 3$ for the search string `010`. One can see that after a Grover iteration the amplitude of the searched-for state increases and hence it is more likely to measure it. This figure was first shown in [4]

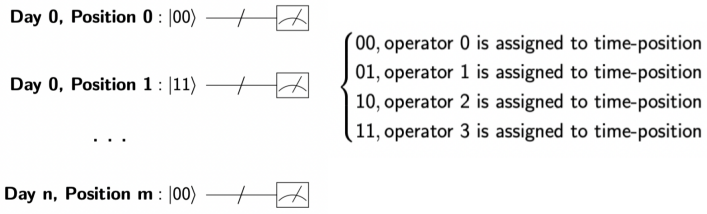

Fig. 3 Illustration of the search space encoding in SQOS. This diagram was first shown in [5]

1. Using difference-based compare operations

Consider constraint III from Table 1: No operator shall be scheduled for more than one position per day. To implement this constraint, we use a global counter register that is incremented each time an operator is assigned to more than one position per day. In [5], this was done by comparing for each operator and day individually whether they are scheduled for more than one position. In this revised version, we make use of the following idea: We can compare two operators that are scheduled for two positions on the same day by doing a bitwise XOR operation, which corresponds to controlled X gates on the qubits encoding the operators. This is easily reversible and uses much smaller gates compared to the multi-controlled Toffoli gates used in [5] for the same purpose. If there are p positions, one can thus determine if the first position is occupied by the same operator as another position, by comparing the values in the last $p - 1$ positions against $|0\rangle$ and incrementing the counter register in case they are equal.

The circuit in Figure 4 demonstrates this technique for one day with two operators and three positions. In the old version, there were $do\binom{p}{2}$ incrementors, where d is the number of days, o the number of operators, and p the number of positions, as we add an incrementor for any day, operator and combination of two different positions. Also notice that these incrementors are rather large as the controls need to check for a specific operator twice. In contrast, the new method uses $d\binom{p}{2}$ incrementors, which is a significant reduction. While we still need to check on every day, that each pair of distinct positions is not occupied by the same operator, we no longer have to repeat this step for each operator. But of course we have additional bitwise CX gates, which are, however, much cheaper. A more detailed analysis of gate costs can be found in Section III.

2. Using new buffer register for sliding windows

Constraints VI and VII enforce a maximum of working days during a time window for operators. Such a constraint is referred to as a *sliding window* constraint as only the length of the time frame is specified, but in fact there are multiple

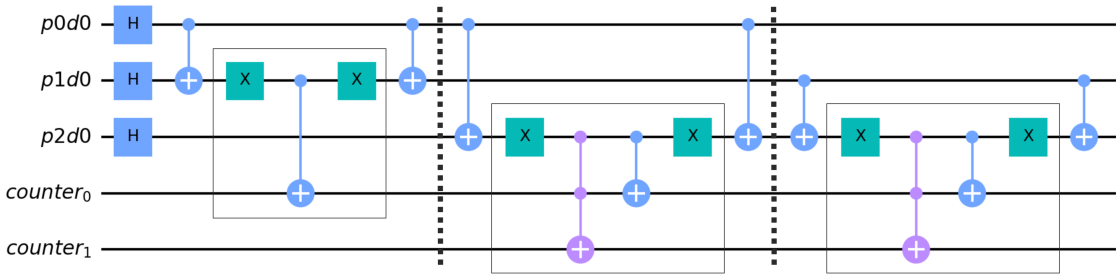

Fig. 4 Sample circuit for difference-based comparisons. The operators scheduled on position 0 and 1 on day 0 are compared by a bitwise CX operation at the beginning. After an incrementation in case of a zero, this operation is reversed and the operators on position 0 and 2, as well as 1 and 2, are compared in the same way. Notice that the incrementors are slightly optimized as the first one can only reach 1, so a smaller incrementor suffices

such time windows which slide over the whole time horizon. Both constraints are inequality constraints of the form

$$\sum_{d \in w} f(d) \leq M, \tag{2}$$

where w is any sliding window, that is time interval of L consecutive days[†], M is the upper bound, d is a day within the sliding window, and $f : D \longrightarrow \mathbb{Z}$ measures the quantity that is bounded. Notice that this means that we have a lot of these constraints: For example, for a 365-day schedule, 20 operators and Constraint VI the function f counts the number of positions a specific operator is working on a specific day. As we have 345 windows, we get 6900 inequality constraints where each one contains 14 summands. The implementation from [4] considers $M = 2$ and $L = 3$, and works by counting a constraint violation if any operator works three consecutive days.

Notice that there exists a classical implementation that makes use of a resource profile: At every point in time t we can sum the function f over the time window $[t - L, t]$, call this function $F : D \longrightarrow \mathbb{Z}$. This means that the sliding window constraint is in fact an upper bound of M for the resource function F, see Figure 5 for an illustration. At every step of the figure one can determine the value at the next time step by subtracting the resource value that is moving outside the window and adding the one that enters it. Assuming a constraint of maximum value of 3 in Figure 5 during such a sliding window, we count a conflict whenever we measure 4, i. e. whenever the blue line intersects with the orange one. One can easily see that we may under count the amount of constraint violations but at the same time do not need to worry about potential overflows of the counter register Even though this solution is classical, it provides us with an idea of how to improve our quantum computer implementation of such constraints.

To do so, notice that from day d to $d + 1$ the resource decreases by $f(d)$ and increases by $f(d + 1)$, we can hence deduce the value of a shifted sliding window by buffering the value and modifying it slightly. However, to do so, we need a buffer register, i. e. we need to use some additional qubits for this purpose. If we are only interested in whether we violate the constraint (instead of counting or identifying all constraint violations), we can reduce the size of the buffer register, however. This is because we can count a constraint violation whenever the buffer register has value $M + 1$ and ignore any overflow in the register. It thus suffices to use $\overline{\log_2(M + 1)}$ qubits to fit the number $M + 1$ in the register.

The implementation therefore has to iterate over the first sliding window per operator, adding one to the buffer register whenever the operator is scheduled and execute constraint checks after every summation. It then modifies the buffer as described above for each shifted sliding window and does a new constraint check. Notice that in case of an enforced constraint III it is possible to exclude the buffer reaching $M + 1$ until $M + 1$ days have been checked, so in this case one can even omit any constraint checks for the first M days. As the buffer register is reused it needs to be cleared again, however, once it is zeroed it can actually be used for auxiliary qubits in other constraints, too. See Figure 6 for an example implementation. The circuit uses small optimizations: As long as the buffer cannot possibly be 3 there are no constraint violation counters, also the first one can only reach 1, so a smaller incrementor suffices. You can see in Figure 6 the initial sliding window together with one shifted sliding window where the buffer is decremented by the

[†]It is possible that sliding windows may overlap with the planning horizon only partially. This is for example useful when extending schedules because then a sliding window may overlap both with the planning horizon and the already existing schedule

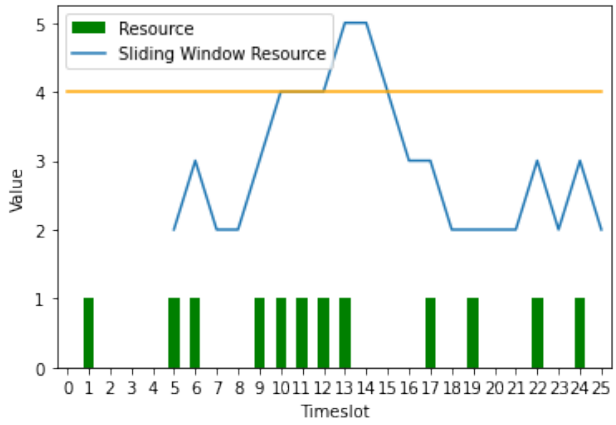

Fig. 5 The green bars show a binary resource over time, whereas the blue line shows the corresponding sliding window resource that sums the values over a time range of 6 in this example.

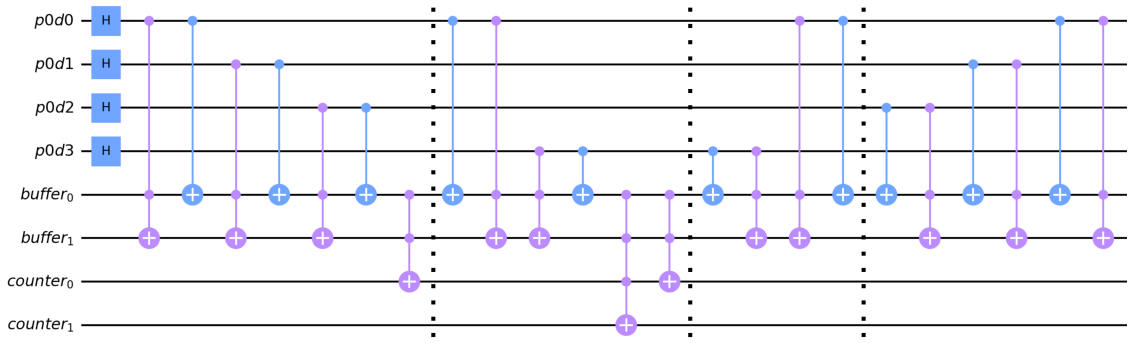

Fig. 6 Sample circuit showing an implementation of a sliding window constraint. This is for 1 position, 2 operators and 4 days, the sliding window has length 3 and maximum 2, shown are two sliding windows for the operator $|1\rangle$. The constraint violation counter checks thus on 3 which is 11 in binary

same value it was incremented initially and incremented if the value in $p0d3$ is $|1\rangle$. At the end, the circuit is reversed to make the buffer zero again for the next operator

B. Constraint Violation Counter

As the biggest parts of the quantum circuit are built from circuits that add one to the counter register in case a certain logical condition is satisfied, it is useful to make these *constraint violation counter* circuits cheaper. One solution is to use gate-optimization libraries to improve these building blocks individually. For this paper, however, we implemented two concrete improvements that do not rely on generic optimization methods. First, we reduced the size of used incrementors by classically excluding counter value ranges, thus using smaller multi-controlled Toffoli gates. Second, we replaced the multi-controlled Toffoli gate based counters by Quantum Fourier Transformation ([13, 14]) based ones to more efficiently count constraint violations. These updates provide a more efficient means of counting constraint violations, which is crucial for ensuring the integrity of the Grover algorithm.

1. Using Incrementing Counter Operations

When building the quantum circuit for SQOS, it is necessary to find an upper bound on the number of possible constraint violations in order to add an appropriate amount of counter qubits. However, after n constraint-based incrementors the counter register can be at most n, thus there is no need to cover all counter qubits. As multi-controlled Toffoli based incrementors are compositions of incrementors for each digit, one can reduce their size and only implement those gates that are needed for the current maximum number of constraint violations. This is basically achieved by

looking at the classical problem and then keeping track of the current number of maximum constraint violations. In total, this allows us to use smaller incrementors in steps where the maximum number of conflicts is lower, which reduces the overall size of the circuit. Figure 7 illustrates an example circuit in which we increment the counter register six times. Notice that the size of the incrementors increases logarithmically with the number of steps, rather than being constant.

2. Using Quantum Fourier Transformation Based Counters

Another alternative is to replace the multi-controlled Toffoli based incrementors by switching to a Fourier basis and incrementing the counter there. As the constraint logic is still implemented using controls, the building blocks are basically multi-controlled +1-gates, but each such +1-gate is a set of rotation gates, implementing this operation in the Fourier basis.

In Figure 8, a basic Fourier-based addition operation is illustrated. Notice that one may need to keep track of the used basis within the quantum circuit as one might need to perform a Fourier transformation on the state of the register to switch to the Fourier basis. However, when comparing with numbers in the standard computational basis, one needs to apply an inverse Fourier transformation first.

To evaluate this optimization, the usage of QFT-based adders was made configurable. This way, we can compare the gate costs of the various approaches in Section III. Also notice that this approach is not combinable with the incrementing-incrementor approach from Section II.B.1. This is because increasing a number by one in QFT_{n-1} base is very different from increasing by one in QFT_n, for example because the least-significant bit rotates a different angle.

C. General Optimizations

The optimizations in this section are aimed to improve the performance, and are not specific to any particular constraint or aspect of the problem.

1. Automatic Determination of Number of Grover Iterations

When using Grover's algorithm one needs to repeat the oracle and diffusion step a number of times, the optimal number for which depends on the percentage of valid solutions. Unfortunately, this number is usually not known in advance, and hence we cannot expect to find a solution with near 100 % probability on one run. For [5] this problem was not addressed, so we implemented an algorithm from [15]. Notice that this solution adapts the algorithm in such a way, that the expected number of total Grover iterations necessary to find a solution, is still of the same asymptotic order as a normal Grover algorithm.

Intuitively, the algorithm works by assuming a large percentage of solutions (and thus a low ideal number of Grover iterations), and in case of no found solutions then tries a larger number of iterations. It does so in a way that estimates for the expected run time can be done.

Notice that a lower (upper) bound on the percentage of valid solutions yields an upper (lower) bound on the ideal number of Grover iterations. This can be used to further improve the algorithm by restricting the used iterations to this interval. In the case of SQOS, it is possible to determine an upper bound for the number of solutions using the following idea. All individual incrementors probe for a specific bit string excluding a subspace of solutions by incrementing the constraint violation counter. By constructing maximal overlaps of these bit strings one can estimate a minimal number of solutions that get excluded, thus giving rise to an upper bound of solutions. However, this argument depends on the precise way constraints are implemented, so we will not present a detailed analysis of its advantages in this paper.

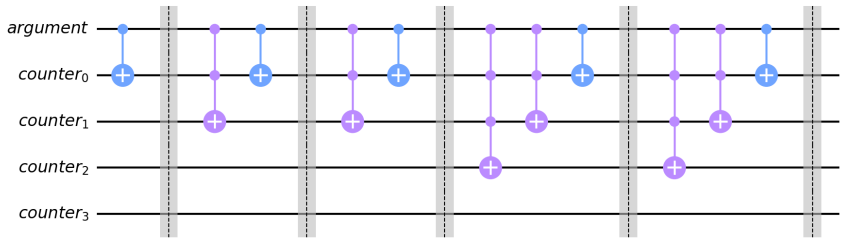

Fig. 7 Sample circuit using efficient incrementors

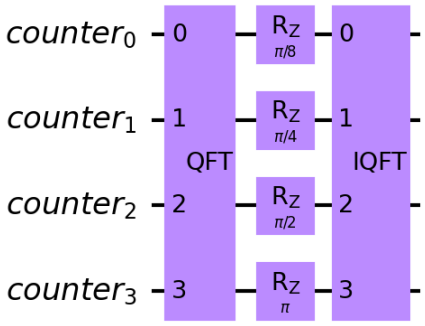

Fig. 8 Sample circuit for a QFT-based incrementor on a counter register of four qubits. Notice that there are no controls shown, and the picture includes QFT transformations for changing the basis. This means that the actual +1-operation in the QFT basis consists only of single qubit rotation gates of different angles

2. Using Phase-Wrong Gates

As the Grover oracle tests at its heart only that the number of constraint violations is zero, i. e. we control for a single basis state, any phase factors that other states pick up, don't matter to the algorithm. This allows us to replace any controlled operation by one that this identical up to a phase, hoping that such a phase-wrong implementation uses fewer gates.

For example, [16] describe a method in which multi-controlled gates can be compiled into a subset of basis gates, which produce the same outcome when measured but do not keep the same phase. Using this method we have been able to reduce the number of basis gates required for CCX gates, and it is possible to apply this to other relevant gates, too. See [16] for sample circuits replacing controlled gates.

III. Analysis

In this section we will see how much gate costs we are able to save by employing the techniques from Section II. Furthermore, we will compare the optimized implementations with the original ones from [5].

Before going into detail, however, let us explain a bit how we compare gates for the different quantum circuits as there are various choices to be made. Depending on the quantum hardware used, there is only a certain subset of gates realized on the machine, so the circuit needs to be transpiled to this gate set. Furthermore, these gates will be typically either single-qubit or multi-qubit, where those for multiple ones are expected to be significantly worse (that is more error-prone) than the single-qubit ones. We thus choose to transpile to the set of basis gates U1, U2, U3, CX, RX, SX, ID, and X which are all single-qubit except for CX. To make CX more expensive we weight it with a factor of 10 and then count all gates in this manner. However, for the stability of qubits the total sum of those gates does not matter, instead the gate cost of the *longest path* of any qubit in the circuit is relevant as this is an estimate of the maximum amount of gates that act on any qubit in the circuit. So any *gate cost* of a quantum circuit Q that we compare in this section is based on this calculation

$$\text{Cost}(Q) = \#\left\{\text{U1, U2, U3, RX, SX, ID, X gates } \in p_{\text{longest}}(\tilde{Q})\right\} + 10 \cdot \#\left\{\text{CX gates } \in p_{\text{longest}}(\tilde{Q})\right\}, \tag{3}$$

where $\tilde{Q}$ denotes the unrolled circuit and p_{longest} picks the longest path in the circuit when seen as a directed acyclic graph.

A. Constraint Implementation

In Figure 9 you can see a large reduction of gate costs when using buffer-based sliding window implementations. This is mainly due to the fact that multi-controlled Toffoli gates scale very badly with their number of qubits, it is thus nearly always a good idea to minimize their usage. In some way one buys this reduction of gate costs at the expense of additional qubits, but this is not as bad as it sounds: For the full problem with constraint VII one has an upper bound of 35, requiring thus 6 qubits for the buffer register which seems to be a reasonable price for a reduction of gate costs by

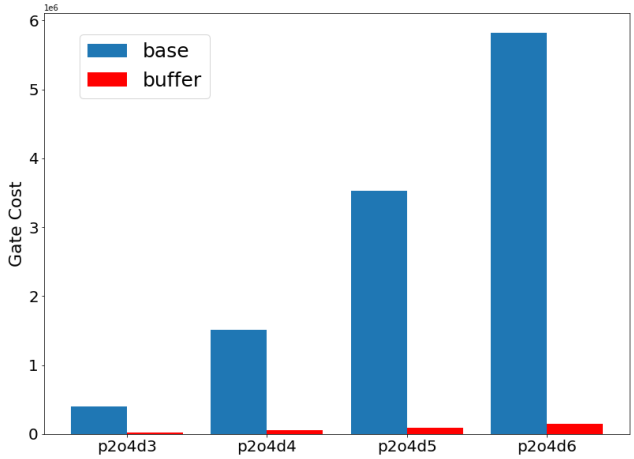

Fig. 9 Gate cost comparison between base and buffer-based sliding windows. Notice that the scale of the vertical axis is 100.000**. The reduction in gate costs by using a buffer register to save an intermediate value of the sliding window is quite large**

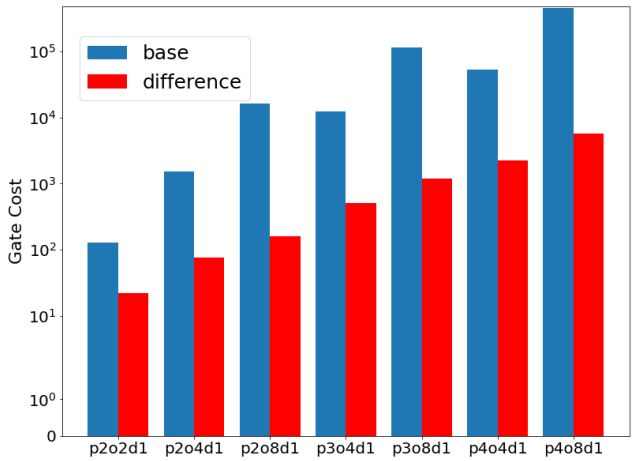

Fig. 10 Gate cost comparison between base and difference-based compare operations. The vertical scale on this graph is logarithmic, so we see a reduction of a factor of roughly 100 **in some basic cases**

more than a factor of 20. As was described in Section II.A.2 we may not count every constraint violation as we only check against specific values. Somewhat surprisingly, this might be useful as this means that the maximum number of constraint violations reachable is lower, and thus we may use a smaller counter register. However, it is difficult to get concrete bounds on this idea, so we were not yet able to reduce the counter size by this argument.

The difference-based comparisons in Section II.A.1 are not usable for every constraint, but for e. g. constraint III it is useful. Still, the rather large improvements can be seen in Figure 10. The high jump of the base implementation with increasing operator count is again very much due to the large number of big multi-controlled Toffoli gates that are replaced by CX gates in case of difference-based comparisons. Notice that the gate costs scale linearly for days as the whole building block needs to be repeated for every day separately. As the maximum reachable number of constraint violations is the same, the number of incrementors is the same for both solutions, however there are much fewer controls needed for the difference-based comparisons.

B. Constraint Violation Counter

In Figure 11 one can see how costs of multi-controlled Toffoli gate based incrementors and QFT-based ones scale with counter size. As the QFT-based incrementor is nearly a drop-in replacement (up to QFT transformations required to change the computational basis), one can see that using QFT-based incrementors reduces gate costs considerably and

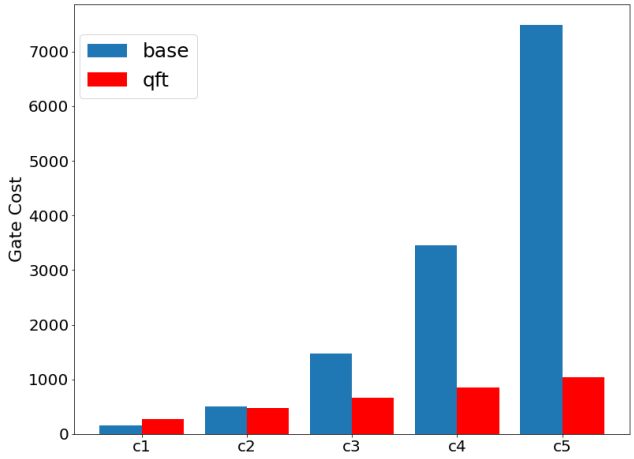

Fig. 11 Gate cost comparison between base and QFT-based incrementors. On the horizontal axis one can see the size of the counter register, whereas the number of control qubits remains constant 3

even more so in case of large counter registers as needed for real-world examples. With increasing register size the costs per multi-controlled Toffoli gate based incrementors increases dramatically, mainly due to the recursive nature of large multi-controlled Toffoli gates. Notice that an increase in control qubits increases costs of both incrementor types but does so in the same way, meaning that Figure 11 is representative.

C. Sample Cases

In [5] we ran simulations for multiple problem samples and evaluated them regarding the number of required qubits. With all the improvements from Section II we ran the simulations again to see how much gate costs we were able to save and to show that the optimizations still work.

See Table 2 for a table of the sample cases from [5] together with unoptimized and optimized gate costs. Here, optimization refers to the usage of difference-based comparisons Section II.A.1, efficient incrementors Section II.B.1 and buffer-based sliding windows Section II.A.2. We did not include the QFT-based incrementors Section II.B.2 improvement because transpilation took extremely long for these circuits. One change since [5] was to remove the phase-flip qubit as it is not needed to actually perform the flip. In the new table there is thus one less qubit, however, with the buffer for the sliding window we require two new additional qubits, explaining the different numbers. Furthermore, we list the gate costs as calculated without the optimizations today, so using the same library versions and phase-flip constructions. Notice that we don't list values for the case IV of 6 days which is due to the fact that the transpilation of the whole circuit takes too long.

Overall, there is a tremendous reduction in gate costs even though we have not yet used the QFT-based incrementors. The precise value is varying quite a lot and this may be due to multiple factors. For example, the number of days changes the amount of sliding windows that are needed and hence the percentage of gates that is due to these constraints. On the other hand, we measure the longest path as explained earlier, so adding gates in some other path might not even change these costs. We still need some more detailed analysis (in particular for sample case III), but for multiple days it might be the case that the longest path comes from the buffer register for the sliding-window constraint, hence an optimization there such as using a QFT-adder may be very beneficial.

IV. Conclusion and Future Work

In conclusion, we presented improvements on a Grover quantum algorithm for solving the on-call operator scheduling problem at the German Space Operations Center. Our approach takes into account various constraints, such as the need to avoid scheduling conflicts and the maximum number of consecutive working days for operators. These improvements comprise a redesign of some constraint implementations, changes to the counting of constraint violations using a quantum Fourier transformation, and general improvements to the circuit design. While still not applicable to real-world problems, one can see that it is indeed possible to solve scheduling problems using Grover's algorithm on a quantum computer.

Table 2 Simulation results for different problem sizes used for the evaluation. The gate costs shown are defined by Equation (3)

Case	I	II	III	IV	V	X
Operators	4	4	4	4	8	4
Positions	2	2	2	2	2	3
Days	3	4	5	6	3	3
Base Used Qubits	15	20	24	28	20	22
Optimized Used Qubits	17	22	26	30	22	24
Used Counter Qubits	3	4	4	4	2	4
Grover Iterations	1	2	3	5	2	2
Base Gate Costs	1.05×10^6	1.04×10^7	6.87×10^7	-	2.15×10^7	1.37×10^7
Optimized Gate Costs	1.02×10^5	2.29×10^6	5.07×10^7	-	8.59×10^6	8.63×10^6
Reduction of Costs	90 %	78 %	26 %	-	60 %	37 %

As for future work, there are several avenues that could be explored to continue improving the performance of the algorithm. One possibility would be to investigate other quantum algorithms, such as the quantum approximate optimization algorithm (QAOA [17]) or various variational quantum eigensolver (VQE [18]) techniques, and compare their performance to that of Grover's algorithm. Even using Grover, it would be interesting to explore variations that use a quantum-classical hybrid solution to improve the performance, such as e. g. Nested Grover Search ([19]). Another important part of future work is to continue investigating the scalability of the algorithm with respect to the problem input size. One approach to address this issue would be to divide the problem into smaller fragments, such as by days or weeks, and then run the quantum algorithm as a subroutine within a classical framework. This would allow for the efficient handling of larger scheduling problems by breaking them down into smaller, more manageable subproblems. Regarding gate-count-optimization strategies, there still remain quite a few possibilities, such as using small phase-optimized Grover algorithms [20] to implement certain constraints, or using black-box libraries that can optimize often-used blocks in the algorithm.

All in all, there remain many directions to explore in quantum computing and problems to apply these techniques to.

References

[1] Burke, E. K., De Causmaecker, P., Berghe, G. V., and Van Landeghem, H., "The state of the art of nurse rostering," *Journal of scheduling*, Vol. 7, No. 6, 2004, pp. 441–499.

[2] Ikeda, K., Nakamura, Y., and Humble, T. S., "Application of quantum annealing to nurse scheduling problem," *Scientific reports*, Vol. 9, No. 1, 2019, pp. 1–10.

[3] Aickelin, U., and White, P., "Building Better Nurse Scheduling Algorithms," *CoRR*, Vol. abs/0803.2967, 2008. URL `http://arxiv.org/abs/0803.2967`.

[4] Prüfer, S., Scherer, A., Spörl, A., Guggemos, T., Pomplun, N., and Lenzen, C., "Quantum Shift Scheduling - A Comparison to Classical Approaches," *12th International Workshop on Planning and Scheduling for Space (IWPSS 2021)*, edited by S. Chien, 2021. URL `https://elib.dlr.de/145763/`.

[5] Scherer, A., Guggemos, T., Grundner-Culemann, S., Pomplun, N., Prüfer, S., and Spörl, A., "OnCall Operator Scheduling for Satellites with Grover's Algorithm," *Computational Science – ICCS 2021*, edited by M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, Springer International Publishing, Cham, 2021, pp. 17–29.

[6] "Quantum Computing Initiative," `https://qci.dlr.de/`, 2022.

[7] Krantz, P., Kjaergaard, M., Yan, F., Orlando, T. P., Gustavsson, S., and Oliver, W. D., "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, Vol. 6, No. 2, 2019, p. 021318. doi:10.1063/1.5089550, URL `https://doi.org/10.1063%2F1.5089550`.

[8] Bruzewicz, C. D., Chiaverini, J., McConnell, R., and Sage, J. M., "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, Vol. 6, No. 2, 2019, p. 021314. doi:10.1063/1.5088164, URL https://doi.org/10.1063%2F1.5088164.

[9] Henriet, L., Beguin, L., Signoles, A., Lahaye, T., Browaeys, A., Reymond, G.-O., and Jurczak, C., "Quantum computing with neutral atoms," *Quantum*, Vol. 4, 2020, p. 327. doi:10.22331/q-2020-09-21-327, URL https://doi.org/10.22331/q-2020-09-21-327.

[10] Slussarenko, S., and Pryde, G. J., "Photonic quantum information processing: A concise review," *Applied Physics Reviews*, Vol. 6, No. 4, 2019, p. 041303. doi:10.1063/1.5115814, URL https://doi.org/10.1063/1.5115814.

[11] "Bundesministerium für Wirtschaft und Klimaschutz," https://www.bmwk.de/, 2022.

[12] Grover, L. K., "A Fast Quantum Mechanical Algorithm for Database Search," *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, NY, USA, 1996, pp. 212–219. doi:10.1145/237814.237866, URL https://doi.org/10.1145/237814.237866.

[13] Nielsen, M. A., and Chuang, I. L., *Quantum Computation and Quantum Information: 10th Anniversary Edition*, Cambridge University Press, 2010. doi:10.1017/CBO9780511976667.

[14] Coppersmith, D., "An approximate Fourier transform useful in quantum factoring," , 2002. doi:10.48550/ARXIV.QUANT-PH/0201067, URL https://arxiv.org/abs/quant-ph/0201067.

[15] Boyer, M., Brassard, G., Høyer, P., and Tapp, A., "Tight Bounds on Quantum Searching," *Fortschritte der Physik*, Vol. 46, No. 4-5, 1998, pp. 493–505. doi:10.1002/(sici)1521-3978(199806)46:4/5<493::aid-prop493>3.0.co;2-p, URL https://doi.org/10.1002%2F%28sici%291521-3978%28199806%2946%3A4%2F5%3C493%3A%3Aaid-prop493%3E3.0.co%3B2-p.

[16] Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., Sleator, T., Smolin, J. A., and Weinfurter, H., "Elementary gates for quantum computation," *Physical Review A*, Vol. 52, No. 5, 1995, pp. 3457–3467. doi:10.1103/physreva.52.3457, URL https://doi.org/10.1103%2Fphysreva.52.3457.

[17] Farhi, E., Goldstone, J., and Gutmann, S., "A Quantum Approximate Optimization Algorithm," , 2014. doi:10.48550/ARXIV.1411.4028, URL https://arxiv.org/abs/1411.4028.

[18] Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P. J., Aspuru-Guzik, A., and O'brien, J. L., "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, Vol. 5, No. 1, 2014, p. 4213.

[19] Cerf, N. J., Grover, L. K., and Williams, C. P., "Nested quantum search and NP-hard problems," *Applicable Algebra in Engineering, Communication and Computing*, Vol. 10, No. 4-5, 2000, pp. 311–338.

[20] Toyama, F. M., van Dijk, W., Nogami, Y., Tabuchi, M., and Kimura, Y., "Multiphase matching in the Grover algorithm," *Phys. Rev. A*, Vol. 77, 2008, p. 042324. doi:10.1103/PhysRevA.77.042324, URL https://link.aps.org/doi/10.1103/PhysRevA.77.042324.