Quantum Shift Scheduling – A Comparison to Classical Approaches

Sven Prüfer, Antonius Scherer, Andreas Spörl, Tobias Guggemos, Nikolas Pomplun, Christoph Lenzen
DLR e.V. - German Aerospace Center
Münchner Straße 20
82234 Weßling, Germany

Abstract

Solving discrete optimization problems with constraints is a very common task in industry and research as it is fundamental in solving many planning tasks.

In this paper we will look at an instance of a time table problem for generating shift schedules at the German Space Operation Center (GSOC). We describe the implementation of a quantum approach and compare the differences to classical optimization strategies, knowing that the problem sizes given to the quantum systems are not competitive yet. By doing so we are establishing a software chain that is able to map our problem to different physical systems which paves the way to problem solving as a hybrid solution where sub-problems are distributed among classical and quantum hardware.

In this study we included three approaches to tackle the described problem. For the quantum part, we included a programmatically generated quantum circuit that yields a solution to a (sub) problem using Grover's algorithm, able to be run on any general quantum computer with sufficiently many qubits of sufficiently high quality. On the classical side, as a validation and benchmark reference, we use a heuristic search method, implemented by GSOC's own planning tool set Plato and PINTA (Lenzen et al. 2012; Nibler et al. 2021) as well as a constraint integer programming formulation solved by an external software framework, such as e. g. GLPK or SCIP (Gamrath et al. 2020).

This paper builds on and extends results from (Scherer et al. 2021).

1 Problem Setup

Creating and updating a valid shift schedule for on-ground support of several missions for different time slots, people and qualifications is a common task at GSOC when running operations. This paper shows an approach to solve this combinatorical scheduling problem with the use of Grover's algorithm obeying auxiliary conditions as described below.

In this paper we will use the notation convention similar to the one in (Scherer et al. 2021). Therefore we introduce binary variables for *operators*, *positions* and *time slots* as shown in Table 1 that span the problem space. A valid solution is a $D \times P \times O$-Matrix of zeros and ones indicating operator o scheduled at position p on day d which does not violate any constraints.

Table 1: Variable names used in this paper, uppercase letters refer to the problem parameter count, whereas lowercase letters refer to specific indices.

Parameter	Total Count	Variable
Operators	O	o
Time Slots (Days)	D	d
Positions	P	p
Given schedule	-	Y_{dpo}
Domain of schedule	-	T

We selected the constraints listed in Table 2 to be required for a valid solution.

At GSOC, with roughly 50 operators that need to work on 20 positions for planning periods of up to 365 days, the search space amounts to roughly $366,000$ binary variables. Due to various other constraints, there usually is no completely valid solution, so some manual manipulation afterwards is needed in order to ignore some individual constraints and arrive at a (nearly-valid) plan. However, to be able to run the computations, we restrict the planning duration in particular for Section 3.3 considerably. Since personnel tend to share positions inside missions, the distribution of operators on positions is rather sparse. This fact may be exploited to reduce the problem size considerably Sections 3.2 and 3.3.

2 Quantum Scheduling

In this section we want to give a brief introduction to the basic ideas and ingredients required to understand the working of Grover's algorithm when searching for a valid solution inside the problem space. We focus on quantum computers using a gate-model, but there exist different approaches such as quantum annealers requiring a different description. See e. g. (Nielsen and Chuang 2010) for a general and thorough introduction.

2.1 Quantum Computing

Similar to classical computers which compute on *bits*, the basic computational unit of a quantum computer is a *qubit* (or registers of multiple qubits). A qubit's state is described by a two-dimensional complex unit vector, i. e. $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ with $|\alpha|^2 + |\beta|^2 = 1$ and $\{|0\rangle, |1\rangle\}$ form

Table 2: Constraints for a valid schedule.

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

[a] This is called an *outage*

[b] This is needed e. g. for updating an on-call schedule during the year when operators have updated their vacations. In this case only the future will be replanned, but the past may influence the applied constraints in this planning interval

an orthonormal basis of $\mathbb{C}^2$ equipped with the standard inner product. Such a linear combination is called a *superposition* of the states $|0\rangle$ and $|1\rangle$.

Similarly to a classical computer, there are *gates* modifying the amplitudes α, β of the qubit's state. These gates correspond to quantum-mechanical operators evolving the qubit state and thus need to be unitary, i. e. complex-linear and preserving the inner product. Common single-qubit gates are the Pauli X, Y, Z gates, or the Hadamard gate H which maps the basis states to the superpositions $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Note that the precise physical implementation of a qubit and the gates may differ drastically between different quantum computers. For example, there exist e. g. superconducting (Clarke and Wilhelm 2008), trapped-ion (Friis et al. 2018) and photonic quantum computers (Adami and Cerf 1999).

To get a result from a quantum computer, it is necessary to *measure* the state of (potentially a subset of) the qubits. In quantum mechanics such a measurement will do two things: It will output the value for the measurement according to a probabilistic process depending on the state but at the same time project the current state onto a state representing the result. As an example, when measuring whether a qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ is in state $|0\rangle$ or $|1\rangle$, we observe $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. When measuring a qubit after applying the H-gate, $|0\rangle$ and $|1\rangle$ are observed with equal probability $1/2$.

To write quantum algorithm we thus need to map our problem onto a set of qubits and add gates corresponding to the algorithm, which should increase the state's amplitude corresponding to the correct solution. At the end we will measure the quantum state and obtain a single result, this will be repeated a suitable amount of times in order to obtain a statistic of observed states and, thus, the most probable solution.

2.2 Grover's Algorithm

Grover's Search quantum algorithm (Grover 1996) features a quadratic speed-up for every problem which can be described as a search in an unsorted database and, thus, problems where finding solutions requires searching through a large part of the input space.

After initialising the input state as a uniform superposition of all possible states, the algorithm contains two more major steps, which are illustrated in Fig. 1:

1. **Initial state** We prepare an input state that corresponds to all possible inputs of the problem, i. e. $\{0, 1, \ldots, 2^n - 1\}$, by applying the aforementioned Hadamard gate on an input register of n qubits. We receive a state $|\psi\rangle = \sum_{k=0}^{2^n-1} a_k\,|k\rangle$ where every possible input will be measured with the same probability $|a_k|^2 = |\frac{1}{\sqrt{2^n}}|^2$
2. **Oracle Invocation** The oracle bears the knowledge about the solution. By invoking it, possible solutions are flagged in the given quantum states. The algorithm uses the fact, that the amplitudes of a superpostion (α, β) can be negative. By negating the amplitude of the inputs that are a solution for the problem (i. e. `010` in Fig. 1), the solution is marked. As a consequence, the arithmetic mean $m = \sum_k a_k$ (red line) over all amplitudes is reduced.
3. **Diffusion** The reduction of the arithmetic mean is used to increase the amplitude of the solution, while reducing those of invalid states. Pictorial speaking, all amplitudes a_k are reflected at m according to $a_{new} = 2m - a_{old}$, during the diffusion step. In the example of Fig. 1 this increases the amplitude of `010` to $\frac{5}{4\sqrt{2}}$ and reduces the others to $\frac{1}{4\sqrt{2}}$.

By repeating step 2 and 3 the measurement probability of the searched space can be significantly increased. The optimal number of iteration is $\frac{4}{\pi}\sqrt{N/k}$, where N is the number of inputs and k is the number of correct solutions. The key for solving a problem with Grover's algorithm is the construction of an efficient set of gates for implementing the *Oracle* in step 2. The implementation of the *Oracle*, or U_f, for the On-Call scheduling problem will be shown in the next section.

3 Approaches to the Scheduling Problem

3.1 Heuristic Search for Solutions

To tackle the on-call problem, GSOC currently uses the *Plato library* for modelling the problem and specifying algorithms as well as Pinta for manual verification and modification, see (Lenzen et al. 2012; Nibler et al. 2021). Several times per year, the on-call plan is adjusted by employing a heuristic search for suitable solutions using the algorithm described as follows. Notice that with the available number and qualification of operators, satisfying all constraints in Table 2 tends

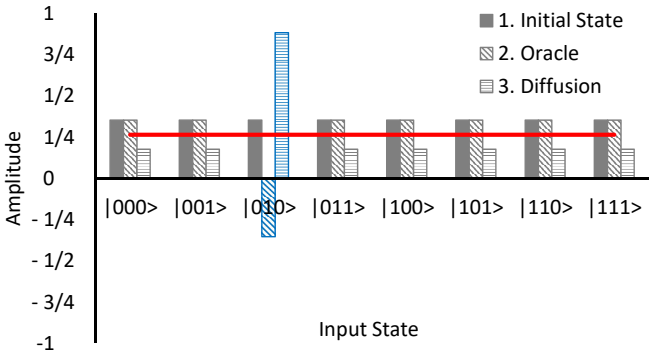


Figure 1: Illustration of the Grover Diffusion with the example of $n = 3$ for the search string `010`.

to be impossible, which is why the algorithm works in an opportunistic way: It schedules as many shifts as possible in a certain order satisfying some subset of constraints and varies parameters to get a set of possibly incomplete plans of which one is chosen according to some criteria. Remaining issues are than fixed by a responsible human on-call planner. The optimization goals are considered in the following order:

1. Minimize the number of times slots that a position is not covered.
2. Minimize the maximum number of scheduled time slots per person.
3. Minimize the maximum number of scheduled time slots per person on holidays.

This algorithm repeats a configurable amount of times using some partially randomized input. Each round contains two nearly identical runs, one for planning whole weeks and a subsequent one for single days to fill gaps left by the first run. From the resulting on-call plans, the algorithm chooses the best result according to the above optimization goals.

One run for either whole weeks or single days looks as follows:

1. Every run tries to fill the timeline by subsequently adding operators to time slots according to the following rules:
   (a) Select an operator with a minimal max-value on the optimization goals, but allow for some random variation.
   (b) For this operator, choose a random position that the operator can serve.
   (c) Add this operator to a time slot for the chosen position by
      i. considering conflict-free times,
      ii. trying to reduce the number of conflicts,
      iii. preferring time slots where the position is not yet occupied,
      iv. preferring time slots far away from existing time slots that the operator may already have, and
      v. some random variation.
2. After adding an on-call shift, the algorithm checks for time slots where positions are covered twice and tries to remove shifts with the maximal max-value on the optimization goals.

Table 3: Constraint implementations for an integer programming optimization suite. Notice that this assumes that the tool supports quadratic constraints or quadratic goals. Also constraints VIII and IX are formulated directly as goals.

Const.	Formula
I	$X_{dpo} = 0 \quad \forall d \in D, o \in O, p \notin P_o$
II	$\sum_{o \in O} X_{dpo} \geq 1 \quad \forall d \in D, p \in P$
III	$\sum_{p \in P} X_{dpo} \leq 1 \quad \forall d \in D, o \in O$
IV	$X_{dpo} = 0 \quad \forall o \in O, d \in D_o, p \in P$
V	$X_{dpo} = Y_{dpo} \quad \forall (d, p, o) \in T$
VI	$\sum_{\substack{i=0..20 \\ p \in P}} X_{d+ipo} \leq 14 \quad \forall o \in O, d \in D_{\text{week}}$
VII	$\sum_{\substack{i=0..104 \\ p \in P}} X_{d+ipo} \leq 35 \quad \forall o \in O, d \in D$
VIII	$\text{min. var}\left(\sum_p X_{dpo}, \ldots, \sum_p X_{d+6po}\right)$
IX	$\text{min. var}\left(\sum_{d,p} X_{dp1}, \ldots, \sum_{d,p} X_{dpO}\right)$

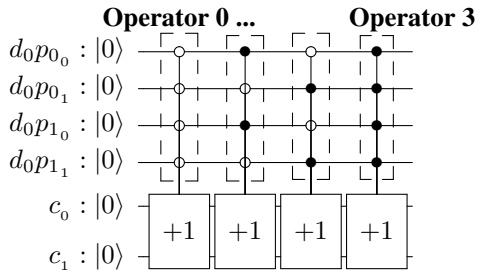
Of course, this algorithm is heuristic and does not generally find global minima. Nevertheless, it has quite a few parameters that one may tweak to get better results and allows easy customization of special cases. We see that the constraints I, III and IV–VII are strictly implemented, whereas the algorithm tries to minimize the violations of constraint II, VIII and IX.

3.2 Integer Programming

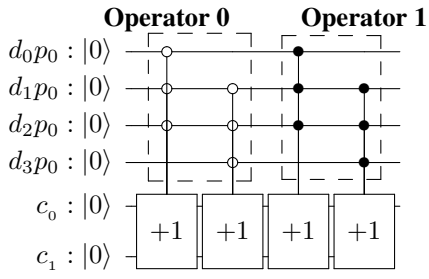
Integer (linear or quadratic) programming and optimization is a non-quantum standard technique for scheduling problems such as the on-call problem at hand. A generic solver for the appropriate subclass of integer programming optimization problem can be applied to find solutions to the problem, such as e. g. SCIP (Gamrath et al. 2020). Notice that there is quite some difference in the speed and capabilities of the available solvers, so performance comparisons between such implementations and other techniques depend heavily on the used software. Modelling the on-call problem for an integer programming optimization is in principle straight forward: A binary variable $X_{dpo} \in \{0, 1\}$ specifies whether or not an operator o is scheduled for position p on timeslot d. We can then translate the constraints from Table 2 as shown in Table 3. Here, we use

- P_o for the set of positions that operator o can support,
- D_o for outages of operator o, i. e. days when they cannot be scheduled,
- Y_{dpo} for a partial on-call plan with (d, p, o) defined in some domain $T \subset D \times P \times O$,
- D_{week} for starting days of on-call weeks, and
- $\text{var}(x_1, \ldots, x_n)$ for the variance of the arguments $x_1, \ldots, x_n$.

Notice that there are multiple reasons why one may need to consider partial plans as inputs. For example, one may extend an already scheduled plan, such as when one replans later parts of a year. Also, it is possible that one embeds



(a) Constraint III for four Operators, one day and two positions; no operator works the same day at two different positions.



(b) Constraint VI for four days, two operators and one position; no operator is allowed to work 3 days in a row.

Figure 2: Exemplary circuits Constraint **III** and **VI** that increment the counter $|c_1c_0\rangle$ if one of the constraints is violated.

this step into an algorithm that first tries to schedule weeks and afterwards refines to days, such as e. g. the heuristic approach in Section 3.1. Furthermore, one may incorporate manual operator input this way. Indeed, the PINTA and Plato approach from Section 3.1 allows this explicitly.

Notice that optimization goals VIII and IX are quadratic and may contain quite a lot of summands causing difficulties when running the optimization tool. It is thus advisable to look for alternative formulations that may relax the problem slightly but are still acceptable. As an example, if x can take only integer values from 0 to 7, the function $x \mapsto x(7-x)$ has minima at 0 and 7, we can thus replace the variance in constraint VIII by

$$\left(\sum_{\substack{i=0..6\\p\in P}} X_{d+ipo}\right)\left(7-\sum_{\substack{i=0..6\\p\in P}} X_{dpo}\right) \tag{1}$$

for every operator o and every on-call week starting on $d \in D_{\text{week}}$. This is an alternative formulation which has fewer terms but optimizes for a subtly different goal.

3.3 Quantum Scheduling

Our third approach to solve the presented Scheduling Problem utilizes Grover's Algorithm in order to find a valid solution. Considering the major steps of Grover's algorithm as in Section 2.2, there are two parts which must be adjusted in order to solve our problem: The encoding or mapping of the problem's variables on the circuit's qubits and a method which allows a scalable construction of oracles that implement the problem's constraints.

Current quantum devices are restricted in the number of available qubits, hence, efficient encoding of the problem to a minimal number of qubits is desired. A naive encoding would assign every binary variable to a single qubit, so that all combinations of day d, position p and operator o are represented in the input state. In contrast, our approach encodes an operator to a time-position in the schedule, as such:

$$|\psi_1\psi_0\rangle_{d,p} = \begin{cases} |00\rangle \text{ , operator } \mathbf{0} \text{ is assigned to position } p \text{ at day } d \\ \cdots \\ |11\rangle \text{ , operator } \mathbf{3} \text{ is assigned to position } p \text{ at day } d \end{cases}$$

This not only reduces the necessary qubits to $D \times P \times \log_2(O)$, but additionally satisfies Constraint II as there is exactly one operator assigned to every tuple of position and day. The other constraints are implemented in the *Grover Oracle* by counting the number of violations of a certain schedule, which is stored in a *counter* register. e. g. , if a state that represents a schedule would imply an operator to work three consecutive days, the counter would be increased by **1**. Fig. 2b and Fig. 2a show our implementation of Grover's Oracle for constraints III and VI respectively. The increase of the counter register is depicted as a controlled $\boxed{+1}$-gate. If and only if the counter equals **0** for a state that represents a schedule, its amplitude will be negated. Hence, there are a maximum of $P \times O \times (D-2)$ increments in the circuit, however only $P(D-2)+D(\binom{P}{2})$ violations can be activated at the same time. Since the constraint counting is binary, this adds a total of $\lceil \log_2(P(D-2)+D(\binom{P}{2}+1)) \rceil$ qubits to the circuit. Those numbers can be seen as upper bounds of the circuit depth and width, and are subject to future optimizations.

Our methods to implement the scalable constraint oracles and the encoding are independent from programming frameworks. In fact, while we mainly used IBM's framework *qiskit*, we can easily convert them to the QASM language, a widely accepted standard for quantum circuits. This allows us to run our simulations on other frameworks, i. e. Atos or Google.

For validation, we use IBM's publicly available simulator which is capable up to 32 qubits. We use a set of small scale sub-problem depicted in Table 4, as well as the constraints II, III and VI in Table 2. The problems differ in the number of days and operators to maximize the number of used qubits. Even though constraint VI originally indicates that an operator can work at most two out of three consecutive *weeks*, we apply it here on days which of course has the same affect as both can be seen as time units.

Given our sub-problems, 15 to 29 qubits are required to encode the operators, days and positions respectively, and another 3 to 4 to implement our conflict counter and perform the phase flip. The increase of required qubits is explained through the higher number of input variables and the accompanied increase in countable conflicts. Executing the circuit through IBM's simulator returns valid results with a success rate of up to $\sim 99\%$. The success rate is calculated by running the circuit 8000 times and counting the returned solutions that represent valid schedules. We can see that the

success rate is stable throughout cases I-IV, with a slight dip for case V. This is mainly explained through the initially high percentage of valid solutions within the search space which limits Grover's search abilities. Case X was used as our true negative test case, for which the optimal number of Grover iterations cannot be determined with the formula given in Section 2.2. Its configuration allows no valid solution which resulted in our method having a correct success rate of 0%.

Of course, using one of IBM's real quantum devices would be desired, unfortunately the circuit depth (number of gates) exceeded their current chip's computational time limits. To get a glimpse on what results we would expect from near term quantum devices, we introduced a simulated gate error rate that mimics one of IBM's available quantum hardware. It decreases the success probability to $\sim 33\%$, which is no longer distinguishable from a uniform distribution over all possible input states.

3.4 Comparison

Comparing the approaches from Sections 3.1 to 3.3 quantitatively is non-trivial as this is highly implementation-dependent and the problem techniques have vastly different ranges of possible problem sizes. To get an idea of considered problem sizes and necessary qubits for the quantum scheduler, consider Table 4. The heuristic approach using the Plato library has a very large overhead for smaller problem sizes. Similarly, current quantum computers are too noisy to run such large circuits but simulators performance depends crucially on the available computation power.

As one can see from the description in Section 3.3, the current Grover implementation does not implement all constraints from Table 2, whereas both the heuristic and the integer programming approach can deal with all constraints. Both of them are easily extendable in case of new requirements, whereas for the quantum scheduler one has to consider and implement each constraint individually. Also, allowing the quantum scheduler to actually optimize instead of just looking for valid solutions requires a different technique or a major change whereas this is easy to implement in the other two approaches.

Both the integer programming and the quantum scheduler techniques suffer from a large number of variables and terms making the implementations grow in size very quickly. Although one can in principle translate some of the constraint implementations from Table 3 to a quantum circuit, the results are rather unfeasible. For example, translating inequality constraints into equalities by using slack variables increases the number of necessary qubits considerably. Thus special care has to be taken when implementing a usable quantum circuit this way. One improvement that one may employ in both integer programming and the quantum scheduling approach is to take the input data better into account when encoding the problem. As an example, it rarely happens that more than eight operators can work at a given position. We can thus drop all variables or qubits that are zero from the start and instead make the encoding input-dependent. This way, we save a lot of variables and summands as well as qubits, respectively. In the case of the quantum circuit this means that implementing additional constraints actually reduces the circuit size.

All three techniques have a well-defined interface and can be nearly exchanged for one another. While the heuristic approach is built into Pinta, the other two algorithms are part of a single software which can import on-call problem input from Pinta. In this way, we can consider these algorithms as back boxes in the overall on-call planning workflow and, up to the existence of large-enough quantum computers, allow the operational on-call schedule to be planned using a quantum computer.

4 Conclusion and Outlook

Knowing of the shortcomings of today's NISQ devices in terms of absolute computational power, we use the classical solvers as a reference of result verification and computational benchmark. Our main focus of course lies on the performance of the quantum algorithm. Due to the absence of a sufficiently large and stable quantum computer to approach a realistic problem setup, we validated our circuits with adequately small input sizes on a quantum simulator capable of processing up to 32 qubits. Even though the problem complexity is small, execution on a real quantum device would still require an improvement of qubit stability and control or application of error-corrected schemes (resulting in more qubits). Nevertheless, once quantum devices become usable, we showed that one can immediately start to incorporate results of sub-problems computed on quantum devices with impressive success rates.

A precursor of software interface utilizes the underlying mathematical representation to map the problem to comparable solvers by classical and quantum means. In a hybrid approach this should be capable of transforming the problem into a performing synergy of quantum and classical processing units with well balanced work loads. To do so, the quantum algorithm is configurable in the range of the variables, allowing the problem to be split by time slices for example.

To judge the value of the interface, a sophisticated comparison with classical methods is still mandatory, but we showed that we are able to establish the interface between classical and quantum computers. This approach allows to run the algorithm as a quantum subroutine within a classical framework.

We restricted our approach to gate based quantum devices (e. g. IBM, Google), but left out quantum annealers such as the one provided by D-Wave. Our problem purely includes logical statements in terms of boolean variables and satisfiability clauses, and can hence be transformed into a QUBO, and used to solve it on D-Dave., as shown here (Ikeda, Nakamura, and Humble 2019). The two quantum computing paradigms have not been directly compared for performance on scheduling problems. Future work should establish a quantitative benchmark for scheduling problems with quantum computers.

Anticipating the next generations of quantum hardware, there will still be restrictions regarding the number of qubits and circuit width. Hence, our next steps are to further minimize the number of required qubits, e. g. by improving conflict counting, and reducing the circuit's width e. g. by applying the ZX-calculus language (Duncan et al. 2020). Additionally, extending the current Grover implementation to a nested

Table 4: Simulation results for different problem sizes used for evaluation.

Case	I	II	III	IV	V	X
Operators	4	4	4	4	8	4
Postions	2	2	2	2	2	3
Days	3	4	5	6	3	3
Used Qubits	15	21	25	29	21	23
Used Counter Qubits	2 (3)	4 (4)	4 (4)	4 (4)	2 (3)	4 (4)
Fraction of Solutions	0.223	0.107	0.048	0.022	0.587	0
Grover Iterations	1	2	3	5	2	-
Success Rate	0.99	0.99	0.99	0.96	0.88	0

Grover Search algorithm (Cerf, Grover, and Williams 2000) or using quantum reinforcement learning (Saggio et al. 2021) are of interest.

References

Adami, C.; and Cerf, N. J. 1999. Quantum Computation with Linear Optics. In Williams, C. P., ed., *Quantum Computing and Quantum Communications*, 391–401. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-49208-5.

Cerf, N. J.; Grover, L. K.; and Williams, C. P. 2000. Nested quantum search and structured problems. *Phys. Rev. A* 61: 032303. doi:10.1103/PhysRevA.61.032303. URL https://link.aps.org/doi/10.1103/PhysRevA.61.032303.

Clarke, J.; and Wilhelm, F. K. 2008. Superconducting quantum bits. *Nature* 453(7198): 1031–1042. doi:10.1038/nature07128. URL https://doi.org/10.1038/nature07128.

Duncan, R.; Kissinger, A.; Perdrix, S.; and van de Wetering, J. 2020. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum* 4: 279. ISSN 2521-327X. doi:10.22331/q-2020-06-04-279. URL http://dx.doi.org/10.22331/q-2020-06-04-279.

Friis, N.; Marty, O.; Maier, C.; Hempel, C.; Holzäpfel, M.; Jurcevic, P.; Plenio, M. B.; Huber, M.; Roos, C.; Blatt, R.; and Lanyon, B. 2018. Observation of Entangled States of a Fully Controlled 20-Qubit System. *Phys. Rev. X* 8: 021012. doi:10.1103/PhysRevX.8.021012. URL https://link.aps.org/doi/10.1103/PhysRevX.8.021012.

Gamrath, G.; Anderson, D.; Bestuzheva, K.; Chen, W.-K.; Eifler, L.; Gasse, M.; Gemander, P.; Gleixner, A.; Gottwald, L.; Halbig, K.; Hendel, G.; Hojny, C.; Koch, T.; Le Bodic, P.; Maher, S. J.; Matter, F.; Miltenberger, M.; Mühmer, E.; Müller, B.; Pfetsch, M. E.; Schlösser, F.; Serrano, F.; Shinano, Y.; Tawfik, C.; Vigerske, S.; Wegscheider, F.; Weninger, D.; and Witzig, J. 2020. The SCIP Optimization Suite 7.0. Technical report, Optimization Online. URL http://www.optimization-online.org/DB_HTML/2020/03/7705.html.

Grover, L. K. 1996. A fast quantum mechanical algorithm for database search. In Miller, G. L., ed., *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 212–219. New York, NY: ACM. ISBN 0897917855.

Ikeda, K.; Nakamura, Y.; and Humble, T. S. 2019. Application of Quantum Annealing to Nurse Scheduling Problem. *Scientific Reports* 9: 12837. doi:10.1038/s41598-019-49172-3. URL https://doi.org/10.1038/s41598-019-49172-3.

Lenzen, C.; Wörle, M. T.; Mrowka, F.; Spörl, A.; and Klaehn, R. 2012. The Algorithm Assembly Set of Plato. In *SpaceOps 2012 Conference*, SpaceOps Conferences. American Institute of Aeronautics and Astronautics. doi:10.2514/6.2012-1255731. URL https://doi.org/10.2514/6.2012-1255731.

Nibler, R.; Mrowka, F.; Hartung, J.; Schneider, A.; and Brogl, S. 2021. PINTA – one Tool to plan them all. To Be Published.

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

Saggio, V.; Asenbeck, B. E.; Hamann, A.; Strömberg, T.; Schiansky, P.; Dunjko, V.; Friis, N.; and et al., S. W. 2021. Experimental Quantum Speed-up in Reinforcement Learning Agents. *Nature* 591: 229–33. URL https://doi.org/10.1038/s41586-021-03242-7.

Scherer, A.; Guggemos, T.; Grundner-Culemann, S.; Pomplun, N.; Prüfer, S.; and Spörl, A. 2021. OnCall Operator Scheduling for Satellites with Grover's Algorithm. In Paszynski, M.; Kranzlmüller, D.; Krzhizhanovskaya, V. V.; Dongarra, J. J.; and Sloot, P. M. A., eds., *Computational Science – ICCS 2021*, 17–29. Cham: Springer International Publishing. ISBN 978-3-030-77980-1.