INSTITUT FÜR INFORMATIK

DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor Thesis

Spacecraft Operator Scheduling with Grover's Algorithm

Antonius Benedikt Anani Scherer

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Spacecraft Operator Scheduling with Grover's Algorithm

Antonius Benedikt Anani Scherer

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Sophia Grundner-Culemann Tobias Guggemos Dr. Andreas Spoerl (DLR) Sven Pruefer (DLR)
Abgabetermin:	19. Januar 2021

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Berlin, den 19. Januar 2021

. . . .

(Unterschrift des Kandidaten)

Abstract

The application of quantum algorithms on some problems in NP promises a significant reduction of time complexity. This thesis uses Grover's Algorithm, originally designed to search an unstructured database with quadratic speedup, to find valid solution bit-strings to the NP-hard personnel scheduling problem. Under consideration of various hard and soft constraints, we implement this by using the IBMQ backend and Qiskit to optimize the German Aerospace Center's spacecraft on-call operator scheduling. We seek an optimal assignment for 52 operators to 17 positions over a period of 180 days under constraints on schedule and personnel. Further, we evaluate the solution quality and compare the performance with classical and quantum alternatives. While still restricted by intermediate-scale quantum devices in the near term, we explore new approaches in encoding and batching the problem to reduce the required number of qubits. In the end, a feasible near-term solution that scales well with the quantum devices of the upcoming years is presented.

Contents

1 Introduction

In 1982, both Richard Feynman [Fey82] and Paul Benioff [Ben82] independently pointed out that quantum systems could be used to perform computation. While Benioff's motivation was founded on the idea to prove the existence of a reversible Turing Machine, Feynman realized that due to its complexity, a quantum system could only be efficiently simulated by another quantum system. This sparked the idea of a new field known as quantum computing. Even though classical computers improved rapidly over the last decades and as Moore's law seems to hold, there are known limitations on what classical programs can solve efficiently. Theoretical computer scientists classified problems that are neither solvable in polynomial time or space, independent from further development of classical computers. Due to their complexity, classical algorithms do often not find an optimal solution in a feasible time or use metaheuristic methods that only find approximate solutions. Extending Benioff's and Feynman's ideas, quantum computing offers a potential solution to these restrictions. In the last three decades, researchers developed quantum algorithms that theoretically provide polynomial or even exponential speedups in comparison to their classical counterparts. A promising application of quantum algorithms are optimizations, especially complex combinatorial optimization problems. However, current quantum devices are very limited in their size and computational power. Even though those NISQ[1] era devices are not yet capable of having a deep impact on industry size problems, we use them to develop and study scalable quantum algorithms that will unfold their potential with improving quantum hardware.

Scope of the Thesis

Like many employers, the German Aerospace Centre (DLR) [2] faces difficulties while creating the personnel schedule for their employees. This thesis investigates the ability of Grover's search algorithm to solve a specific instance of DLR's personnel scheduling problem, the on-call spacecraft operator scheduling. While handling a variety of spacecraft missions from DLR's mission control center, the OnCall spacecraft operator schedule ensures the constant presence of capable operators on their dedicated missions. Here, an efficient assignment is crucial to ensure a frictionless performance as well as a reduction in personnel cost. Therefore we develop a method that ensures an efficient translation of the underlying optimization problem to quantum hardware using IBM's quantum developer kit Qiskit. Knowing that NISQ era devices are not yet capable of providing enough computational power to solve the whole problem instance, we ensure that the method is scalable and able to use the full power of future quantum hardware.

[1]Noisy Intermediate-Scale Quantum, or NISQ, is a term coined by John Preskill. It describes the era where quantum computers surpass the abilities of classical computers, but won't be big enough to provide fault-tolerant implementations of those underlying quantum algorithms. [Pre18]

[2]Deutsches Zentrum für Luft- und Raumfahrt

Structure of the thesis

The thesis consists of five parts: In Chapter 2 we give an introduction to quantum computing and explain Grover's search algorithms, quantum complexity theory, and pin down the scheduling problem. Chapter 3 describes the on-call operator scheduling problem and performs a requirement analysis for the method we develop for the DLR. In Chapter 4 we give an overview of related work that solves scheduling problems classically and uses quantum algorithms to solve optimization problems. Further, Chapter 5 describes the two core methods we develop to efficiently map our problem to a quantum device. Lastly, Chapter 6 shows the implementation of the methods in Chapter 5 with IBM's Qiskit and evaluates the results.

2 Background

At its core, quantum computing manipulates quantum systems. A quantum system has a state that is represented by a complex vector space and a quantum algorithm is composed of linear transformations acting on that vector space. There are several ways a quantum computer can be designed on the hardware side, but the basic axioms of quantum mechanics do not vary across the different platforms. The following section gives a brief introduction to the basic concepts and terminology of quantum computing.[1]

We start by explaining what qubits and multi-qubit systems are, how they are manipulated through quantum gates and how a quantum circuit is composed. Further, an introduction to a basic quantum algorithm is given with an in-depth explanation of Grover's algorithm, followed by the basic concept of quantum arithmetic operations. The chapter finishes with an overview of quantum computational complexity.

2.1 Qubit

In classical computation and information theory, the bit represents the smallest, indivisible unit of information. The quantum information counterpart is called 'quantum bit' or qubit. The classical bit has two possible states, $\{0, 1\}$, while the simplest possible states in a quantum system is described through the basis vectors $\{|0\rangle, |1\rangle\} \in \mathbb{C}^2$. So the state of a qubit $|\psi\rangle$ is a vector in a complex Hilbert space $\mathcal{H} = \mathbb{C}^2$ described by a linear combination of its basis vectors $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \tag{2.1}$$

with $\{\alpha, \beta\} \in \mathbb{C}$ and normalized as

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.2}$$

In quantum mechanics, the Bra-Ket or Dirac notation is often used to describe a quantum state. While the Ket notation denotes a column vector so that $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$, the Bra notation is its entry-wise complex conjugated transpose and therefore a row vector. The linear combination in Equation 2.1 is also called a superposition and reflects an arbitrary direction in which the state vector points within the Hilbert space. To make this idea more accessible, we can create a three dimensional geometric representation of such a state ϕ by rewriting Equation 2.1 as

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right). \tag{2.3}$$

[1]See [NC02] for a detailed introduction to quantum computing and quantum information theory

$e^{i\gamma}$ is called the global phase, but since it has no observable effects, we can simply write:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\,. \tag{2.4}$$

The real numbers θ and φ define a point in the so called Bloch sphere, which is illustrated in figure 2.1.

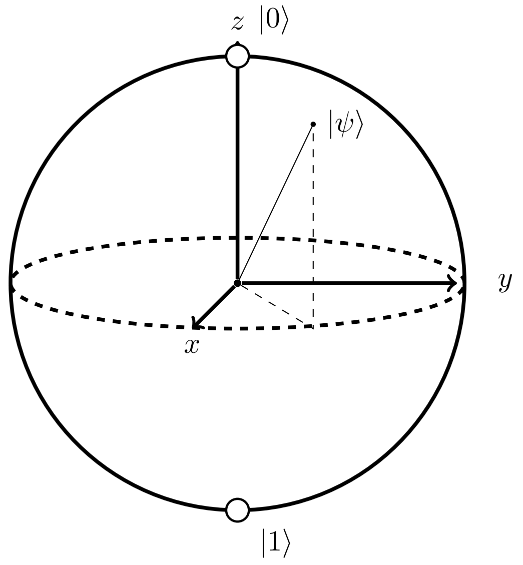


Figure 2.1: Bloch Sphere representation of a qubit state $|\psi\rangle$.

One conclusion could be that a quantum state can store an infinite amount of information since there are infinite points on the Bloch sphere. This is true for its quantum representation, but for us to get this information we need to perform a measurement. This measurement collapses the state $|\psi\rangle$ and we receive either one of the computational basis states $|0\rangle$ and $|1\rangle$ as classical information. We obtain this information with a probability of $|\alpha|^2$ for 0 and $|\beta|^2$ for 1. So the actual amplitudes α and β, which have all the information about the quantum state, can not be read directly and the quantum information can not be translated perfectly into classical information. This is true for all values for α and β except 0.

2.2 Multi Qubits

Given a two qubit system, the four computational basis states are $|00\rangle, |01\rangle, |10\rangle, |01\rangle, |11\rangle$. The joint state of a system of qubits is described by the tensor product $\otimes$, while $|0\rangle \otimes |0\rangle$ is often shortened to $|00\rangle$. So by taking the definition stated in Equation 2.1, the state vector describing the superpositions of theses four states is

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle\,. \tag{2.5}$$

The complex coefficients α_i are the state's amplitudes and squaring them gives the probability the state collapses in one of the corresponding classical states 00, 01, 10, or 11. As in Equation 2.2, the normalization condition requires that the sum of all the probabilities of all possible states is 1. One can say that the mathematical structure of a single qubit generalizes to any higher-dimensional quantum system. It is important to note that the dimension of the state space 2^n grows exponentially in the number of qubits n. So one can say that any quantum state can be written as a linear combination of its basis states. However, there are multi-qubit states, that cannot be written as the tensor product of their single qubit subsystems. For two-qubit states, one of the most famous examples are the Bell states or EPR[2] pairs, of which

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{2.6}$$

is one of them. It demonstrates a part of quantum mechanics that does not exist in the classical world. Both qubits are entangled. So measuring one qubit determines the state of the other. In the Bell state $|\Phi^+\rangle$ this is either $|00\rangle$ or $|11\rangle$, with equal probability of $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$. Entanglement is the essential part of quantum computing, since it creates a 2^n dimensional complex vector space out of n qubits, also called superposition, to perform our computations in.

2.3 Quantum Gates

So far, we showed how the quantum state of a qubit is defined. But in order to perform quantum computation, we need to be able to manipulate quantum information and therefore the state of a qubit. This is done through so-called quantum logic gates, whose functionality is analogous to classical gates. As in classical computation, a quantum circuit is composed of quantum wires, that carry the information and given quantum logic gates, that manipulate a certain input to get the desired output.

2.3.1 Single Qubit Gates

In the following section, a summary of the most important quantum gates is provided. Quantum gates acting on a single qubit state can always be described by 2x2 matrices. Following our normalization condition in Equation 2.2, which must be true before and after the application of a quantum gate, those matrices must be unitary. So for all quantum logic gates U, $U^\dagger U = I$, where $U^\dagger$ is the adjoint of U. Generally speaking, every unitary 2x2 matrix can be a quantum gate on a single qubit state. And since the state of a qubit is represented as a vector in a complex Hilbert space, a quantum logic gate can be seen as a rotation of this vector. Let's start with the simple NOT gate. As expected, the quantum NOT gate, called X-gate in quantum computation, acts on the computational basis state in the following way:

$$X|0\rangle = |1\rangle$$
$$X|1\rangle = |0\rangle$$

Further it acts linearly on the general superposition state as seen in (2.1):

$$X(\alpha|0\rangle + \beta|1\rangle) = \alpha|1\rangle + \beta|0\rangle \tag{2.7}$$

[2]Named after a paradox described in a paper published by Einstein, Podolsky and Rosen (EPR) in 1935 [EPR35]

Since the quantum state $\alpha |0\rangle + \beta |1\rangle$ can be written as

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \tag{2.8}$$

the NOT or X gate is represented by

$$X := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{2.9}$$

so that,

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}. \tag{2.10}$$

Within the bloch sphere (Figure 1) the X-gate is a rotation about the x axis by 180 degrees. So are the Y and Z gates,

$$Y := \begin{bmatrix} 0 - i \\ i \quad i \end{bmatrix} \tag{2.11}$$

$$Z := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \tag{2.12}$$

rotations around their respective axes. In general a quantum gate can be written as a rotation matrix,

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \tag{2.13}$$

since the rotation matrix itself is unitary Another essential quantum gate that clearly involves quantum effects is the Hadamard Gate, or H-gate,

$$H := \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.14}$$

Applied on the computational basis states,

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.15}$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \tag{2.16}$$

it brings them in an equal superposition, so that after a measurement the outcome has, just like a perfect coin toss, a uniform probability distribution between 0 and 1. This is true for both superpositions, often denoted as $|+\rangle$ and $|-\rangle$ respectively, with the only difference being the negative phase for $|-\rangle$. Loosely speaking, the Hadamard gate allows us to expand the computational state space we can access and is therefore one of the quantum logic gates that drive our quantum computational power.

2.3.2 Multi Qubit Gates

The single-qubit gates give us the fundamentals for universal quantum computation. But in order to use the full potential of quantum computation, we need a way for qubits to interact with each other. This involves quantum gates over two or more qubits. In Equation 2.5 we saw how a multi-qubit state vector is written and with the bell state, we already saw an example of how an application of a multi-qubit gate can look like. If we stick to that example and introduce the CNOT gate, which is one of the two components that leads to the Bell state. The CNOT gate,

$$CNOT = CX = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \end{bmatrix} \tag{2.17}$$

is directly inspired by its classical counterpart, the XOR gate, and maps

$$\begin{aligned} |00\rangle &\rightarrow |00\rangle \\ |01\rangle &\rightarrow |01\rangle \\ |10\rangle &\rightarrow |11\rangle \\ |11\rangle &\rightarrow |10\rangle \end{aligned} \tag{2.18}$$

So it sets the first qubit as control and if and only if one, applies an X-gate on the second qubit. We can rewrite this as

$$|x, y\rangle \rightarrow |x, y \otimes x\rangle\,. \tag{2.19}$$

Here $\otimes$ acts as addition modulo 2 and the comma are simply inserted to provide a better read. This notation is commonly used and will return later, when we discuss Grover's Algorithm.

This can also be extended to two (or more) control qubits, which then is a three-qubit gate known as Toffoli gate or CCNOT. The Toffoli gate applies an X gate to the target qubit if and only if both control qubits are 1. It, therefore, acts as an AND or NAND gate on the target qubit, depending on whether the target qubit was in $|0\rangle$ or $|1\rangle$ before. To introduce yet another common notation and because its matrix representation getting quite confusing, the Toffoli gate can be described through

$$CCNOT = |11\rangle\langle 11| \otimes X + (I - |11\rangle\langle 11|) \otimes I. \tag{2.20}$$

As the ket notation represents a column vector and the bra notation its complex conjugated transposed, $|\psi\rangle\langle\psi|$ is the outer product of state ψ.[3] I is the commonly known 2x2 identity matrix. In fact, the Toffoli gate is universal for classical computation, therefore can construct any boolean function, but not for quantum computation as shown in [Tof80]. Further, we will also use multiple controlled quantum gates, which are usually constructed by multiple single and two-qubit quantum gates. [4]

[3]The Bra-Ket notation also implies that $\langle\psi|\psi\rangle$ is the inner product, which is by definition one.

[4]See `https://www.scottaaronson.com/qclec/16.pdf` for a closer description of gate universality.

2.3.3 Quantum Circuits

With the essential building blocks of quantum computation, we can construct the Bell state in a quantum circuit:

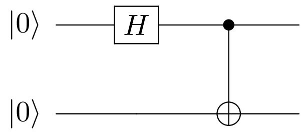


Figure 2.2: Quantum circuit for preparing a Bell state

Each horizontal line represents a qubit, with its initial state notated on the far left side. Here both qubits are initialized in the computational basis state $|0\rangle$. The circuit itself is read sequentially from left to right, with gates acting on the respective qubit. While the Hadamard gate is self explanatory, the CNOT gate is written as

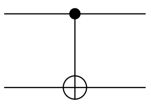


Figure 2.3: CNOT Gate

where the upper dot is the control qubit and the lower $\otimes$ denotes the target qubit, on which the X-gate is applied if and only if the control qubit is 1. Therefor the above circuit encodes,

$$\begin{aligned} CNOT(H \otimes I)\,|00\rangle &= CNOT\,((H\,|0\rangle) \otimes (I\,|0\rangle)) \\ &= CNOT\left(\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \otimes |0\rangle\right) \\ &= CNOT\left(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)\right) \\ &= \frac{1}{\sqrt{2}}(CNOT\,|00\rangle + CNOT\,|10\rangle) \\ &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \end{aligned} \tag{2.21}$$

To obtain the results as classical information, a measurement operator is applied on every qubit, usually denoted by a meter symbol:

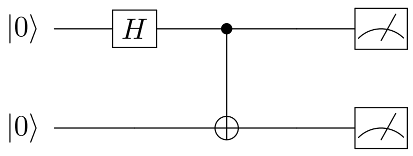


Figure 2.4: Quantum circuit for preparing a Bell state with measurement

As a result we receive the maximal entangled two-qubit state and as stated above, the qubits will always be in the same state after measurement. The qubit itself is measured in the computational basis.

2.3.4 Quantum Algorithms

Given the quantum logic gates and the quantum circuit, we can construct any quantum algorithm following these steps:

Algorithm 1 Construct Quantum Algorithm

```
Input: Input register of $|\psi\rangle^{\otimes n}$
1: Encode data into the state of the input register qubits
2: Apply a sequence of quantum gates on the set of input qubits
3: Obtain classical information by measuring one or more qubits at the end or any other
   point in time
```

The central algorithm we will use throughout this thesis is Grover's quantum search algorithm. The next section is dedicated to a deep introduction.

2.4 Grover's Algorithm

Grover's Algorithm [Gro96] was discovered shortly after Shor's algorithm [Sho99] in 1995. Even though it has a quadratic speed up, rather than an exponential one compared to Shor's algorithm, it is applicable to a wider range of problems. One application is the speedup of problems for which a polynomial-time algorithm exists. Its original task was to find an element in an unordered database. A classical computer algorithm would need a linear number of queries, $O(n)$, to find the element. That is because the rules for the O - Notation requires looking at the worst-case scenario, where the desired object is located at the n-th index. But even if one would take the more realistic approach to look at the average number of queries, it would still take a linear in n number of queries $\sim \frac{n}{2}$. Grover's algorithm on the other hand requires a maximum of $O(\sqrt{n})$ 'quantum' queries. In its standard version it achieves this by using a relatively low number of qubits, $O(log(N))$, and gates, $O(\sqrt{n}\log(n))$. To understand the functionality of Grovers search algorithm, it is important to understand that the scenario of searching for an element in an unstructured database might be intuitive, but not quite right. In fact, Grover's algorithm does not search through a list of elements, it searches through a list of possible inputs x for the function f that returns true or false. So the function $f(x)$ is defined as

$$f(x) = \begin{cases} 1 & \text{if } x = x^* \\ 0 & \text{if } x \neq x \end{cases} \tag{2.22}$$

where x are bit-strings and x^* are the solution bit-strings we are looking for. Given $f(x) = 1$, the circuit will flip an ancilla qubit that is prepared in $|-\rangle$ and therefore 'mark' the correct solutions by flipping their amplitude.[5] This routine is called an oracle. It is important to

[5]Ancilla qubits are additional qubits in a quantum circuit that are either necessary for a certain algorithm or are implemented to reduce the complexity and depth of a circuit. Initialized to $|0\rangle$ they are usually brought back to that state through either reversed application of the precious gates or through a reset, depending on the requirements of the circuit. They do not affect the output directly and are often reused.

note that the oracle does not need to know the exact solution, but it must recognize it. The oracle is also often called a black box since its internal construction is very specific to the problem one needs to solve and is not required in order to understand the concept of the overall algorithm[6]. For now, it is enough to assume that it is constructed efficiently, it recognizes the correct solutions, and will flip their amplitude. So given the function from above, an oracle O, which in turn is a unitary operator, has $|x\rangle$ as the input register and $|q\rangle$ as the ancilla or oracle qubit:

$$|x\rangle\,|q\rangle \xrightarrow{O} |x\rangle\,|q \otimes f(x)\rangle\,. \tag{2.23}$$

As mentioned, the oracle qubit $|q\rangle$ is prepared in the state $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. So the application of the oracle can be written as:

$$|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \xrightarrow{O} (-1)^{f(x)}\,|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right). \tag{2.24}$$

Meaning, if the oracle is applied to a non-solution state, it does not change the state. But if applied to a correct solution state, it will shift its phase.

2.4.1 Procedure

Grover's search algorithms consists of three main parts:

- State Preparation
- Oracle
- Grover Diffusion

The oracle and the Grover diffusion form the Grover operator or iteration (the actual denotation varies across the literature). The overall algorithm is:

Algorithm 2 Grover's Algorithm

Input: (1) Input register of $n+1$ qubits; (2) Oracle O that recognizes a solution state and flips the oracle qubit as in Equation (2.23)
Output: Searched bit-string x^*

```
1: Apply state preparation on $|0\rangle^{\otimes n}\,|0\rangle \rightarrow \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$
2: Apply the Grover operator $R$ times $\rightarrow ((2\,|\psi\rangle\,\langle\psi| - I)O)^R \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$
3: Measure the first $n$ qubits $\rightarrow x^*$
```

Note that especially the implementation of a complex oracle usually requires additional ancilla qubits. As this section only describes the core functionality of Grover's algorithm, this is not taken into account and the oracle function itself is treated as a black box. We will use the following section to explain the three main components and to give an intuition of how they work together. For a more detailed explanation see [NC02].

[6]This notion can be misleading since, in order to actually solve a problem with Grover's Algorithm, the efficient oracle construction is the hardest work. And eventually, we know exactly what is inside the black box.

State Preparation

An important part when finding the right implementation of Grover's algorithm is an efficient encoding of the problem (or search space) into the input register. We will discuss this more deeply in section 5, since it is specific to a problem or even an instance of it. It will define what the single qubits actually stand for. After an encoding is found, the input state itself is always prepared in the same way.[7] From n input qubits combined in the state $|\psi\rangle$, the overall search space is spanned by putting $|\psi\rangle$ in a uniform superposition. Here for we simply apply a Hadamard on every qubit in $|\psi\rangle$ such that:

$$|\psi\rangle = H^{\otimes n} |0\rangle_n = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle , \tag{2.25}$$

with $N = 2^n$. So far we only prepared the input register encoding our problem space, next we need to prepare grover qubit $|q\rangle$. As mentioned above, $|q\rangle$ need to be in $|-\rangle$. Starting from $|0\rangle$, we achieve this by:

$$|q\rangle = HX |0\rangle = H |1\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \tag{2.26}$$

Given an example with $n = 3$, the resulting circuit is:

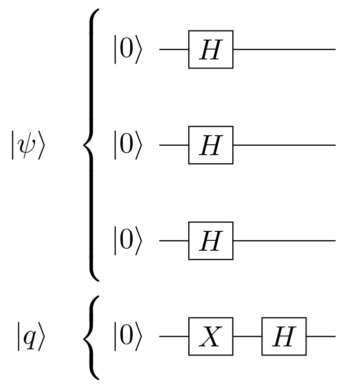


Figure 2.5: Circuit after state preparation

For every search conducted with Grover's algorithm, this step must only be made once. The next two parts are combined known as the Grover operator. They will be repeated several times, depending on the fraction of valid states within the overall search space. At first, let us have a look at the oracle. The gates contained in the oracle will constrain the search-space, while it grows exponentially with the number of qubits. Here lies the power of quantum computation, since already a fairly small circuit of 30 qubits can represent a space of $N = 2^{30} \sim 1 * 10^9$.

[7]Always is not quite right, since there may some instances where we would not need the entire 2^n computational basis states of the input state.

Oracle

As stated above, the oracle has the task to recognize a valid state and flips its amplitude. Given an input state, composed of a valid input string $|x^*\rangle$ and the prepared Grover qubit in state $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$, the oracle O must act in a way such that,

$$O\,|x^*\rangle \frac{|0\rangle-|1\rangle}{\sqrt{2}} \to |x^*\rangle \frac{|f(x^*)\otimes 0\rangle - |f(x^*)\otimes 1\rangle}{\sqrt{2}} = |x^*\rangle \frac{|1\rangle-|0\rangle}{\sqrt{2}} = -\,|x^*\rangle \frac{|0\rangle-|1\rangle}{\sqrt{2}}, \quad (2.27)$$

with f being the function defined in Equation 2.22. As discussed in Equation 2.23, we can see that the oracle only flips the amplitudes of the states x for which $f(x) = 1$ and therefore represents a solution. Applied to our three input qubit example, the oracle acts on the input register $|\psi\rangle$ and the Grover qubit $|q\rangle$, resulting in the circuit:

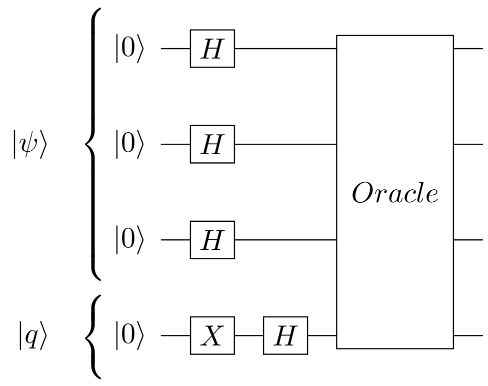


Figure 2.6: Circuit after oracle implementation

Grover Diffusion Operator

While the solution state is now marked by its rotated phase, it is still not measurable, since a rotated phase does not change the squared amplitude and therefor the possibility of a certain states' measurement. Here the core of Grovers Search algorithm comes to play, also known as the 'Grover Diffusion Operator'. In short, it creates the average of all amplitudes and inverts all amplitudes about that mean while negating them. So it effectively amplifies the valid solution state and decreases all other amplitudes. The diffusion operator D is defined as

$$D = H^{\otimes n}(2\,|0\rangle\langle 0| - I)H^{\otimes n} = 2\,|\psi\rangle\langle\psi| - I. \quad (2.28)$$

As in Equation 2.25, $|\psi\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle$ is a uniform superposition over all basis states. So the actual phase flipper is located in a so called Hadamard sandwich. I is the identity matrix of dimension N. We can see the effect of D, when we apply it on a general state with amplitudes a given by $\sum_x a_x\,|x\rangle$ so that,

$$(2\,|\psi\rangle\langle\psi| - I) = \sum_x (2\,\langle a\rangle - a_x)\,|x\rangle\,. \quad (2.29)$$

Here we can see the reflection of the state about the mean, since $\langle a\rangle = \frac{\sum_x a_x}{N}$ represents the average amplitude. The diffusion operator acts only on the input register $|\psi\rangle$.

Now we can compose both oracle and diffusion operator to form the Grover operator G so that,

$$G = (2\,|\psi\rangle\,\langle\psi| - I)O, \tag{2.30}$$

which translates into Figure 2.7.

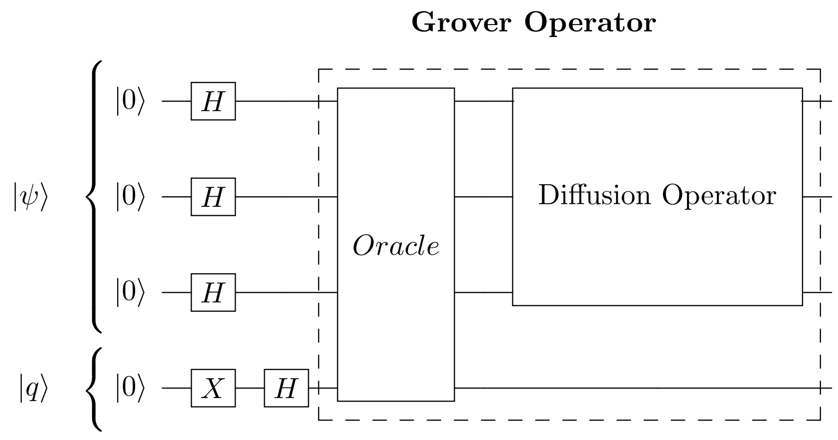


Figure 2.7: Circuit with Grover Operator

At its core, Grover's algorithm marks the solution state by changing its amplitude. To illustrate this method, let us take a look at the amplitudes of all states before the application of the Grover operator in Figure 2.8. Here we see a uniform distribution of amplitudes, all with a height of $\frac{1}{\sqrt{N}} = \frac{1}{\sqrt{8}}$. As before, N is defined as $N = 2^n$ with n being the number of qubits. x^* denotes our solution or searched state, indicated through a blue bar in Figure 2.8. After the application of the oracle, the phase of x^* gets flipped and the resulting amplitudes are given in Figure 2.9. It is only after the application of the diffusion operator that the solution state is distinguishable from the other state, as stated in Equation 2.28 and presented in Figure 2.10.

We can see that after we apply the Grover operator once, which is also called the Grover iteration, we managed to increase the amplitude of the solution state to about $\frac{3}{\sqrt{8}}$, while decreasing the amplitude of all other states. This already gives us a higher probability of the searched state, but to be certain, we need to repeat the Grover iteration until the probability is maximized. For a single searched element, this requires $\sqrt{N}$ iterations or queries, which explains the quadratic speedup compared to N queries in the classical sense.

If one searches for multiple items, Grover's algorithm peaks at R $= \frac{\pi}{4}\sqrt{\frac{N}{K}}$ iterations, where K is the number of marked elements. This behavior is explained through the sinusoidal curve the success probability follows. Another case is where we do not know the number of marked elements K? This typically leads to the soufflé problem [Bra97], where we either under- or overcook our solution. Meaning, caused by the sinusoidal success probability curve, we have mostly unmarked states with too little iterations, but also mostly unmarked states if we overshoot the optimal number of iterations. Either way, the soufflé is cursed. A basic solution is to run the algorithm a random number of time between 0 and $\sqrt{N}$ so that most of the time we get around the middle of the sinusoid. That won't lead to perfect, but constant probability.

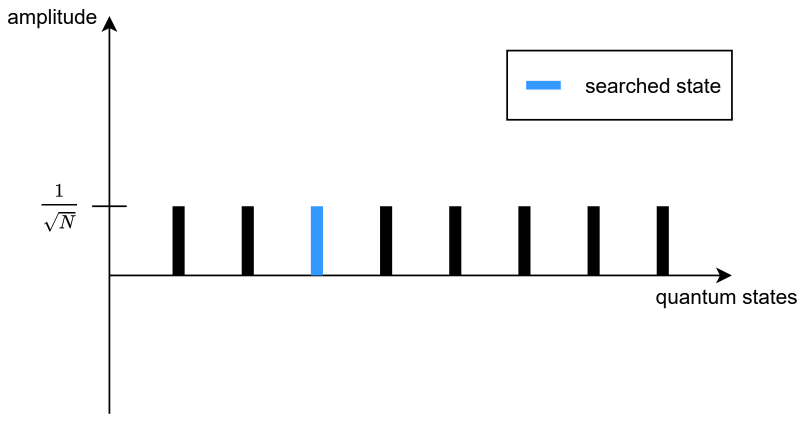


Figure 2.8: Amplitudes of states before Grover operator

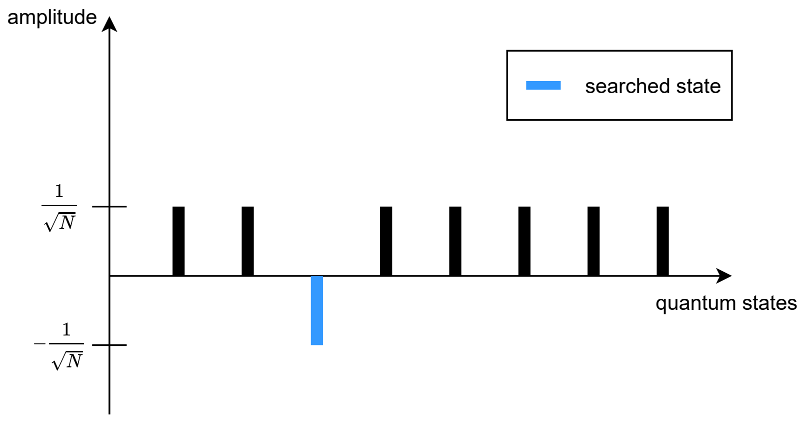


Figure 2.9: Amplitudes of states after oracle application

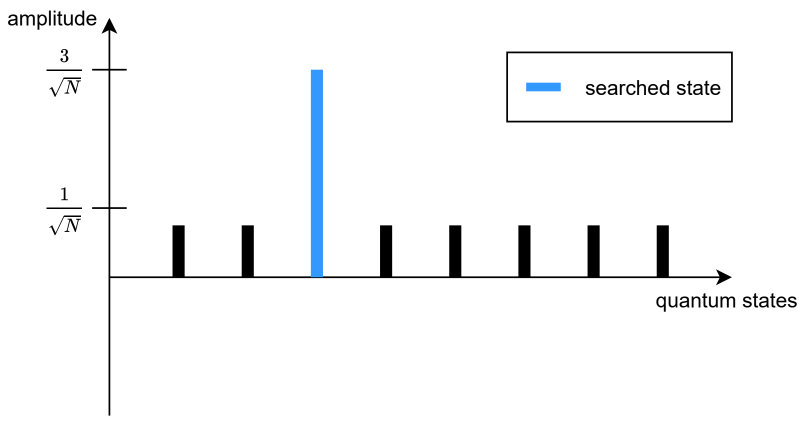


Figure 2.10: Amplitudes of states after Grover operator

2.4.2 Quantum Arithmetic Operations

As in classical computing, arithmetic operators also play an important role in quantum computing. While it is rather complex to implement a reversible full-adder, the general quantum incrementer is straight forward. For a three qubit circuit, the incrementer consists of one Toffoli gate, one CNOT gate and one X gate so that:

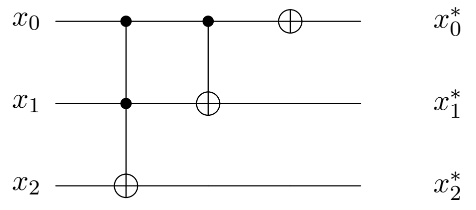


Figure 2.11: Three qubit quantum incrementer

In logical expressions, the incrementer acts on the three input qubits in a way such that,

$$\begin{aligned} x_2 = &\quad x_2 \quad \text{XOR} \quad x_1 \quad \text{AND} \quad x_0 \\ x_1 = &\quad x_1 \quad \text{XOR} \quad x_0 \\ x_0 = &\quad \text{NOT} \quad x_0, \end{aligned} \tag{2.31}$$

which results in the following truth table:

Input	$\vert 000\rangle$	$\vert 001\rangle$	$\vert 010\rangle$	$\vert 011\rangle$	$\vert 100\rangle$	$\vert 101\rangle$	$\vert 110\rangle$	$\vert 111\rangle$
Output	$\vert 001\rangle$	$\vert 010\rangle$	$\vert 011\rangle$	$\vert 100\rangle$	$\vert 101\rangle$	$\vert 110\rangle$	$\vert 111\rangle$	$\vert 000\rangle$

So the function of the incrementer is to add one to the binary representation of the current state of the system. Right now, there is a lot of research being done in the area of making quantum arithmetic more efficient. Especially for a fixed size counter register, some advances are being made in the recent years. In general, one can say that as for many other quantum algorithms, there is always a trade of between time and space. So with a growing number of available qubits, operations tend to be way faster. For a further read see [LYTJ+14] and [Gid20].

2.4.3 Quantum Computational Complexity

But how can we be sure that quantum algorithms are actually capable of providing any speedup when there are no large-scale quantum devices yet available? The answer lies in theoretical computer science, which also started studying formal computation long before the first large-scale computer was build. It gives us an idea on what speedup we can expect and what resources are required to solve a problem. In theoretical computer science, computational complexity theory classifies the time and space resources a classical computer needs to solve a certain problem. Even if the exact algorithm is not known yet, it proves lower bounds on those resources required to solve the problem.

The problems itself are part of complexity classes such as:

• **P**	Problems that are solvable in polynomial time.
• **NP**	Problems that are verified through a deterministic polynomial-time algorithm.
• **NP-hard**	Problems to which every **NP** problem can be reduced to in polynomial time.
• **NP-complete**	Both in **NP** and **NP-hard**.

Famously, **P** = **NP** is yet to be proven. Those four classes are just the most common ones and theoretical computer scientists are constantly defining more.[8] In 1993, [BV97] introduced the first complexity class explicitly defined for quantum computational complexity, Bounded-Error Quantum Polynomial Time or **BQP**. Following [BV97], a decision problem in **BQP** can be solved by a quantum algorithm in 2/3 of the cases in polynomial time. Therefore it represents the quantum counterpart to **P** or to be more precise **BPP** (bounded-error probabilistic polynomial time) which uses a probabilistic Turing machine instead of a deterministic one. And since we said before that the Toffoli gate can simulate the classical universal gate, we can say that $\mathbf{P} \subseteq \mathbf{BQP}$. Further looking at Shor's quantum algorithm, which does factoring in polynomial time, we can even say that there are problems outside **P**, which are included in **BQP**. We can also define a not finale upper bound for **BQP**, saying $\mathbf{BQP} \in \mathbf{PSPACE}$. [BV97] showed this by proving that it is possible to simulate a quantum computer classically with exponential time and polynomial memory. Hence, quantum computers are at least as fast as classical computers and at most exponentially faster. Figure 2.12 shows some of the complexity classes graphically.

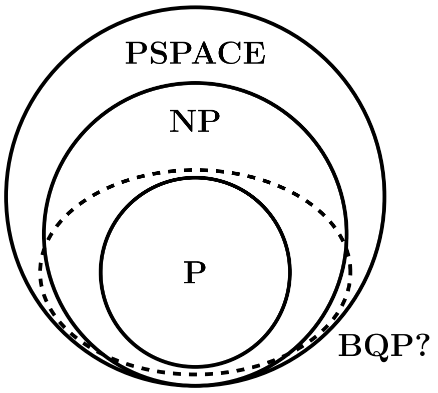


Figure 2.12: Complexity Classes

We can see that,

$$\mathbf{P} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE}.$$

This leads to a problem. As long as we can not proof that $\mathbf{P} \neq \mathbf{PSPACE}$, we also don't know if $\mathbf{P} \neq \mathbf{BQP}$. However, even if we are not able to solve **NP**-complete problems in polynomial time, we could still achieve a quantum speedup. An example is the Adiabatic

[8] `https://complexityzoo.uwaterloo.ca/Complexity_Zoo` database for all sorts of complexity classes.

Algorithm by [FGGS00]. It achieves a strong (potentially up-to exponential) speedup for NP-complete problems by exploiting their structure. The problem is that its performance varies over different structures. This makes it still very valuable for real use cases but does not satisfy theoretical computer scientists. But the same is true for non-quantum algorithms. For NP-complete problems, like the CircuitSAT, it is obvious that the speedup provided by Grover is a real advantage since there is no known classical algorithm that is faster than brute force. But for example, 3SAT, a more structured NP-complete problem, is solvable in $O(1.3^n)$. Here the focus would be more on combining existing algorithms with Grover's search algorithm, than applying it alone. An overview of such algorithms will be given in section 4. In terms of optimization problems, like the scheduling problem we try to solve, even a relatively small advantage in computational speed might already lead to significant benefits.

2.4.4 Scheduling Problems

The root of scheduling problems goes back as far as 1954, when [Edi54] and [Dan54] tackled the problem of traffic delays caused by badly assigned toll booths operators. They introduced the first scheduling problem, which, constraint by minimizing the cost of personnel, aimed to reduce the average delay for each car. Since then, a variety of such scheduling problems has been identified and researched, such as personnel scheduling problems. And with increasing computational power and algorithmic development over the years, its popularity is growing. This increase could be motivated by the economic factor of decreasing labor costs, which is a major fixed cost for many companies. A good overview of this scheduling problem subset is given in [BBDB$^+$13]. Our scheduling problem is closely related to the known nurse scheduling/rostering problem, in turn, an especially complex version of the personnel scheduling problem. Nurse rostering is known to be NP-hard and intends to optimize a schedule consisting of multiple shifts and nurses while considering a variety of soft and hard constraints.

So far, we have discussed the principles of quantum computation, Grover's algorithm, and quantum computational complexity. This will give us the necessary background knowledge to proceed with the rest of the thesis. Before we explain the actual methods and their implementation, we will describe the actual problem and perform a requirement analysis in the next chapter.

3 Problem Description and Requirement Analysis

In the following chapter, we describe the on-call operator scheduling problem, perform a requirement analysis and conclude with the problem formulation.

3.1 OnCall Operator Scheduling

The German Aerospace Center (DLR) operates a variety of spacecraft missions from its mission control center, which requires the constant presence of one or more operators per subsystem for each mission. Those operators are organized through an on-call spacecraft operator scheduling which is created multiple times per year. At the moment, this schedule is either created manually or through a classical solver tool that uses a brute-force approach to find an optimal assignment. Due to the nature of combinatorial optimization problems, the complexity grows exponentially with their size. So, beginning from fairly small problem instances, this leads to infeasible computation times. To by-pass this problem, schedulers either rely in their own intuition or use heuristics, both are very unlikely to find the optimal solution. Thus, the DLR decided to explore the application of quantum computation in order to potentially find an optimal solution in feasible time.

3.2 Requirement Analysis

To be applicable to the actual use case, such a software or algorithm must fulfill functional and non-functional requirements. The functional requirements will describe what the solution must do, while the non-functional requirements indicate how the solution must solve the problem.

3.2.1 Functional Requirements

The schedule must allocate 52 operators on 17 positions over a period of 180 days and be updated if certain positions can not be filled as planned. An on-call shift has a duration of 24 hours, so per day, which we use as our time interval, there is at most one operator assigned to each position. Further, every operator must have a set of subsystems he is able to operate depending on their background. To be valid, the computed schedule must also fulfill the following hard constraints:

- Per time-position only one operator can be assigned
- Out of three time units, an operator is only allowed to work two
- An operator can be assigned to at most one position a day

The general aim is to find an optimal assignment of the operators to their respective positions while not violating those constraints. Since the core idea is to solve the problem through quantum computation, IBM's quantum systems and its developer tool Qiskit must be used to implement the quantum algorithms. Those algorithms must contain:

- Method to define the underlying quantum circuit
- Methods that implement the above mentioned constraints within the Grover oracle
- Method that implements the actual Grover circuit into the quantum circuit
- Method that runs the overall circuit and interprets the results

All the methods must be written within a Jupyter notebook, which builds the overall scheduling tool. The tool must be able to take an existing schedule or scheduling problem in form of a JSON/YAML data format as an input and return a data format which is readable by the generic planning tool PINTA.

3.2.2 Non-functional Requirements

In addition, there are four non-functional requirements the tool must meet:

Scalability

The tool should be scalable. Since it is expected that the current quantum hardware limits the solvable problem size, all methods should be written in a way that they can dynamically adapt to future, more powerful quantum hardware. So while as of today a circuit is limited to roughly 30 qubits, it should also run on devices with more, while fully using their computational power.

Performance

Even though it is not required for the present hardware, the performance should surpass the benchmark given by classical solutions.

Expandability

The tool must be expandable, so that future changes to the constraints can be implemented. It must also be possible to add and subtract constraints without limiting the functionality of the tool.

Maintainability

In order to ensure a comfortable usage, the tool must be maintainable. Methods should be defined in a clear manner and reused to avert duplications.

In this section, we defined the on-call scheduling problem and the requirements to a solution given by the DLR. The next chapter discusses the related work.

4 Related Work

The following section discusses research conducted in the area of solving optimization problems, both in classical and quantum computing. After providing an overview of state-of-the-art classical approaches, which a quantum solution will eventually face as a benchmark to break, the quantum section starts with a historical overview of important quantum algorithms. Further, recent research in the area of enhancing and applying Grover's search algorithm will be presented, followed by a short survey of alternative quantum optimization methods. For further information on basic quantum algorithm, [YM08] and [NC02] is recommended.

4.1 Classical Algorithms

As discussed earlier, our operator scheduling problem maps perfectly on the well-studied nurse scheduling/roastering problem. As a reference, it helps to have a look at classical solutions to solve this problem. They mainly fall into three categories: mathematical exact, metaheuristic and hybrid approaches.

Mathematical exact solutions

Exact solutions are given by [ABHW73] and [JSV98], which approach the problem through linear programming. Even though they find optimal solutions, their lack of constraints makes their approach not feasible for real-world applications. [GDT09] on the other hand proposed a model working with GRASP and Knapsack, that actually provides a significant improvement over existing solutions.

Metaheuristics

Metaheuristic approaches produce not optimal, but reasonably good solution within a limited running time. Popular approaches uses either simulated annealing [AZD13] [CLPR10], tabu search [BW06] [BBK$^+$10] or genetic algorithms [ABL08] [ABL06].

Hybrid approaches

Another interesting solution is presented by [QH08]. They use hybrid constraint programming while decomposing the nurse rostering problem into weekly sub-problems which in turn model a constraint satisfaction problem. Followed by a forward search to generate a complete solution, they use a variable neighborhood search to further improve the solutions. Studying the current landscape of classical solutions, hybrid solutions tend to be the best performing algorithms for scheduling problems.

4.2 Quantum Algorithms

The following section will show research conducted in the field of quantum algorithm related to Grover's search algorithm.

Amplitude Amplification

Originally, Grover's Algorithm was designed to find a single item in an unstructured search space. [BHMT02] further developed the algorithm by generalizing its core idea of amplitude amplification. Their algorithm requires no knowledge of the exact solution and can also search for multiple solutions. Further, they introduce amplitude amplification, which uses Shor's phase estimation to estimate the success probability of a quantum algorithm. Since there are some polynomial-time heuristic search algorithms, they show that the combination of classical heuristic and amplitude amplification would still lead to a quadratic speedup in the estimated time a solution is found.

Fix-point quantum search

Grover's algorithm and the generalized amplitude amplification both are hard to use if the fraction of valid states within the overall search space is unknown. A solution is the so-called fix-point search as proposed in [Gro05] which only needs a lower bound on this fraction and always amplifies marked states. Through running the algorithm long enough, it improves its success probability asymptotically. But the price is high, because the initial quadratic speedup is lost. However, [YLC14] recently presented a fixed-point search that achieves both through, as a way to brief summary, adjusting the phases of Grover's reflection operator. It can be used as a subroutine for every amplitude amplification application and eliminates the need to run the algorithm multiple times as suggested in section 2.

Grover Adaptive Search

A promising work regarding its application on combinatorial optimization problems is done by [GWG19] with the use of Grover Adaptive Search. Grover Adaptive Search is based on the work of [DH96], which uses amplitude amplification from [Bra97] to solve the minimum searching problem with Grover's quadratic speedup. [BBW03] then introduced a way to implement pure adaptive search with the generalized version of Grover's Search Algorithm and coined the method Grover Adaptive Search. It searches for the optimum value of a function by iteratively applying Grover's Search Algorithm while defining thresholds and using them to further optimize the solution. In other words, it samples randomly from all the better solutions and subsequently acts similar to classical sequential approximation methods. [GWG19] uses this method and in its core provides a framework for an efficient automated oracle construction. This framework is especially efficient for constraint polynomial binary optimization and especially for quadratic unconstraint binary optimization. Both are common to model combinatorial optimization problems. This method was also implemented in Qiskit and explored as a potential method to solve our scheduling problem. However, the required ancilla qubit overhead is too big for current simulators, so while it still remains an efficient method further down the road, it will not be further considered in this thesis.

4.3 Quantum Algorithms to Solve Optimization Problems

Due to their complexity, combinatorial optimization problems are also the focus of various other quantum algorithms. A broad overview is given in [ZZ17] and a more focused one on NISQ era devices is given in [SBC$^+$20].
In the following section, I focus on the three most popular ones. All of them use a Hamiltonian matrix H to represent the respective optimization problem. A Hamiltonian is usually used to describe the energy of a system and if it encodes an optimization problem, its lowest energy state is the optimal solution of the problem.

Variational Quantum Eigensolver

The Variational Quantum Eigensolver (VQE) was introduced by [P$^+$13] to find the eigenvectors and eigenvalues of the corresponding Hamiltonian. It is declared as a hybrid classical/quantum that uses a parameterized circuit in a fixed form, in which parameters are constantly updated with intermediate solutions, to find the smallest eigenvalue. This is in turn the ground state of the Hamiltonian and therefore the solution of the optimization problem.

Quantum Approximate Optimization Algorithm

Another algorithm to solve combinatorial optimization problems is the Quantum Approximate Optimization Algorithm (QAOA) proposed by [FGG14]. Similar to the VQE it creates a variational circuit and optimizes its parameters starting from a cost and mixer Hamiltonian. In the end, it samples from the circuit to receive an approximate ground state and therefore solution to the optimization problem. Even though it is specifically designed to solve combinatorial optimization problems, it does not have an equal speed up over all of them. However, as suggested in [FH16] it might be a strong candidate to achieve quantum supremacy on NISQ era devices.

Quantum Annealing

Quantum Annealing is another algorithm focused on solving an optimization problem by finding the ground state of a Hamiltonian. Introduced by [KN98] it displays the quantum alternative to simulated annealing and is therefore a metaheuristic method to solve combinatorial optimization problems. It evolves an initial Hamiltonian to its final form which is its ground state and therefore the solution. When the dynamics are strictly adiabatic and the Hamiltonian complex enough, it is equal to adiabatic quantum computing as shown in [AVDK$^+$08]. In contrast to universal gate quantum computers, quantum annealer, devices that are specifically designed to perform quantum annealing, are much simpler to build and therefore obtain more qubits than state-of-the-art universal quantum computers. This makes quantum annealing appealing to early adopters since the size of the solvable problems is sufficiently large. Two examples are its application on the nurse scheduling problem in [INH19] and the personnel scheduling problem in [exa20], both closely linked to our scheduling problem. However, it still remains an open question if quantum annealer actually provides a computational speed up.

We began this section by giving an overview of different classical approaches in order to solve the scheduling problem. Next, we presented research that extended Grover's algorithm to either make it applicable to more problems or boost its performance. We concluded with a general overview of other quantum algorithms that were specially developed to solve optimization problems. Now, we can go on to describe the methods we use to solve our stated problem.

5 Methods

The following section will describe the core methods we developed to map and solve the on-call scheduling problem on a quantum computer. First, we will explain our encoding process followed by the automated oracle construction to implement the constraints.

5.1 Encoding

Especially in the NISQ era, qubit encoding is a low hanging fruit in order to decrease the number of qubits needed to represent the data. As mentioned before, Grover's search algorithm usually takes n qubits as input and applies Hadamard gates, so that

$$|\psi\rangle = H^{\otimes n} |0\rangle_n = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \tag{5.1}$$

represents the input registers, with $N = 2^n$. The number of qubits n is determined by the number of binary variables $X \in \{0, 1\}$. With a naive implementation for our optimization problem, X would be defined as $X_{d,p,o}$. Every binary variable X would correspond to an operator o assigned to position p on day d. So when initializing a quantum circuit, every qubit $|0\rangle_{d,p,o}$ corresponds to a binary variable $X_{d,p,o}$. Applying a Hadamard gate leads to

$$\begin{aligned} H|0\rangle_{d,p,o} &= \frac{1}{\sqrt{2}}(|0\rangle_{d,p,o} + |1\rangle_{d,p,o}) \\ \rightarrow X_{d,p,o} &= \begin{cases} 0, \text{operator o is } \textbf{not} \text{ assigned to time-position} \\ 1, \text{operator o is assigned to time-position} \end{cases}, \end{aligned} \tag{5.2}$$

after measuring the qubit in the computational basis. For a time-position with 4 operators, the corresponding circuit can be written as:

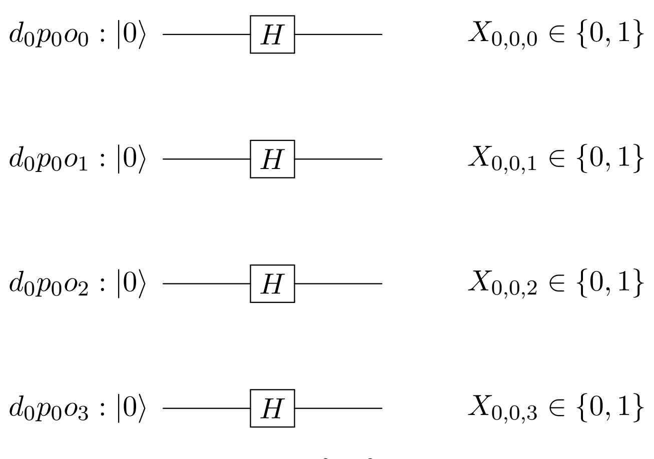


Figure 5.1: Input state for four time-positions

So, a corresponding bit-string, read top-button, can take the form $X = |0010\rangle = [0, 0, 1, 0]^T$, indicating that operator 2 is assigned to position 0 on day 0. But since $|0\rangle^{\otimes 4}$ is in an equal superposition, it can take all 2^4 possible combinatoric values. Thus, X might also have a value like $X = [1, 1, 0, 0]$, which would violate our '1 operator per time-position' constraint. Next to the number of qubits, it would also require additional gates that implement the constraint in the oracle. By providing an alternative form of encoding, we are able to reduce both qubit and gate complexity, also referred to as circuit width and depth respectively. Our method assigns a $\log_2(\#\text{operators})$ number of qubits to each time-position, such that the resulting binary number corresponds to one of the operators. Taking the example from above with 4 operators and 1 time-position, we build the circuit:

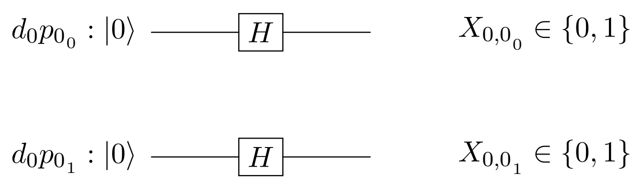


Figure 5.2: Reduced input state

The resulting bit-string will be in one of the $|0\rangle^{\otimes n}$ computational basis state, each of which represents one of the operators:

- $|00\rangle \rightarrow$ Operator 0
- $|01\rangle \rightarrow$ Operator 1
- $|10\rangle \rightarrow$ Operator 2
- $|11\rangle \rightarrow$ Operator 3

In the literature, the computational basis states of a n-qubit systems are sometimes also represented by a decimal number, e.g. $|10\rangle = |2\rangle = [0, 0, 1, 0]^T$. We use this convention to assign each operator a unique state and since the system can only be in one of the basis states, we inherently satisfy our '1 operator per time-position' constraint. Further the width of the circuit is reduced from its naive implementation

$$\mathbf{W}_{naive} := \sum_{d=0}^{D-1} \sum_{p=0}^{P-1} \sum_{o=0}^{O-1} x_{d,p,o} \quad \text{with} \quad d, p \in N \tag{5.3}$$

to the width of binary implementation

$$\mathbf{W}_{binary} := \sum_{d=0}^{D-1} \sum_{p=0}^{P-1} \sum_{r=0}^{R-1} x_{d,p,r} \quad \text{with} \quad d, p \in N \tag{5.4}$$

with $R = \lceil \log(O) \rceil$ and $x_{d,p,o} = x_{d,p,r} = 1, \forall d, p, o, r$ with x being a single qubit.

5.2 Automatic Oracle Generation

The Grover Operator consists of two parts, the oracle and the Grover diffusion operator. While the diffusion operator is fixed for all implementations of Grover's algorithm, the oracle must be tailored to the specific search problem. Its purpose is to flip the amplitude of all states that satisfy the previously defined constraints. But in general, the oracle has to be constructed as a quantum circuit and its efficient construction is crucial for the overall advantage of Grover's algorithm. Hence, we need to define an operator O as a subcircuit, that takes our input state $|x\rangle$ in superposition and flips the amplitudes of all states $|x^*\rangle$ through the function $f(x)$ that satisfies the constraints. So to present a small recap, function $f(x)$ is defined by

$$f(x) = \begin{cases} 1, \text{if} & x = x^* \\ 0, \text{if} & x \neq x^* \end{cases} . \tag{5.5}$$

Given an input state, composed of a valid input string $|x^*\rangle$ and the prepared ancilla in state $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$, the oracle operator must act in a way such that,

$$O\,|x^*\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \rightarrow |x^*\rangle \frac{|f(x^*) \otimes 0\rangle - |f(x^*) \otimes 1\rangle}{\sqrt{2}} = |x^*\rangle \frac{|1\rangle - |0\rangle}{\sqrt{2}} = -\,|x^*\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{5.6}$$

In contrast to the classical example where we search for one state in the overall state space, the on-call spacecraft operator scheduling problem may have multiple states that represent valid schedules. To make sure that we only flip the amplitudes of those valid states, the core circuit-building algorithm is the same for constraints B and C.

Before we can apply the actual automatic oracle construction algorithm, we need to define the algorithm for an incrementer.

Incrementer

As described in Figure 2.11, the incrementer G adds one to the binary value of a counter register or circuit $|\phi\rangle$. Similar to the naive encoding we assume that the two computational basis states of a qubit $|0\rangle$ and $|1\rangle$ represent the binary values 0 and 1 respectively. Writing it as a linear transformation we get,

$$G\,|\phi\rangle = |\phi + 1\rangle \tag{5.7}$$

For our purpose we extend the incrementer so that every gate inside the actual incrementer is applied if and only if all the qubit registers within the constraint subset are one. So our incrementer algorithm is:

Algorithm 3 Incrementer

```
Input: 1) Input register $|\psi\rangle$ of $n$ qubits; (2) counter register $|\phi\rangle$ of $m$ qubits;
Output: If control qubits True, $|\phi + 1\rangle$
1: for $i$ to $m - 1$ do
2:     Multi-Toffoli(Control: $[|\psi\rangle + |\phi_{m-2-n}\rangle]$, Target: $|\phi_{m-1-n}\rangle$)
3: end for
4: NOT($\phi_0$)
```

First, we are defining sets of qubits whose simultaneous presence in a $|1\rangle$ state would violate one of the constraints. How this is realized varies in practice, since different packages for building quantum circuits have different approaches. However, let us first discuss our constraints.

Constraints

In this thesis, we will only use hard constraints and also reduce the overall number of constraints to a minimum. This is mainly caused by the intention to show a theoretical use case for a quantum computer and a significant limitation of the currently available hardware. Nevertheless, our aim is to provide a framework that will be of use with scaling quantum hardware.

In order to map our scheduling problem on a quantum device, we need to formulate it as a constraint binary optimization problem. We introduce a binary variable $X_{d,p,o} \in \{0,1\}$ that is one if and only if operator o holds position p on day d. Further, we define our constraints as follow:

A. For all time-positions there must be at least one operator:

$$\forall p, d : \sum_o X_{d,p,o} \geq 1 \tag{5.8}$$

B. Each operator can work at most two out of three consecutive days:

$$\forall d : \sum_{i=0}^{2} \sum_p X_{d+i,p,n} \leq 2 \tag{5.9}$$

C. For an operator per day there can be at most one position:

$$\forall o, d : \sum_p X_{d,p,o} \leq 1 \tag{5.10}$$

This gives us an intuition on how the constraints are mathematically seen within the constraint optimization problem. Now we need to implement them within our oracle:

Constraint A

As written in the previous section, the encoding process already ensures that constraint A is fulfilled. So no further implementation within the oracle is required.

Constraint B

The first constraints states that on each day d an operator o can only be assigned to one position p. Since our operators are encoded as a binary number in a register for each position on each day, we store the indices of all equal binary numbers within all positions for each day.

Constraint C

The second constraint states that an operator is allowed to work at most two out of three days. For that we store the indices of all equal binary numbers that appear on any position over more than two days.

Now as we defined our constraints and stored the indices of the respective qubits, we apply our incrementer defined in Algorithm 3 iteratively on those sets. It takes the indices of each constraint subset and connects them through logical AND gates, so that the counter register is only incremented if and only if all qubits, referred to by the indices within the constraint subset, are $|1\rangle$. After the incrementer is implemented for every constraint, the counter register is checked for its value. If and only if the value of the counter register is 0, an X gate is applied on the Grover qubit, flipping all phases of the corresponding states according to Equation 5.6. Since the amplitude flipping is only possible through the entanglement of the input qubits and the whole Grover Operator must be applied multiple times, we need to de-entangle the states again after the phase flip. As the gates within incrementer are unitary, we just apply them in reversed order again for every constraint subset.[1]

Hence, we define the following algorithm for the automatic oracle construction:

Algorithm 4 Automatic Oracle Construction

```
Input: (1) Input register $|\psi\rangle$ of $n$ qubits; (2) counter register $|\phi\rangle$ of $m$ qubits; (3) Set $C$ of
    constraints; (4) increment function $g(x)$; (5) Grover qubit $|p\rangle$
Output: Flipped phase for all valid states $|x^*\rangle$
  for elem in $C$ do
  $|\psi\rangle \leftarrow g(|\psi\rangle, elem)$
  end for
  NOT($|\phi\rangle$)
  Multi-Toffoli(Control: $|\phi\rangle$,Target: $|p\rangle$)
  NOT($|\phi\rangle$)
  for elem in $C$ do
  $|\psi\rangle \leftarrow g'(|\psi\rangle, elem)$
  end for
```

We described an algorithm for an automatic oracle construction that applies constraints within Grover's algorithm and how we limit the number of required qubits and gates by using problem specific encoding. Now, we can use IBM's qiskit to implement our methods on a quantum circuit.

[1]One could achieve the same result by negating the controlled part of the incrementer so that it is activated not with all states in $|1\rangle$ but in $|0\rangle$.

6 Implementation

In this section, we will use the previously stated methods and implement them using IBM's Qiskit. Snippets of the actual code and the underlying circuits will be shown and an evaluation of the results will be presented. We will also discuss how different numbers of iterations and sizes of the counter register influence the results. All code is open source and can be accessed on GitHub[1].
Qiskit is an open-source quantum development tool created by IBM [Bib20]. It is written in Python and offers a variety of tools to create and manipulate quantum programs, which then can be either simulated on a local device or run on the IBM Q backend. Even though there are a variety of pre-written quantum algorithms and optimization tools, we mainly use its Terra and Aer package to create quantum circuits at the machine code level and simulate them respectively. The implementation itself follows three steps:

1. Circuit preparation
2. State preparation
3. Grover iteration

6.1 Circuit Preparation

Before we can apply quantum gates, we need to define an initial quantum circuit. In Qiskit, quantum circuits are composed of quantum registers, which in turn are composed of single qubits. This has the advantage, that we can use the registers to clearly define the components of our circuit and also apply gates to registers instead of single qubits.

Encoding

Following Section 5.1, we define every time position as a binary number that we implement as a quantum register. The length of those registers, e.g. the number of required qubits, is given by $\log(operators)$ with $operators$ being the number of operators:

```
# Defines the number of qubits needed to represent the operators as a binary
    number
log2_operators = math.ceil(math.log2(operators))
```

The quantum registers are created and stored in a dictionary. This is not only handy for future access, but also required by Qiskit to generate quantum circuits on the fly. The variable $operators, days, positions$ are defined as the number of their respective entities:

[1]https://github.com/schererant/operator-scheduling

```
# QuantumRegisters with time positions are created in a dictionary
time_positions = {'p%id%i'%(position,day):  QuantumRegister(log2_operators,
    'p%id%i'%(position,day)) for day in range(days) for position in range(
    positions)}
```

Input Register

Since there is one time-position for each position on every day, the overall input register size

```
# Calculate number of input register qubits
var_size = positions * days * log2_operators
```

For our batch size, the overall number of input qubits is 12 separated in 6 quantum registers á two qubits.

Counter Register

To implement the constraints as in Algorithm 4, a counter register is required, which needs to be incremented for every forbidden state. Additional working qubits are often called ancilla qubits and as we reverse our incrementer, the counter register is reusable for every Grover iteration. The size of the counter register is approximately growing with $O(\log(n))$ with n being the input register size.

```
anc = QuantumRegister(counter_size, 'counter')
```

Oracle Qubit

The oracle qubit will be initialized in $|-\rangle$ and is required to flip the amplitude of the valid states. Like the counter register, it is reusable, so its size is fixed to one. Even though it only has one qubit, for consistency we assign it its own register:

```
f = QuantumRegister(1, 'oracle_qubit')
```

Initial Circuit

Now, we compose all the registers from above in one quantum circuit:

```
# Quantum Circuit gets assembled from the time_position dictionary
qc = QuantumCircuit(*[qubit for qubit in time_positions.values()], anc,
    measure, f)
```

To ensure a lucid layout, the circuit is reduced to 2 time-positions and 2 counter qubits.

6.2 State preparation

As mentioned above, two main preparations are required: The oracle qubit register in $|-\rangle$ and the input registers containing the time-position must be brought in a superposition over all input registers. Please note that Qiskit always initializes a qubit in $|0\rangle$.

Oracle Qubit preparation

By applying the X gate we flip to $|1\rangle$ and the Hadamard gate will bring us to $|-\rangle$. With qc being the initial quantum circuit, we apply both gates through:

```
# Output register gets prepared in |->
qc.x(f[0])
qc.h(f[0])
```

Input register preparation

To prepare the input register, we apply a Hadamard gate on every qubit. So with n being the size of the input register, we get:

$$H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n} |x\rangle \tag{6.1}$$

The implementation with Qiskit is fairly easy and follows the stated logic:

```
# Input register initialized with hadamard gates
for i in range(n):
  qc.h(i)
```

Of course it is also possible to apply the Hadamard gate on a whole register, but since our input register consists of multiple smaller registers, it is easier to loop through the length of the input register.

6.3 Grover iteration

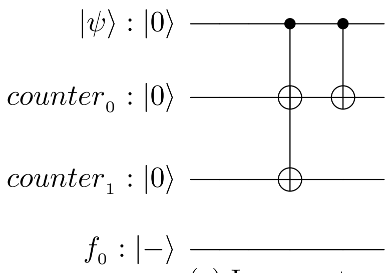
After we prepared the input state, we now implement the Grover operator or iteration, consisting of the oracle and Grover's diffusion operator.

Oracle Construction

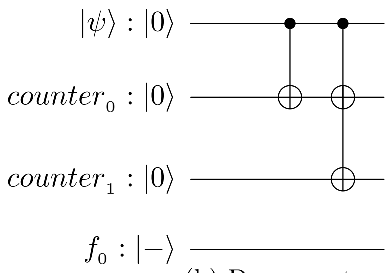
We construct the oracle according to Algorithm 4. The following will present the implementation of both constraints in Figure 2.5. We go into detail for constraint B and just give an outlook for constraint C. This has two reasons: 1. our example circuit has just one day and therefore fulfills constraint C a priori and 2. the underlying structure is similar in both cases. Before we start, we need to define how the incrementer is implemented since it is a crucial part of the oracle. For a more in-depth explanation of the incrementer algorithm see Algorithm 3.

Incrementer

There is a lot of discussion on how to potentially implement an efficient quantum adder. For further literature see [NC02] and [Gid20]. In our case, we do not need a reversible quantum adder per se. We just build an incrementer consisting of n multi-Toffoli gates, with n being the number of counter register qubits. We build two gate groups, one to increment and one to decrement, given in Figure 6.1.



(a) Increment



(b) Decrement

Figure 6.1: Arithmetic operations

These implementations have the advantage that they are easily scaled to bigger counting registers. That can be made clear by looking at the corresponding Qiskit implementation:

```python
def increment(source, target):
    n_f = len(target) - 1
    for n in range(n_f):
        lst = source + target[:n_f - n]
        qc.mct(lst, target[n_f - n])
    qc.mct(source, target[0])
```

source can be either a single qubit or multiple qubits and acts like the control gate. *target* is the counter register. Note that this implementation is the only scalable quantum incrementer. However, for a fixed size of the counting registers one can find more efficient alternatives.

Constraint B

Now we set to implement the actual automatic oracle constructor. The first constraint states that per day, a maximum of one position per operator can be assigned. So we define a list of all registers contained in one day, getting their respective indices through the previously defined time-position dictionary. Afterwards, we loop through those lists and apply the incrementer on each of them. So we have:

```python
for day in range(days):
    pd_lst = []
    for position in range(positions):
        pd_lst.append(time_positions['p%id%i'%(int(position), int(day))])
    registers.append(pd_lst)

for register in registers:
    increment_constraint(register)
```

Now the counter register must be checked if it is equal to zero. If so, a Toffoli gate applies the actual amplitude flipping. Since there are no Toffoli gates in Qiskit that actually check for a $|0\rangle$ state, we simply apply an X gate on both counter qubits. The resulting circuit is presented in Figure 6.2.

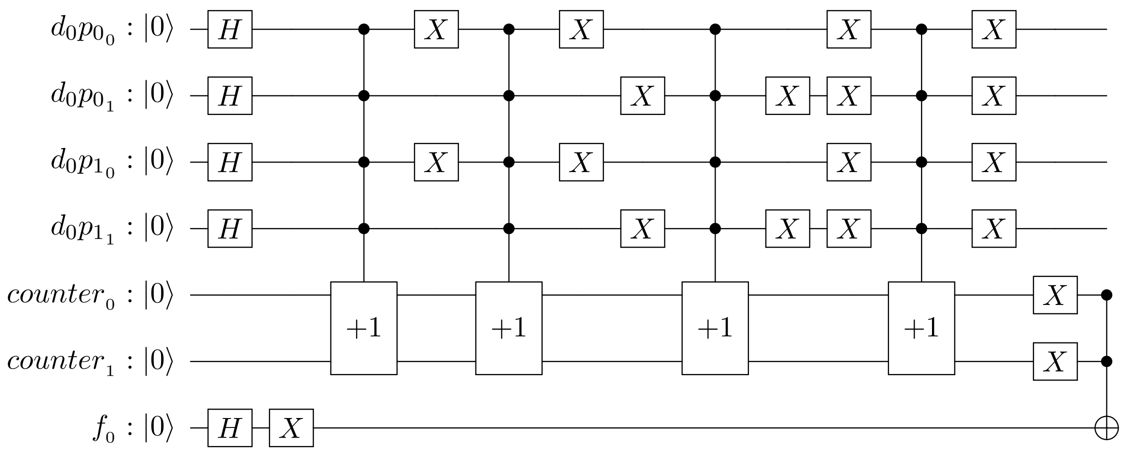


Figure 6.2: Circuit with constraint A

Constraint C

The procedure is similar for constraint C, so that we have:

```python
position_combos = [f'{i:0b}'.zfill(days) for i in range(2**days)]

for pos_co in position_combos:
        pd_lst = []
        for day, position in enumerate(pos_co):
            pd_lst.append(time_positions['p%id%i'%(int(position), int(day))])
        registers.append(pd_lst)

    for register in registers:
        increment_constraint(register)
```

The difference is, that we define a list *position_combos*, which contains binary numbers indicating that an operator is working three days in a row. Our now composed circuit is shown in Figure 6.3.

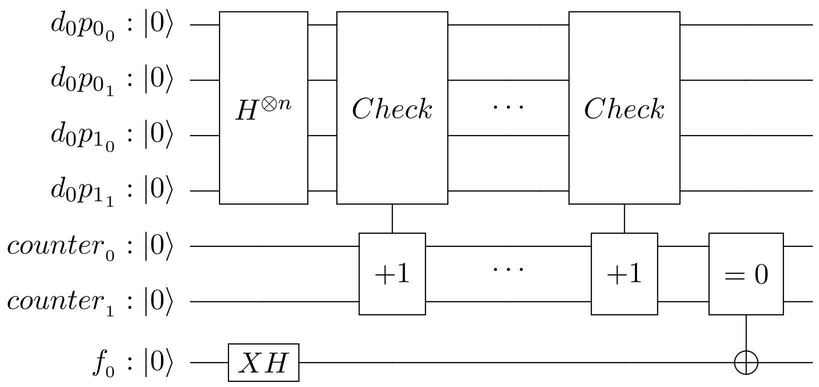


Figure 6.3: Circuit with both constraints

Until now, all valid states are entangled and marked. Since we need to apply the Grover iterator multiple times, we need to de-entangle the states again and reset the counter register.

As explained above, we are doing this by using the same constraint subset to decrement the counter register using the decrement operator described in Figure 6.1.

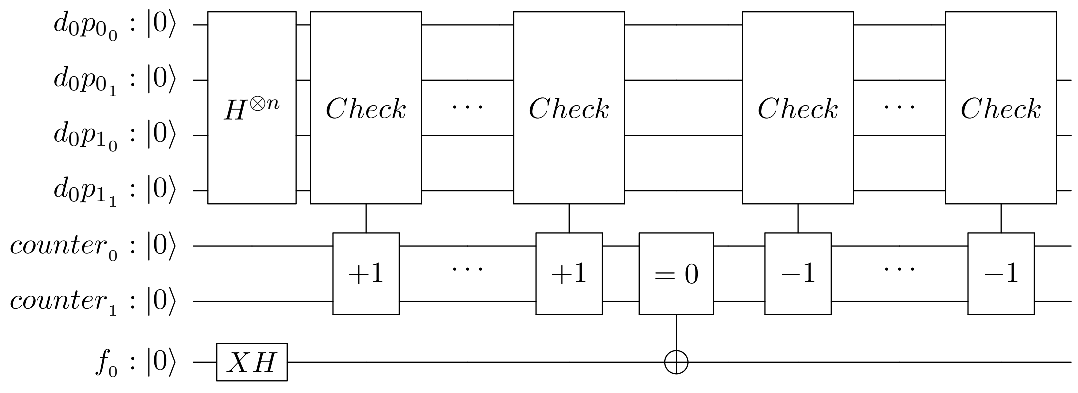


Figure 6.4: Circuit with operational oracle

Grover Diffusion

So far we have implemented two of the three main functions of Grover's algorithm, state preparation and the oracle. In order to amplify the marked states we need to apply the Grover diffusion operator. This operator is fixed in its functionality as described by

$$\text{Diffusion operator} := (2\,|\psi\rangle\,\langle\psi| - I) \tag{6.2}$$

and independent from the oracle. However, it depends on the size of the input register, since I has the dimension as the input state $|\psi\rangle$. The actual implementation is:

```python
def grover_diffuser(qc, n_nodes: int):
    for x in range(n_nodes):
        qc.h(x)
    for x in range(n_nodes):
        qc.x(x)
    qc.h(n_nodes-1)
    qc.mcx([qc[x][1][0] for x in range(n_nodes)], qc.qubits[-1].register)
    qc.h(n_nodes-1)
    for x in range(n_nodes):
        qc.x(x)
    for x in range(n_nodes):
        qc.h(x)
    return qc
```

The method takes a quantum circuit and the number of input qubits as an input, applies the necessary gates and returns the corresponding circuit shown in Figure 6.5.

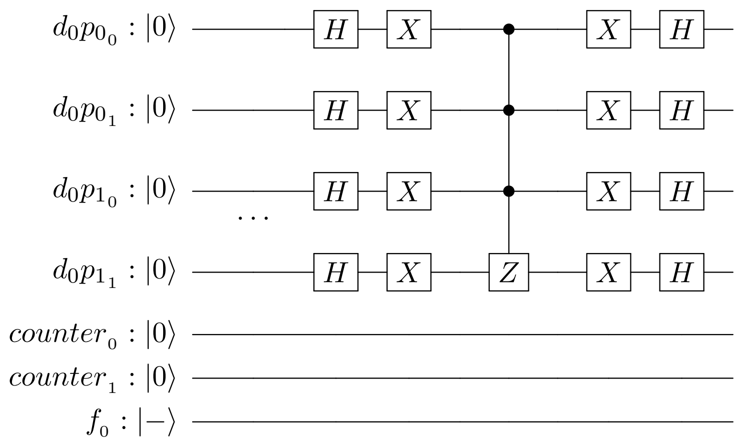


Figure 6.5: Grover diffusor

Its functionality is explained in Equation 2.28. One note, to actually flip the phase about the mean, one can either use a Z gate or an X gate on the Grover qubit. With the Grover diffusor, we can now compose the Grover operator. The corresponding finale circuit with one Grover iteration is given in Figure 6.6.

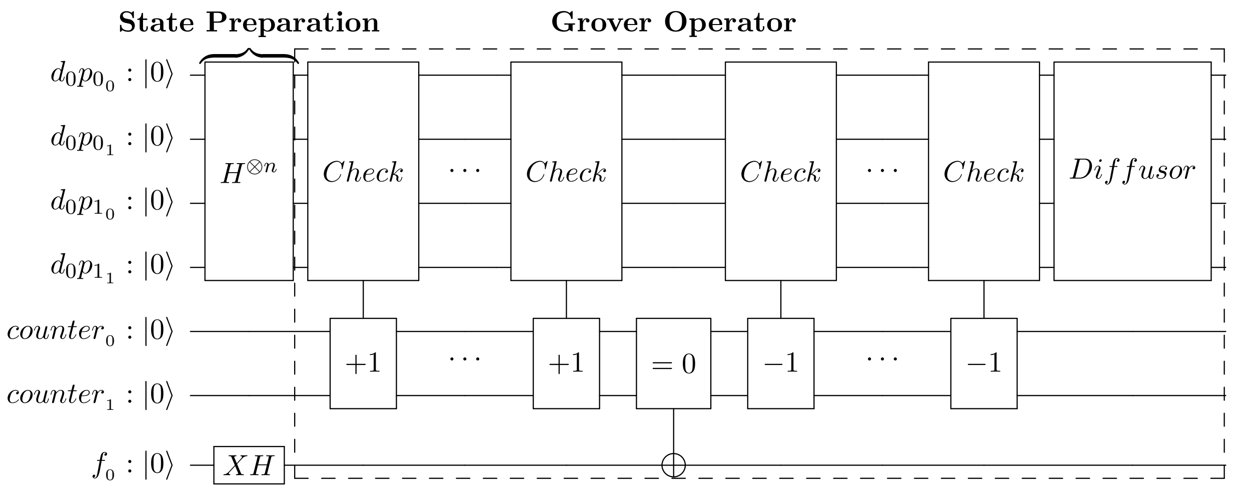


Figure 6.6: Finale circuit with Grover operator and state preparation

6.4 Results

After the Grover operator is repeated as often as required, a measurement to the input register is applied, which gives us back a bit string that states on which time-position each operator work or not.

Validity Check

Since the scheduling problem is in NP, we can check in polynomial time if the returned bit-string represents a valid solution. For this, we developed a simple method that checks the returned bit-string according to each constraint. Because we run the simulation multiple times, it takes the resulting bit-strings as a list in *results*, the log_2(number of operators) and the number of days *days* as an input and returns a dictionary of valid results. A further modification of this method calculates the share of valid solutions in all solutions to measure how good the algorithm performs. The corresponding code for the former method is:

```python
def check_validity(results, log2_operators, days):

    valid_results = {}

    for elem in results.keys():
      # Separates the solution bitstring into chunks the size of
    log2_operators. Necessary to
      # identify individual operators and check their assignments.
        o_div = [elem[i:i + log2_operators]
                 for i in range(0, len(elem), log2_operators)]
        o_div_new = [o_div[i:i + log2_operators]
                 for i in range(0, len(o_div), log2_operators)]

        o_div_bool = True

        # Check for solution with less than 3 shifts
        for sub_elem in o_div_new[0]:
            summe_max3 = 0
            for i in o_div:
                if sub_elem == i:
                    summe_max3 += 1
                if summe_max3 >= 3:
                    o_div_bool = False
                    break

        # Check for multiple shifts per day
        for i in o_div_new:
            if o_div_bool == False:
                break
            new_list = []
            for oper in i:
                if oper not in new_list:
                    new_list.append(oper)
                else:
                    o_div_bool = False
                    break

    # Creates dictionary of valid results
        if o_div_bool:
            valid_results[elem] = results[elem]

      return valid_results
```

Execution Method

We now define a method that takes our quantum circuit as an input and runs it on the IBM backend *qasm_simulator*. It simulates a noiseless quantum computer with up to 30 qubits. Next, we need to define the number of execution shots as $shots = 10000$. It defines how often the circuit is run on the device. The results are stored in the dictionary *results*, which in turn delivers the input to the validity check method. Hence, our execution method is defined as:

```
def run_simulator(qc: QuantumCircuit, shots = 10000):
    # defines backend
    backend = Aer.get_backend('qasm_simulator')
    # tracks overall execution time
    start_time = time()
    # stores the original results
    orig_counts = execute(qc, backend=backend, shots=shots)
                    .result().get_counts()
    end_time = time()
    print("Total Job Submission Time - %.5f (ms)"
              % ((end_time - start_time) * 1000))
    # converts orig_counts for check_validity method
    results = {k: v for k, v in sorted(orig_counts.items(),
                    key=lambda item: item[1], reverse=True)}

    return results
```

We can now run our tool. We need to define five different parameters: number of days $DAYS = 3$, number of positions $POSITIONS = 2$, number of operators $OPERATORS = 4$, size of the counter register $COUNTER\_SIZE = 2$ and number of Grover iterations $GROVER\_ITERATIONS = 1$. The respective implementation is given by:

```
DAYS = 3
POSITIONS = 2
OPERATORS = 4
log2_operators = math.ceil(math.log2(OPERATORS))
VAR_SIZE = POSITIONS * DAYS * log2_operators

COUNTER_SIZE = 2

GROVER_ITERATIONS = 1

SHOTS = 10000

qc = create_circuit(POSITIONS, DAYS, OPERATORS, COUNTER_SIZE)
qc = add_constraint(qc, POSITIONS, DAYS, OPERATORS)
qc = add_diffuser(qc, VAR_SIZE)
qc = add_measure(qc, VAR_SIZE)
results = run_simulator(qc, SHOTS)
check_validity(results, log2_operators, DAYS)
```

Results

As a result we receive:

```
Total Job Submission Time - 547.50896 (ms)
Out of 2357 different results, there are 863 valid results.
In total, 61.05 % of the solutions were correct.
```

So from 10.000 shots, we receive 2357 different schedules as a result, of which 863 are valid. Overall, the share of valid schedules within the 10.000 shots is 61.05%. We can compare this to a benchmark, where we do not apply the Grover operator. Here 21.77% of the results are valid schedules. So we see, that the underlying circuit actually works and finds valid schedules within the search space. Grover's algorithm never delivers the answer with a

probability of one, that is why we need to run the circuit multiple times. If we can increase the share of valid schedules will be the subject of further studies.
To return a solution, we simply pick the state with the highest occurrence. In this case, the corresponding schedule is:

OnCall Operator Schedule			
Days	Day 0	Day 1	Day 2
Operator 0	Position 0	—	—
Operator 1	Position 1	—	Position 0
Operator 2	—	Position 1	Position 1
Operator 3	—	Position 0	—

6.5 Evaluation

The following section will further evaluate the results and show how our model would perform on a real quantum device. It will end with an outlook on how in general methods based on Grover's algorithm perform in the NISQ era.

Optimal number of iterations and size of the counter register

To check whether we found the optimal number of Grover iterations, we can run the circuit multiple times with different iterations. Starting from zero iterations, our above-mentioned benchmark, we expect a sinusoidal change in correct solutions with a growing number of iterations. As we can see in Figure 6.7, the share of correct solutions is indeed at its peak with one Grover iteration. Further, we can check how different sizes of the counter register affect the share of correct solutions. As Figure 6.8 indicates, a counter register of two qubits is necessary to identify all the constraints. It is important to keep the size of the counter register low, since additional qubits are either rare with real quantum devices or increase our computational time significantly as shown in Figure 6.9. So our choice of a counter register consisting of two qubits is optimal.

Width and depth of our circuit

After we determined the optimal number of iterations and counter qubits, we can define the gate complexity and the width and depth of the circuit. With a problem size of four operators, three days and two positions, as well as a counter register of two qubits and one Grover iteration, the overall width of the circuit is 27. Included are 15 qubits and 12 classical measurement bits. Its depth is 320, with a gate complexity of 738 consisting of 509 X-gates, 177 multi-Toffoli gates and 39 Hadamard gates. [2]

[2]Since real quantum devices can not implement multi-Toffoli gates directly, we would need to further transpile the circuit so that it only consists of rotation gates, here denoted as u3, and CX gates. This would lead to a circuit depth of 86749 and a gate complexity of 104906, consisting of 52988 u3 gates and 51906 CX gates.

Run on real quantum device

So far, we used IBM's Aer package with the *qasm_simulator* to simulate a perfect quantum computer, since current, publicly available devices do either not have enough qubits to run our circuit or the calculation time of our circuits exceeds the time limit on the devices. However, there is a way to test its performance on a quantum computer. We can use artificial noise that imitates the error rates of a NISQ era device.
Qiskit offers a variety of options to introduce noise while executing on the *qasm_simulator*. One can either manually assign different error rates to certain gates or use the error rates of an existing quantum device. In our example, we use the noise of the backend 'ibmq_16_melbourne', which is available through the IBMQ cloud. It has 15 super-conducting qubits with a quantum volume of 8. [3] So executing the same circuit as above leads to:

```
Total Job Submission Time - 5493257.68423 (ms)
Out of 4096 different results, there are 912 valid results.
In total, 29.287000000000003 % of the solutions were correct.
```

We see that the result is similar to our benchmark, a near uniform distribution of all possible states. As an effect, the solution states are no longer distinguishable from the other states, which means that our method no longer returns a solution to the scheduling problem. Those results are mainly explained through the high error rates of current quantum devices like the 'ibmq_16_melbourne'. While the error rate could be reduced by the application of quantum error correction, this would in turn require more qubits and gates that would further shrink our executable problem sizes. The relatively long submission time is caused by the transpilation of the circuit and by simulating the error for each gate.

This leads to the general question, whether it makes sense to implement our method on NISQ era devices. We showed that even small problem sizes require a circuit length or depth that are not implementable on current quantum computers. But there is even a further discussion on whether a quadratic speed up, as provided maximally by Grover's algorithm, is enough to provide a quantum advantage in a time where expensive error correction must be applied.

Quadratic speedups on NISQ era devises

[BMG+20] addresses this question by investigating which conditions must be satisfied such that a polynomial speedup realizes an actual quantum advantage in a fault-tolerant quantum device. Especially algorithms that use amplitude amplification, such as Grover's algorithm, are highly parallelizable on classical devices, which can lead to speedups up to an order of 10^3. Unfortunately, this does not shine a good light on the use of quantum algorithms that provide a quadratic speedup on NISQ era devices, since the 'break-even' point of quantum advantage in runtime lies at up to 880 years if the classical solution is parallelized. But there is some hope since only a quartic polynomial speedup would reduce this 'break-even' point to 29 minutes for the same parallelism speedup. Such query reduction algorithms were proposed by [ABB+17] and [ABDKT20], but are still short of a practical use case and implementation.

In this chapter we showed the implementation of our method using Qiskit. We executed the method on its *qasm_simulator*, presented and evaluated the results and showed what

[3] Quantum Volume is a measurement of quantum computational performance introduced by IBM. For more information see [CBS+19]

an execution on a noisy current quantum device would return. In the end we discussed the general question whether it makes sense to implement our method on a NISQ era quantum device.

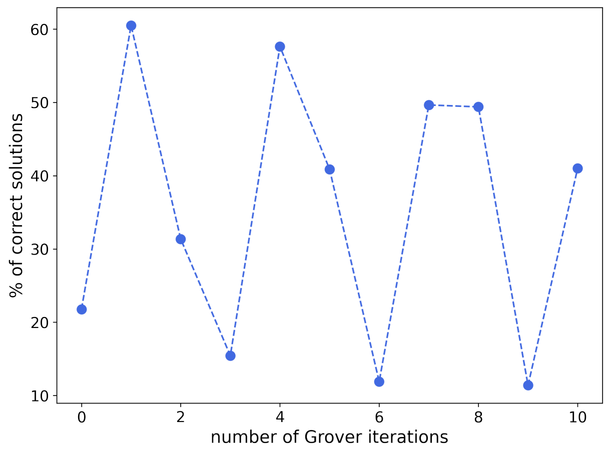


Figure 6.7: Share of correct solutions over Grover iterations

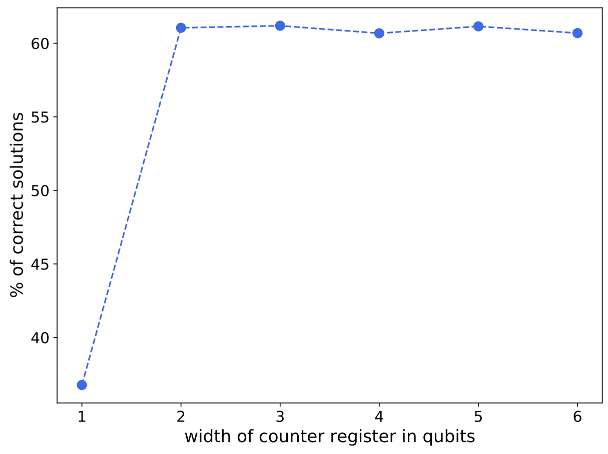


Figure 6.8: Share of correct solutions over counter register width

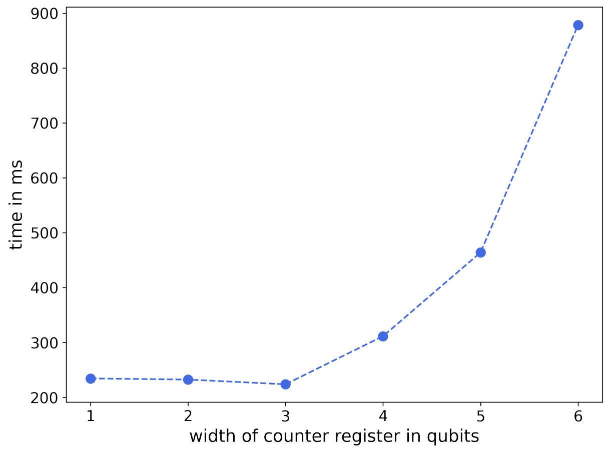


Figure 6.9: Computational time over counter register width

7 Conclusion

This thesis provides a scalable method to solve a scheduling problem with Grover's search algorithm. As an example, we use the on-call spacecraft operator scheduling problem faced by the German Aerospace Center, which seeks for an optimal assignment of 52 operators to 17 positions over a period of 180 days, under constraints on schedule and personnel. With a quadratic computational speed up, Grover's algorithm promises to find an optimal solution for the underlying combinatorial optimization problem. As NISQ era quantum devices are scarce in available qubits, we developed a new way of mapping the problem onto the respective qubits. In contrast to the usual one-to-one mapping of each combinatorial binary variable to a qubit, we represent every operator through a binary number in a quantum register for every time-position. This graph coloring inspired encoding process reduces the number of required qubits significantly and simultaneously implements one of our three constraints. The remaining two constraints are implemented in the oracle, for which we created an automatic construction method that scales with different problem sizes. This is necessary, since the available number of qubits do not allow the encoding and solving of the whole problem size. To test the functionality of our implementation, we reduced the problem size to a batch consisting of 4 operators, 2 positions and 3 days. For this instance, we executed our circuit on Qiskit's qasm_simulator which imitates a noiseless quantum computer. The results show that our implementation finds a solution to our problem within the search space. For our reduced batch size, we further evaluated the optimal number of Grover iterations and the required number of counting qubits for our automatic oracle construction method. Due to the length of our circuit, we were not able to test our implementation on a real quantum device, but a simulation of the expected level of noise on the qasm_simulator showed that the error rate is too high to produce valid results. Even though it became clear that solving the on-call operator scheduling problem on real quantum devices is not feasible at the moment, our scalable implementation can be used to assess the performance of Grover's algorithm on future quantum devices. For that, the presented code is publicly available on GitHub[1].

Future work

While Grover's algorithm provides at most a quadratic speedup, the actually achieved speedup varies with the respective implementation. Therefore, the lower and upper bounds of our implementation should be defined and compared to the performance of state-of-the-art classical algorithms. Further, it should be investigated whether the share of correct solutions can be improved and how it changes with a growing number of implemented constraints. To improve performance, it might also be necessary to optimize the oracle by eliminating redundant gates and to study the potential use of methods like autoencoders to further enhance the encoding process. Another direction of future work would be to expand the constraint space and consider the optimziation of soft constraints, which would also allow a full integration of the method within the German Aerospace Center's scheduling tool.

[1]https://github.com/schererant/operator-scheduling

List of Figures

Bibliography

[ABB+17] Ambainis, Andris ; Balodis, Kaspars ; Belovs, Aleksandrs ; Lee, Troy ; Santha, Miklos ; Smotrovs, Juris: Separations in query complexity based on pointer functions. In: *Journal of the ACM (JACM)* 64 (2017), Nr. 5, S. 1–24

[ABDKT20] Aaronson, Scott ; Ben-David, Shalev ; Kothari, Robin ; Tal, Avishay: Quantum Implications of Huang's Sensitivity Theorem. In: *arXiv preprint arXiv:2004.13231* (2020)

[ABHW73] Abernathy, William J. ; Baloff, Nicholas ; Hershey, John C. ; Wandel, Sten: A three-stage manpower planning and scheduling model—a service-sector example. In: *Operations Research* 21 (1973), Nr. 3, S. 693–711

[ABL06] Aickelin, Uwe ; Burke, Edmund K. ; Li, Jingpeng: Improved squeaky wheel optimisation for driver scheduling. In: *Parallel Problem Solving from Nature-PPSN IX*. Springer, 2006, S. 182–191

[ABL08] Aickelin, Uwe ; Burke, Edmund K. ; Li, Jingpeng: An evolutionary squeaky wheel optimization approach to personnel scheduling. In: *IEEE Transactions on evolutionary computation* 13 (2008), Nr. 2, S. 433–443

[AVDK+08] Aharonov, Dorit ; Van Dam, Wim ; Kempe, Julia ; Landau, Zeph ; Lloyd, Seth ; Regev, Oded: Adiabatic quantum computation is equivalent to standard quantum computation. In: *SIAM review* 50 (2008), Nr. 4, S. 755–787

[AZD13] Akbari, Mohammad ; Zandieh, M ; Dorri, Behrouz: Scheduling part-time and mixed-skilled workers to maximize employee satisfaction. In: *The International Journal of Advanced Manufacturing Technology* 64 (2013), Nr. 5-8, S. 1017–1027

[BBDB+13] Bergh, Jorne Van d. ; Beliën, Jeroen ; De Bruecker, Philippe ; Demeulemeester, Erik ; De Boeck, Liesje: Personnel scheduling: A literature review. In: *European journal of operational research* 226 (2013), Nr. 3, S. 367–385

[BBK+10] Bai, Ruibin ; Burke, Edmund K. ; Kendall, Graham ; Li, Jingpeng ; McCollum, Barry: A hybrid evolutionary approach to the nurse rostering problem. In: *IEEE Transactions on Evolutionary Computation* 14 (2010), Nr. 4, S. 580–590

[BBW03] Bulger, David ; Baritompa, William P. ; Wood, Graham R.: Implementing pure adaptive search with Grover's quantum algorithm. In: *Journal of optimization theory and applications* 116 (2003), Nr. 3, S. 517–529

[Ben82] BENIOFF, Paul: Quantum mechanical hamiltonian models of turing machines. 29 (1982), Nov, Nr. 3, S. 515–546. `http://dx.doi.org/10.1007/BF01342185`. – DOI 10.1007/BF01342185. – ISSN 1572–9613

[BHMT02] BRASSARD, Gilles ; HOYER, Peter ; MOSCA, Michele ; TAPP, Alain: Quantum amplitude amplification and estimation. In: *Contemporary Mathematics* 305 (2002), S. 53–74

[Bib20] *Qiskit*. `https://qiskit.org`. Version: Dec 2020. – [Online; accessed 12. Dec. 2020]

[BMG$^+$20] BABBUSH, Ryan ; MCCLEAN, Jarrod ; GIDNEY, Craig ; BOIXO, Sergio ; NEVEN, Hartmut: Focus beyond quadratic speedups for error-corrected quantum advantage. In: *arXiv preprint arXiv:2011.04149* (2020)

[Bra97] BRASSARD, Gilles: Searching a quantum phone book. In: *Science* 275 (1997), Nr. 5300, S. 627–628

[BV97] BERNSTEIN, Ethan ; VAZIRANI, Umesh: Quantum complexity theory. In: *SIAM Journal on computing* 26 (1997), Nr. 5, S. 1411–1473

[BW06] BARD, Jonathan F. ; WAN, Lin: The task assignment problem for unrestricted movement between workstation groups. In: *Journal of Scheduling* 9 (2006), Nr. 4, S. 315–341

[CBS$^+$19] CROSS, Andrew W. ; BISHOP, Lev S. ; SHELDON, Sarah ; NATION, Paul D. ; GAMBETTA, Jay M.: Validating quantum computers using randomized model circuits. In: *Physical Review A* 100 (2019), Nr. 3, S. 032328

[CLPR10] CORDEAU, Jean-François ; LAPORTE, Gilbert ; PASIN, Federico ; ROPKE, Stefan: Scheduling technicians and tasks in a telecommunications company. In: *Journal of Scheduling* 13 (2010), Nr. 4, S. 393–409

[Dan54] DANTZIG, George B.: Letter to the editor—A comment on Edie's "Traffic delays at toll booths". In: *Journal of the Operations Research Society of America* 2 (1954), Nr. 3, S. 339–341

[DH96] DURR, Christoph ; HOYER, Peter: A quantum algorithm for finding the minimum. In: *arXiv preprint quant-ph/9607014* (1996)

[Edi54] EDIE, Leslie C.: Traffic delays at toll booths. In: *Journal of the operations research society of America* 2 (1954), Nr. 2, S. 107–138

[EPR35] EINSTEIN, Albert ; PODOLSKY, Boris ; ROSEN, Nathan: Can quantum-mechanical description of physical reality be considered complete? In: *Physical review* 47 (1935), Nr. 10, S. 777

[exa20] EXAMPLES dwave: *employee-scheduling*. `https://github.com/dwave-examples/employee-scheduling`. Version: Dec 2020. – [Online; accessed 13. Dec. 2020]

[Fey82] Feynman, Richard P.: Simulating physics with computers. 21 (1982), Jun, Nr. 6, S. 467–488. `http://dx.doi.org/10.1007/BF02650179`. – DOI 10.1007/BF02650179. – ISSN 1572–9575

[FGG14] Farhi, Edward ; Goldstone, Jeffrey ; Gutmann, Sam: A quantum approximate optimization algorithm. In: *arXiv preprint arXiv:1411.4028* (2014)

[FGGS00] Farhi, Edward ; Goldstone, Jeffrey ; Gutmann, Sam ; Sipser, Michael: Quantum computation by adiabatic evolution. In: *arXiv preprint quant-ph/0001106* (2000)

[FH16] Farhi, Edward ; Harrow, Aram W.: Quantum supremacy through the quantum approximate optimization algorithm. In: *arXiv preprint arXiv:1602.07674* (2016)

[GDT09] Goodman, Melissa D. ; Dowsland, Kathryn A. ; Thompson, Jonathan M.: A grasp-knapsack hybrid for a nurse-scheduling problem. In: *Journal of Heuristics* 15 (2009), Nr. 4, S. 351–379

[Gid20] Gidney, Craig: Quantum block lookahead adders and the wait for magic states. In: *arXiv preprint arXiv:2012.01624* (2020)

[Gro96] Grover, Lov K.: A fast quantum mechanical algorithm for database search. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, S. 212–219

[Gro05] Grover, Lov K.: Fixed-point quantum search. In: *Physical Review Letters* 95 (2005), Nr. 15, S. 150501

[GWG19] Gilliam, Austin ; Woerner, Stefan ; Gonciulea, Constantin: Grover Adaptive Search for Constrained Polynomial Binary Optimization. In: *arXiv preprint arXiv:1912.04088* (2019)

[INH19] Ikeda, Kazuki ; Nakamura, Yuma ; Humble, Travis S.: Application of Quantum Annealing to nurse Scheduling problem. In: *Scientific reports* 9 (2019), Nr. 1, S. 1–10

[JSV98] Jaumard, Brigitte ; Semet, Frederic ; Vovor, Tsevi: A generalized linear programming model for nurse scheduling. In: *European journal of operational research* 107 (1998), Nr. 1, S. 1–18

[KN98] Kadowaki, Tadashi ; Nishimori, Hidetoshi: Quantum annealing in the transverse Ising model. In: *Physical Review E* 58 (1998), Nr. 5, S. 5355

[LYTJ$^+$14] Li, Xiaoyu ; Yang, Guowu ; Torres Jr, Carlos M. ; Zheng, Desheng ; Wang, Kang L.: a Class of Efficient Quantum Incrementer Gates for Quantum Circuit Synthesis. In: *International Journal of Modern Physics B* 28 (2014), Nr. 01, S. 1350191

[NC02] Nielsen, Michael A. ; Chuang, Isaac: *Quantum computation and quantum information*. 2002

[P$^+$13] PERUZZO, A u. a.: A variational eigenvalue solver on a quantum processor. eprint. In: *arXiv preprint arXiv:1304.3061* (2013)

[Pre18] PRESKILL, John: Quantum Computing in the NISQ era and beyond. In: *Quantum* 2 (2018), Aug, 79. `http://dx.doi.org/10.22331/q-2018-08-06-79`. – DOI 10.22331/q–2018–08–06–79. – ISSN 2521–327X

[QH08] QU, Rong ; HE, Fang: A hybrid constraint programming approach for nurse rostering problems. In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence* Springer, 2008, S. 211–224

[SBC$^+$20] SANDERS, Yuval R. ; BERRY, Dominic W. ; COSTA, Pedro C. ; TESSLER, Louis W. ; WIEBE, Nathan ; GIDNEY, Craig ; NEVEN, Hartmut ; BABBUSH, Ryan: Compilation of fault-tolerant quantum heuristics for combinatorial optimization. In: *PRX Quantum* 1 (2020), Nr. 2, S. 020312

[Sho99] SHOR, Peter W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In: *SIAM review* 41 (1999), Nr. 2, S. 303–332

[Tof80] TOFFOLI, Tommaso: Reversible computing. In: *International Colloquium on Automata, Languages, and Programming* Springer, 1980, S. 632–644

[YLC14] YODER, Theodore J. ; LOW, Guang H. ; CHUANG, Isaac L.: Fixed-point quantum search with an optimal number of queries. In: *Physical review letters* 113 (2014), Nr. 21, S. 210501

[YM08] YANOFSKY, Noson S. ; MANNUCCI, Mirco A.: *Quantum computing for computer scientists.* Cambridge University Press, 2008

[ZZ17] ZAHEDINEJAD, Ehsan ; ZARIBAFIYAN, Arman: Combinatorial optimization on gate model quantum computers: A survey. In: *arXiv preprint arXiv:1708.05294* (2017)