

---

# A QUADRATIC UNCONSTRAINED BINARY OPTIMIZATION APPROACH FOR QUBIT MAPPING

---

A thesis presented for the title of  
**Diplom-Ingenieurin (Dipl.-Ing.)**

within the studies of  
**Computational Science and Engineering (066 646)**

submitted by  
**Luise Prielinger, MSc**  
Student no. 01426567

supervised by  
Assoc.-Prof. Dr. Nysret Musliu  
Ass.-Prof. Dr. Sebastian Feld

conducted at the Databases and Artificial Intelligence Group,  
Faculty of Informatics, TU Wien

in cooperation with the Quantum Machine Learning Group,  
Quantum and Computer Engineering Department, TU Delft

Vienna, on 12. Juni 2023

\_\_\_\_\_  
(Signature author)

\_\_\_\_\_  
(Signature supervisor)

## Abstract

Logical bit operations are an essential part of computer programming. They involve manipulating individual bits within binary numbers to perform various tasks, such as bitwise AND, OR, XOR, and NOT operations. This is very similar to quantum software, where quantum logic gates are the quantum counterparts of classical logical bit operations and are used to manipulate quantum bits, the quantum equivalent to a bit and thus, the unit of quantum information.

In order to run a quantum software on a quantum computer, the logical qubits used in the program have to be assigned to the physical qubits on the hardware. The act of assigning logical qubits in a quantum program to physical qubits on a quantum computer is called qubit mapping. It is a vital part of a quantum program's compilation as it affects how well the program will run on a quantum computer.

The qubit mapping problem is the corresponding mathematical problem that aims to minimize the changes to a quantum program needed in order to make it executable on a quantum computer. This is a crucial objective since current quantum devices have limited computing power, making it difficult to carry out long and complex computations. Improving the qubit mapping can thus significantly enhance the performance of quantum programs.

Because of its great importance, the qubit mapping problem has been studied and approached in recent years with a wide range of different techniques. This work aligns itself among these techniques, however, presents a new strategy by using a type of optimization called quadratic unconstrained binary optimization in order to improve upon existing results. The evaluation showed, that within some of the relevant benchmarks, results from this approach are similar to or better than other existing algorithms.

## Zusammenfassung

Logische Bitoperationen sind die Grundbausteine von Computer Software. Hierbei werden Bits manipuliert, um verschiedene Aufgaben auszuführen, beispielsweise bitweise AND-, OR-, XOR- und NOT-Operationen. Dies ist der Quantensoftware sehr ähnlich. Hier werden sogenannte Quantum Gates - die Quantengegenstücke von klassischen Bitoperationen - zur Manipulation von Quantenbits, der kleinsten Einheit der Quanteninformation, verwendet.

Die Zuordnung von logischen Qubits in einem Quantenprogramm zu den physischen Qubits eines Quantencomputers wird als Qubit-Mapping bezeichnet. Dieser Schritt ist Teil der Kompilierung eines Quantenprogramms und daher Voraussetzung für die Ausführbarkeit jedes Quantenprogramms auf einem Quantencomputer. Während dieses Mapping-Schritts werden zusätzliche Operationen in das Programm eingebaut, die für die Ausführbarkeit des Programms notwendig sind. Da Quantencomputer aktuell eine sehr begrenzte Rechenleistung haben, ist das Ziel so wenig zusätzliche Operationen wie möglich in das Quantenprogramm durch das Mapping einzubauen. Das Qubit-Mapping Problem beschreibt das Optimierungsproblem, welches diese notwendigen zusätzlichen Operationen minimiert. Es wird in dieser Arbeit eine neue Methode zur Lösung des Qubit-Mapping Problems vorgestellt, die auf quadratischer unbeschränkter binärer Optimierung und einem evolutionären Algorithmus basiert. Der Ansatz erzielt für eine Reihe von Quantenprogrammen signifikant bessere Lösungen zu aktuellen Mapping-Algorithmen.

# Contents

<b>Contents</b>	<b>1</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>4</b>
2.1 The basic unit of quantum information . . . . .	4
2.2 How to perform logic with quantum states . . . . .	5
2.3 Multiple qubits . . . . .	6
2.4 Quantum Program Representations . . . . .	8
2.5 Quantum Algorithms . . . . .	9
2.5.1 Shor’s Algorithm . . . . .	9
2.5.2 The Quantum Fourier Transform (QFT) . . . . .	11
2.6 Quantum Software Development . . . . .	12
2.7 Quantum Technologies in the NISQ era . . . . .	13
2.7.1 A word on limitations . . . . .	16
2.7.2 Connectivity of quantum hardware . . . . .	16
<b>3 Qubit Mapping via QUBO</b>	<b>18</b>
3.1 The Qubit Mapping Problem in general . . . . .	18
3.2 Related Work . . . . .	19
3.3 Motivation to use QUBO for qubit mapping . . . . .	20
3.4 Preliminaries . . . . .	20
3.4.1 Definitions . . . . .	20
3.4.2 Assignments and Transformation Operations . . . . .	22
3.5 The Cost Function . . . . .	26
3.5.1 Find Assignments . . . . .	27
3.5.2 Minimizing Transformation Operations . . . . .	32
3.5.3 The Final Form . . . . .	33
3.5.4 On the Problem Size and Choice of Solver . . . . .	35
3.6 Slicing with a Genetic Algorithm . . . . .	35

<b>4</b>	<b>Implementation</b>	<b>37</b>
4.1	Overview Modules . . . . .	37
4.2	Functions . . . . .	38
4.2.1	Fitness . . . . .	38
4.2.2	Validation and SWAP Count . . . . .	38
4.2.3	Genetic Operations . . . . .	39
4.3	Program Flow . . . . .	41
4.4	Additional Functions . . . . .	43
<b>5</b>	<b>Evaluation</b>	<b>45</b>
5.1	Experimental Setup . . . . .	45
5.2	Benchmark selection . . . . .	46
5.3	Results by SWAP Count . . . . .	48
5.3.1	Experiment 1 . . . . .	48
5.3.2	Experiment 2 . . . . .	51
5.4	Results by Execution Time . . . . .	54
<b>6</b>	<b>Conclusion</b>	<b>57</b>
<b>7</b>	<b>Acknowledgements</b>	<b>59</b>
<b>A</b>	<b>Quantum Fourier Transform</b>	<b>60</b>
<b>B</b>	<b>Related Work</b>	<b>62</b>
<b>C</b>	<b>Results</b>	<b>63</b>
C.1	Experiment 1 . . . . .	63
C.2	Experiment 2 . . . . .	65
	<b>Bibliography</b>	<b>66</b>

## Chapter 1

# Introduction

As quantum theory became widely accepted in the midst of the last century, physicists and computer scientists started to discuss the idea of simulating a quantum system with a classical computer. Richard Feynman for example wrote in his 1982's article, "*Can a quantum system be probabilistically simulated by a classical universal computer? If you take the computer to be the classical kind and there're no changes in any laws, and there's no hocus-pocus, the answer is certainly, No!*" [1]. Thus, a quantum computer needs to be fundamentally different from the classical one. In 1985 David Deutsch conceptualized the notion of a quantum computer, and raised the question of whether quantum computers could solve some problems faster than classical computers [2]. And in fact, a few years later, Peter Shor introduced a quantum algorithm for factoring numbers [3], which would run in polynomial time on a quantum computer, while there is no known algorithm for classical computing with this efficiency. It was the first glimpse at the potential of quantum computing and has ever since been encouraging physicists, computer scientists and engineers to put great effort in developing both quantum software and quantum hardware.

Now - almost 40 years later - the so-called Noisy Intermediate-Scale Quantum era unfolds. It refers to the current state-of-the-art quantum computers, which work well enough to produce meaningful results, even though they are suffering from restricted hardware control and errors from unwanted environmental interactions [4]. Due to the hardware's proclivity for errors, one major aspect of current quantum computing is to mitigate these faults. As a consequence a whole new research field, known as quantum error correction has emerged since 1995 [5, 6].

Another implication of the status quo is that many current quantum computers are restricted to a set of operations they can perform, referred to as their *native gate set*. Before a quantum program can be executed, the program is translated to the native gate set of the quantum hardware. This is one part of the compilation procedure of a quantum program in order to execute it on the physical qubits. Compilation in general refers to the process of adapting a quantum program from

its abstract, theoretical description, to a description only comprised of native hardware operations. The compilation process encompasses a number of distinct stages, including optimizing the quantum program prior to assigning operations, known as *gate synthesis*; the *initial placement*, which allocates the logical qubits of the program to the physical qubits of the quantum hardware; and the *routing*, which entails transferring the placed qubits as required to various locations within the hardware. Initial placement and routing are typically summarized in the concept of *qubit mapping*, on which this study will focus on.

In the recent years, different approaches to qubit mapping have been explored, ranging from permutation-based search algorithms to machine-learning based solutions [7–15]; They of course vary a lot in their strategy, e.g. exact or heuristic, where most use some kind of local based search strategy. They oftentimes use similar optimization metrics; e.g. execution time and number of changes applied to the quantum program. Most mapping algorithms target hardware constraints; some of them include noise, and error statistics of the hardware [16]. A typical mapping heuristic starts out with an initial placement, i.e. assigning the logical qubits involved in the quantum program to the physical qubits in the hardware. Then the heuristic loops through the quantum operations of the program and applies them to physical qubits according to the constraints of the quantum hardware.

The introduced method aligns itself among these techniques, however, it utilizes in contrast to many of the mentioned works a non-local view of the problem. In other words, many existing works rely on local search strategies, while this study focuses on a holistic view of the quantum program and its requirements. This is especially beneficial when it comes to inaccuracies stemming from too restricted search windows. The latter is not a limitation for the used strategy, since it captures the quantum program and its requirements in one piece. The contributions can be summarized as follows:

- **Algorithm:** mapping procedure that considers hardware connectivity constraints to reduce the total execution time and thus enhancing the quality of the output state.
- **Holistic strategy:** instead of piecewise iterating over gates, the quantum program's requirements are captured as a whole by a special kind of optimization formulation, termed Quadratic Unconstrained Binary Optimization. Further, a simple evolutionary algorithm is utilized to optimize how to the quantum program is best prepared for the mapping.
- **Accessibility:** The introduced work provides an open-source alternative to existing algorithms and is accessible through Github [17].
- **Evaluation:** The results from this approach are compared to relevant existing algorithms and show comparable or better results for a set of quantum

programs that can be run on close-to-mid-term quantum computers.

One could ask, why we choose QUBO as our dedicated problem formulation: QUBO in itself presents an elegant approach towards qubit mapping, firstly, because it can represent the problem exactly, while it can be solved heuristically. This gives rise to two advantages: a) since new and better matrix solving strategies are developed continuously, the method can always be dynamically adapted with the best currently available solving algorithm without changing the problem formulation, and b) the exact formulation gives a lower bound to the problem, meaning that if it is solved exactly, it will always give the best possible solution. Secondly, QUBO is especially interesting for the quantum computing community, because it can not only be solved by classical heuristics, but also with adiabatic quantum computation [18, 19]. Since the construction of the first quantum adiabatic computer (also called quantum annealer; built by D-Wave Systems Inc. in the early 2000s) a lot of effort has been made to formulate QUBOs for different NP-hard optimization problems, ranging from graph isomorphism [20], Traffic Flow Optimization [21] to training machine-learning models [22]<sup>1</sup>. Lastly, there have been only rudimentary experiments with using QUBO for qubit mapping [24], which makes it an interesting research question, hence there is no proof-of-concept yet.

This work presents an optimization formulation of the qubit mapping problem using QUBO, and an evolutionary algorithm is utilized to optimize the mapping in an iterative manner. The thesis is organized as follows: Chapter 2 introduces some fundamentals of quantum computing, gives insights into two famous quantum algorithms and discusses current quantum technologies. Chapter 3 describes the qubit mapping problem in its general form, lists related work and outlines the introduced approach in detail. Chapter 4 explains the implementation in both pseudocode listings and flow diagrams. Finally, Chapter 5, presents the results obtained and discusses the comparisons to two state-of-the-art mapping algorithms; Chapter 6 concludes the work with discussion for possible future tasks.

---

<sup>1</sup>Even though the computational solving ability of these devices are restricted to small problem sizes still, the fast development of these systems gives hope to expect that more realistic problems sizes can be addressed in the near to mid term future [23].



## Chapter 2

# Background

This chapter presents some background on quantum computing following fundamental literature [25]. After delving into the functionality of two well-known quantum algorithms, Shor's Algorithm and the quantum Fourier transform, the chapter explains the concept of a quantum *stack* and a selection of frameworks available for software development in quantum computing. The chapter closes with capabilities and limitations of various current quantum technologies.

## 2.1 The basic unit of quantum information

The concept of a classical bit is distinct from that of a quantum bit, or qubit in short. Unlike classical bits which are consistently in a definite state of either 0 or 1 throughout a computation, qubits are only definite when measured, at which point they collapse into either state. At all other times, a qubit is described by a more complex state that cannot be captured by binary values, due to the subatomic nature of the physical system it represents; which follows the principles of quantum mechanics. This allows a qubit to exist in multiple states simultaneously, referred to as *superposition*, which can be described using the mathematical framework of a vector space, typically represented as the vectors  $|0\rangle$  and  $|1\rangle$ , i.e. the low and high state, respectively. These basis vectors span the vector space, where all possible superpositions of the basis states live.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (2.1)$$

All superpositions  $|\psi\rangle$  are then described as a linear combination of both states, reading

$$|\psi\rangle = a|0\rangle + b|1\rangle = a \begin{bmatrix} 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}, \text{ where } a, b \in \mathbb{C}. \quad (2.2)$$

The probability to measure a qubit being in the  $|0\rangle$  or  $|1\rangle$  is  $|a|^2$  or  $|b|^2$ , respectively. Since a qubit has to decide for either  $|0\rangle$  or  $|1\rangle$ , the probability to measure  $|0\rangle$  or  $|1\rangle$  is 100%; as a consequence the coefficients must add up to 1, i.e.  $|a|^2 + |b|^2 = 1$ . A common way to display the state of a single qubit is the so called *Bloch sphere* depicted in 2.1.

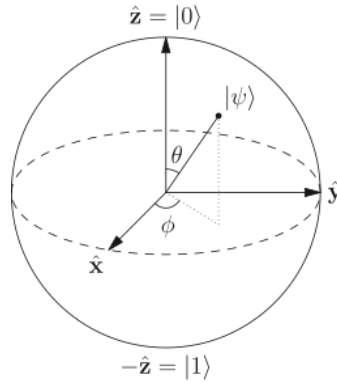


Figure 2.1: The Bloch sphere for representing single quantum states [26].

The two basis states lie on  $\pm z$  axes and all possible superpositions lie in between on the complex plane described by the coefficients  $a = \cos \theta/2$  and  $b = e^{i\phi} \sin \theta/2$ , where  $\phi$  and  $\theta$  are real values (also called *phases*).

**Example 2.1.1.** *Let us assume some arbitrary state of a qubit, where  $\theta = \pi/2$  and  $\phi = 0$ .*

$$|x\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle. \quad (2.3)$$

*Then the probability to measure the qubit in  $|0\rangle$  is the same as to be in  $|1\rangle$  state, i.e.  $|a|^2 = |b|^2 = 0.5$ . This makes perfect sense, since the vector points in the  $+x$  direction, to the dashed boundary line in Fig. 2.1 which lies right in the middle of the two basis states.*

## 2.2 How to perform logic with quantum states

A *quantum gate* is a unitary operation which can be applied to one or more qubits. A unitary operation  $U$  is defined as  $U^\dagger U = \mathbb{I}$ , where  $U^\dagger$  is the complex conjugated and transposed version of  $U$ .

Gates are typically described by matrices; the general single-qubit gate is called *U-gate*, which reads

$$U(\theta, \lambda, \phi) := \begin{bmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda} \sin(\frac{\theta}{2}) \\ e^{i\phi} \sin(\frac{\theta}{2}) & e^{i(\phi+\lambda)} \cos(\frac{\theta}{2}) \end{bmatrix}.$$

Configurations of the phase variables  $\theta, \lambda, \phi$ , which are commonly used, have their own name; a selection of the most prominent single-qubit gates is listed in Table 2.1.

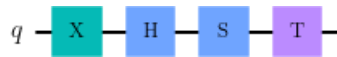
Pauli X	Pauli Y	Pauli Z	Hadamard	S gate	Phase Shift
$q$ — <b>X</b> —	$q$ — <b>Y</b> —	$q$ — <b>Z</b> —	$q$ — <b>H</b> —	$q$ — <b>S</b> —	$q$ — <b>T</b> —
$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$
$\pi$ rotation around x-axis	$\pi$ rotation around y-axis	$\pi$ rotation around z-axis	$\pi$ rotation around $[1,0,1]$ <sup>1</sup> -axis	$\pi/2$ rotation around z-axis	$\pi/4$ rotation around z-axis

Table 2.1: Most relevant single-qubit gates.

<sup>1</sup>the axis between the x and z

Let us consider a simple sequence of gates applied to one qubit.

**Example 2.2.1.** *Applying the following gates*



to a qubit changes its initial  $|0\rangle$  state as shown in Figure 2.2.

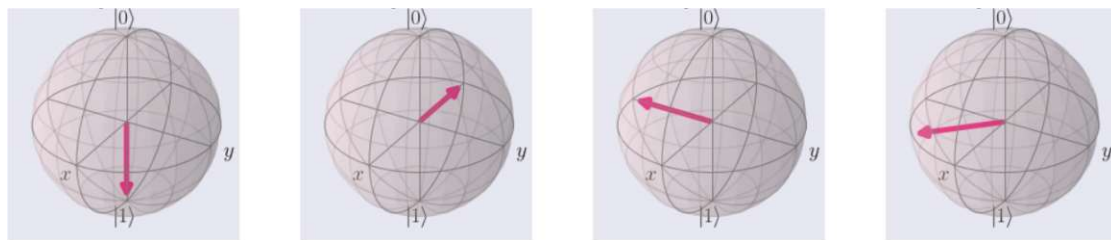


Figure 2.2: Transformations of qubit state starting from the basis state  $|0\rangle$ .

## 2.3 Multiple qubits

We will now look at how we represent multiple qubits, and how these qubits can interact with each other. Let us start with the simplest case: two qubits

live in a space spanned by four well-defined states  $|00\rangle, |01\rangle, |10\rangle$  and  $|11\rangle$ , i.e. four orthogonal vectors are needed. This is mathematically realized with a four-dimensional vector space, e.g. with the eigenvectors

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

and using a linear combination for all possible superpositions

$$|\phi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle,$$

where  $a, b, c, d \in \mathbb{C}$ .

The dimension of the vector space thus grows exponentially with the system size, i.e. for  $n$  qubits the dimension is  $2^n$ . As such do the gates applied to multiple-qubit states, i.e. a unitary operation  $U$  applied to  $n$  qubits has the size  $2^n \times 2^n$ .

Hence, for two qubits, the matrix representation needs to be a  $4 \times 4$  unitary matrix; a list of the most relevant two-qubit gates is given in Table 2.3.

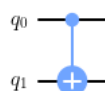
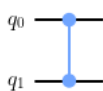
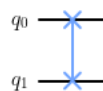
CNOT	CZ	SWAP
		
$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
<p><math>\pi</math> rotation around <math>x</math> on the target qubit <math>q_1</math> if the control qubit is in the state <math> 1\rangle</math>, and leaves it unchanged otherwise.</p>	<p><math>\pi</math> rotation around <math>z</math> on the target qubit <math>q_1</math> if the control qubit is in the state <math> 1\rangle</math>, and leaves it unchanged otherwise.</p>	<p><math>q_0</math> receives the state of <math>q_1</math> and vice versa.</p>

Table 2.2: Most relevant two-qubit gates.

As already mentioned, a native gate is one that can be performed using the basic control and manipulation operations available on the quantum hardware. These native gates build up more complex gates, i.e. multi-qubit gates involving

three or more qubits. The most prominent three-qubit gate with its own name is the CCX or Toffoli gate, it is represented by a  $8 \times 8$  matrix, which is composed of several one-qubit T- and Hadamard-gates, and six CNOT-gates; it reads

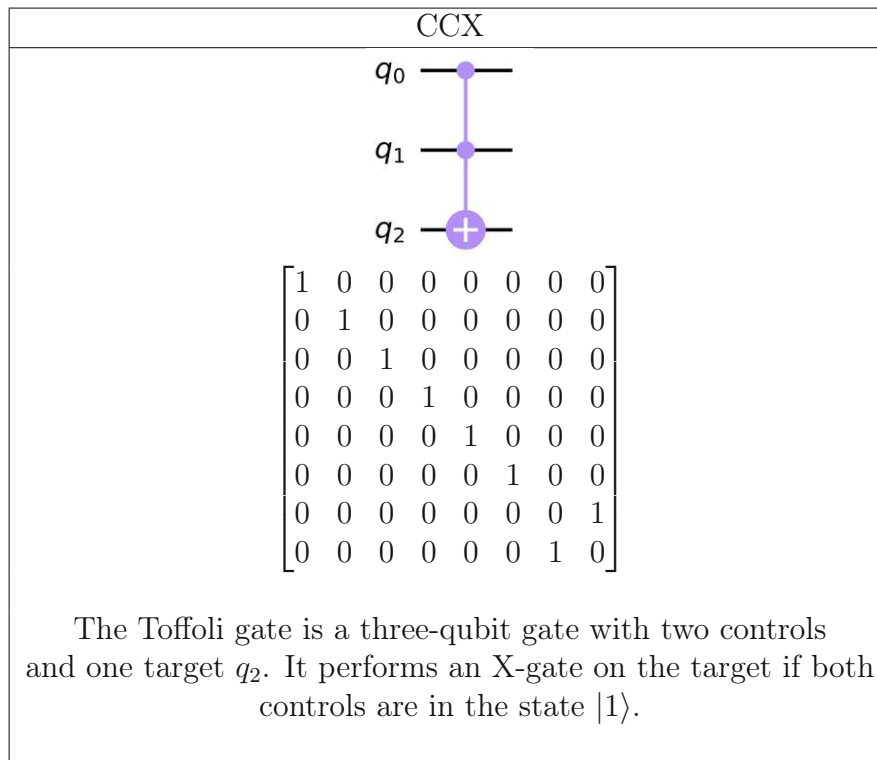


Table 2.3: The most relevant three-qubit gate.

## 2.4 Quantum Program Representations

A *quantum program*, also called *quantum circuit* describes a sequence of gates applied to logical qubits. One typical description is graphical, where quantum gates are represented by the boxed symbols and qubits are represented by lines as listed above; Figure 2.3 shows an example circuit in its graphical representation.

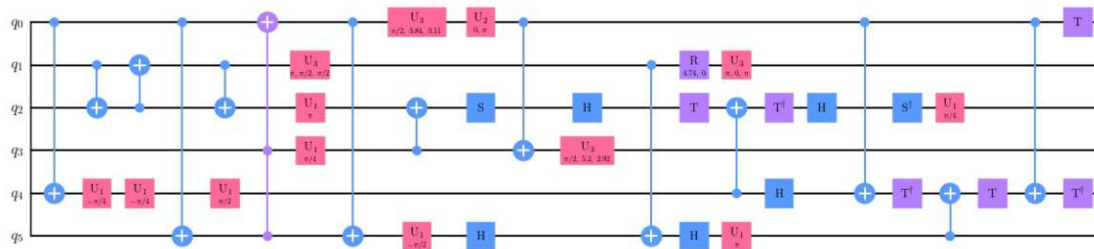


Figure 2.3: Graphical representation of a quantum circuit; The horizontal lines represent the time axes for their respective qubits. The colored boxes and vertically linked circles mark the gates. Some gates can be executed in parallel, when they do not involve the same qubits. The number of time steps in total is called the *depth* of a quantum program. (The graphic was generated in Python.)

Similar to classical computing, it is practical to define a *universal set of gates*. It is a set of quantum logic gates that can be used to perform any unitary operation on one or more qubits by combining them in different (but finite) ways. A set example is the Pauli gates (X, Y, Z), plus the phase shift S gate, plus the CNOT gate [27].

Another important concept is the *depth* of a quantum circuit; it refers to the number of time-steps of a quantum circuit, assuming one time-step always covers all gates that can be executed in parallel.

## 2.5 Quantum Algorithms

A crucial research endeavour of quantum computing is to discover quantum algorithms to solve classically hard problems, which have equivalent or reduced time complexity in comparison to their classical counterparts. This section will delve into Shor's famous algorithm and the quantum Fourier transform, showing the elegance of quantum logic and how beneficial they can be when applied to the appropriate problems.

### 2.5.1 Shor's Algorithm

Currently, computer networks rely on RSA encryption to keep communication over the internet safe. Simply put, the method uses a large<sup>1</sup> number  $N$ , which is semi-prime, i.e. the result of multiplying two prime numbers. The encryption is based on the phenomenon that it is computationally infeasible (non-polynomial for classical computers) to find the prime factors of  $N$ .

<sup>1</sup>The size of the semi-prime number  $N$  is 2048 bits for most systems (612 decimal digits) [28].

Shor's Algorithm, however, finds them in polynomial time [3] and on a big enough quantum computer, it would pose a real threat to RSA encryption.

The overall goal is to find the factors  $p, q$  of  $N$ . It is however actually enough to find a number that shares factors with a multiple of  $N$ , i.e.  $m \cdot N = p \cdot q$ , where  $m \in \mathbb{N}_+$ , since as soon as a shared factor is found, one can find the greatest common divisor of  $N$  and the found number (with the 2000 year old Euclidean Algorithm) and divide the latter by the divisor to receive the actual factor. It is also known, that for some even number  $p$  and random integer  $g$ ,  $g^p$  will *always* for some  $p$  result in  $m \cdot N + 1$ . Ergo, the objective turns into finding  $(g^{p/2} \pm 1)$ , which share factors with  $m \cdot N$ . The crucial quantum part unfolds with the so called *period*  $p$ . In classical computing one would have to try for a long time guessing the right  $p$ . With the quantum Fourier transform this period can be found with only one calculation. This is achieved by combining many different choices for  $p$  via a quantum superposition of states representing the random integer  $g$  to the power of  $p$ , i.e.  $|p_1, r_1\rangle + |p_2, r_2\rangle + |p_3, r_3\rangle \dots + |p_n, r_n\rangle$ , where  $r$  is the remainder of the calculation  $g^p = m \cdot N + r$ . If we then measure the remainder part of the superposition and the outcome happens to be e.g.  $r_3$ , then the  $p$  part of the superposition will only comprise the elements which also had the remainder of  $r_i = r_3$ . Since we know from pure mathematics, that then there is a repeating property between the periods left in the superposition<sup>2</sup>, we can use the quantum Fourier transform on the superposition to determine the period  $p$ .

**Example 2.5.1.** *Let us assume the measurement of the remainder gave  $r = 2$  and  $r_3 = r_7 = r_{86} = 2$ , then superposition after measuring the out the remainder part would be  $|p_3\rangle + |p_7\rangle + |p_{86}\rangle$ . Since we know from pure mathematics, that then there is a repeating property between the periods, namely  $p_3, p_7$  and  $p_{86}$  are then some multiple of  $p$  apart from each other, we can use the quantum Fourier transform on the superposition to determine the period  $p$ .*

Here is a high level overview of the steps of Shor's algorithm:

1. Choose a random integer  $g < N$ .
2. Generate a superposition of as many periods as possible (restricted by the numbers of qubits in the quantum computer at hand) and measure it.
3. Use quantum Fourier transform on the superposition generated by the measurement in step 2. to find the period  $p$ .
4. Use  $p$  to determine the factors  $g^p = m \cdot N + 1$ , which share factors with  $N$  and determine the actual factors  $p, q$  with the Euclidean Algorithm.

---

<sup>2</sup> $g^x - m_1 \cdot N = g^{x+p} - m_2 \cdot N = g^{x+2p} - m_3 \cdot N = \dots = g^{x+np} - m_n \cdot N = r$ , where  $m_i \in \mathbb{N}_+$

**Will RSA technology soon be broken?** No. According to quantum cryptanalysis from 2021 [29] resource capabilities to decipher the RSA-2048 below 24 hours would need 8,194 logical qubits, which translates to 8.7 Mio. physical qubits (reminder: most promising current technologies comprise around 200 physical qubits). Another more recent survey of the Global Risk Institute [30] asked 50 experts in the field (from the USA, Canada, EU, China and Australia) to give their estimates when quantum technology will be at this level. On average those experts find it unlikely (below 30% chance) that the necessary capabilities will be reached within the next 10 years; however, within the next 20 and 30 years the experts reasoned a likely-hood of 70% and 96%, respectively on average to accomplish this technological stage.

## 2.5.2 The Quantum Fourier Transform (QFT)

As we saw, a very important ingredient to Shor's algorithm is the QFT. Classically, a Fourier transform reveals the frequency components of a time-dependent function. QFT achieves the same within polynomial time. It takes a general quantum state  $|x\rangle$  and transforms it into its Fourier basis  $|y\rangle$ , which holds the information of the states frequency components. Given two quantum states  $|x\rangle$  and  $|y\rangle$  for  $N$  qubits, this mathematically reads

$$|x\rangle = \sum_{j=0}^{N-1} x_j |j\rangle \quad |y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle \quad (2.4)$$

$$\text{where } y_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{i2\pi nk/N}, \quad (2.5)$$

where Eq. 2.5 is just the formula for the classical inverse Fourier transform with the imaginary unit  $i$ . To keep things (relatively) simple, we set  $N = 2^n$ , where  $n \in \mathbb{N}_+$  and assume  $|x\rangle$  is a basis state  $|x = j\rangle$  for some  $j \in [0, N - 1]$ ; then

$$y_k = \frac{1}{\sqrt{N}} e^{i2\pi xk/N}. \quad (2.6)$$

The goal is to find a practical unitary transformation  $U$ , yielding

$$U |x\rangle = |y\rangle. \quad (2.7)$$

Applying the necessary algebraic transformations to Eq. 2.4 and Eq. 2.6 detailed in Appendix A leads to the the following expression

$$U |x\rangle = \frac{1}{\sqrt{2^n}} (|0\rangle + e^{i2\pi x2^{-1}} |1\rangle) \otimes (|0\rangle + e^{i2\pi x2^{-2}} |1\rangle) \otimes \dots \otimes (|0\rangle + e^{i2\pi x2^{-n}} |1\rangle). \quad (2.8)$$



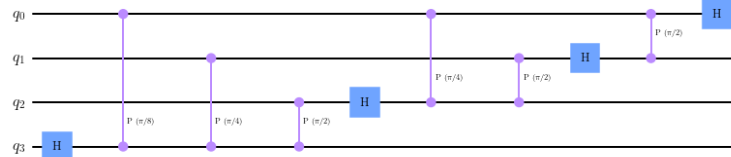


Figure 2.4: QFT for a four qubit quantum state.

where Eq. 2.8 already reveals the quantum program; it can be computed with Hadamard gates and controlled rotations. The time complexity to do this is polynomial, more precisely it is  $\mathcal{O}(N \log^2 N)$  [31].

**Example 2.5.2.** *Figure 2.4 shows the QFT for a 4 qubit system state.*

## 2.6 Quantum Software Development

Similar to programming for classical computation, one can develop quantum software at assembly language level (e.g. OpenQL) or on some higher level, in a oftentimes more user friendly software development kit (e.g. QISKit SDK). This software is then compiled / translated to the quantum hardware of use. The typical stack of a quantum computer is listed in Table 2.4. A step downwards from the SDK level lies the compiler layer, which includes circuit synthesis [32] and qubit mapping. The next lower level is where quantum error correction is implemented. This results in the quantum program reaching the lowest level of the stack, where the instructions contain error-corrected operations that can be executed by the quantum device's native controls.

In the last two decades different software has been developed at each of these abstraction layers. As real quantum hardware is however still scarcely available, quantum software simulators [33, 34] are commonly used to validate and test the functionality of the upper layers, as for quantum algorithms, compilation and quantum error correction. A selection of relevant programming frameworks that are available for writing and simulating quantum programs are listed below

- OpenQL (Open Quantum Programming Language) is an open-source quantum programming framework that aims to provide a high-level programming interface for quantum computing. OpenQL allows developers to write quantum programs using a C-like syntax and then translates these programs into a machine-specific format that can be executed on a variety of quantum architectures [35].

- Quipper is a functional programming language for quantum computing developed by the University of Edinburgh. Quipper is designed to be used with the Glasgow Haskell Compiler [36].
- QISKit (pronounced "quiskit") is an open-source quantum computing software kit developed by IBM. QISKit allows users to write quantum circuits and programs using Python, and it includes tools for simulating quantum circuits and running quantum programs on real quantum hardware [37].
- Q# (pronounced "Q sharp") is a programming language developed by Microsoft for writing quantum programs. Q# is designed to be used with the Quantum Development Kit of Microsoft [38].
- Cirq provides a Python software library for writing, manipulating, and optimizing quantum circuits, and then running them on quantum computers and quantum simulator, developed by Google. [39].

The list is surely not exhaustive, since there are many other languages and frameworks that have been developed for this purpose [40], and the field is still rapidly growing. Also, the described stack picture is simplified and a lot of aspects to every layer have not been described; for further reading on the topic please refer to [25, 41].

SDK: code abstract quantum program
Compilation and Optimization: Gate synthesis and translation to native gate set
Quantum Error Correction: correct errors while executing
Quantum chip: physical operation and control

Table 2.4: Typical (simplified) stack of a quantum computer [42].

## 2.7 Quantum Technologies in the NISQ era

The previous section elaborated on quantum logic and software and how it is finally executed on a quantum device. In this chapter quantum devices themselves and their different technological approaches will be outlined. Current Quantum technologies are called *NISQ* computers [43]. NISQ stands for "Noisy Intermediate-Scale Quantum". It refers to a moderate number of qubits (typically between 5 and 100) that can perform some (10-20) quantum operations [44]. Their main purpose is to demonstrate the capabilities of quantum computing, i.e. performing tasks that are difficult or impossible for classical computers, such as simulating quantum systems, solving selected optimization problems and quantum algorithms,

which require a small numbers of qubits. Different promising approaches have been developed in the past decade; A selection of promising technologies and their functionality is given below with a summary of their characteristics in Table 2.5.

- **Superconducting qubits** are typically made from thin films of superconducting materials, patterned as tiny wire-loops on a silicon wafer. The quantum state of the qubit is encoded in the tiny flow of current through this loop. The energy levels of the qubit can be manipulated by applying electromagnetic radiation (such as microwaves) to the loop, which causes transitions between the energy levels. The energy levels of the qubit can also be controlled by applying a magnetic flux through the loop. Superconducting qubits are considered as one of the best-developed physical current qubit systems [45, 46].
- **Trapped ions** are qubits held in place via electromagnetic fields (using lasers). Calcium ions have been used extensively in this research, due to their relatively simple electronic structure and the availability of efficient laser cooling techniques. The accuracy of the manipulation and readout are dependent on the laser-ion interaction rates and the quality of the laser beams. Next to superconducting qubits this technology is evaluated to be one of the most promising approaches [47].
- **Spin qubits** are based on the spin of an electron or the nuclei of certain atoms, which are controlled by electric fields. They are fabricated in solids (e.g. silicon or gallium, using existing semiconductor technology or diamonds). Typically, spin qubits are more susceptible to decoherence due to the interactions with the environment, such as the interaction with nuclear spins or phonons (vibrations of the substrate material) [48]. However, the negatively charged **Nitrogen-Vacancy (NV) center** in diamonds is one of the most promising solid-state qubit platforms in this regard. It uses diamond defects to add nitrogen atoms in order to capture electrons. These qubits are controlled with optics and can operate at room temperature. [49]
- **Adiabatic quantum computation** is a quantum optimization method that uses quantum fluctuations to search for the global minimum of the physical system, which encodes the solution landscape. In contrast to the other aforementioned technologies, this technology is not a representative of the quantum gate model. D-Wave Systems is the leading company on this field and their quantum annealers consist of more than thousands of qubits [50].
- **Topological qubits** are realized via quasiparticles. They can exist in certain materials, such as semiconductors. These tiny particles create multiple

possible states of lowest energy in the physical system, which can be used to store quantum information. These states are spread out across the system in a way that is not limited to a specific location (non-local). This is important because it means that the quantum information is more resistant to so called *local noise* which is noise that is limited to a specific location. By encoding the information non-locally, it is protected from this type of noise. Even though the "inherently error corrected" nature of topological qubits is promising, the technology has not been demonstrated yet [51].

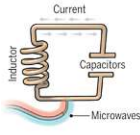
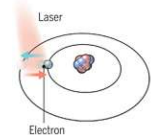
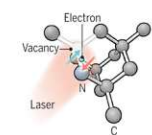
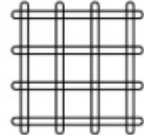
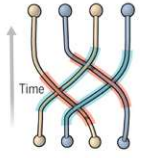
	Super-conducting	Trapped Ion	Nitrogen Vacancy	Quantum adiabatic computation	Topological
# Qubits	50-100	50	5	2000	? <sup>1</sup>
Lifetime	0.001s	1000s	100s	100 $\mu$ s	?
C-NOT Fidelity	99.7%	99.9%	99.2%	- <sup>2</sup>	100% (error free)
C-Not Time	60ns	200 $\mu$ s	1ns	-	?
Connectivity	Nearest neighbor	All-to-all	Nearest neighbor	All-to-all	?
Modality	Gate model	Gate model	Gate model	Adiabatic	Gate model
Difficulties	Operation at low Temp (mK)	Complex optics and slow operation	Difficult control	Special purpose solver	Hard to realize, states necessary not found yet
Symbolic Image					
Companies	Google, IBM, Rigetti	IonQ, AlpineQT, Honeywell	QuTech, Intel, Quantum Brilliance	D-Wave, Google	Microsoft, BellLabs

Table 2.5: Different quantum technologies; Data from [45, 51–55], Graphics from C. Bickel/Science and [56].

<sup>1</sup>No success of realization yet.

<sup>2</sup>Adiabatic quantum computing is not gate based.

The list of selected qubit technologies is for sure not exhaustive, and there are many other developments in the field of quantum technology that are also showing promise, e.g. photonic qubits [57], electron spin qubits [58] or neutral atoms [59]. Also, the field is dynamic, and new technologies are being developed and explored every year [60, 61].

### 2.7.1 A word on limitations

Despite comprehensive research in the last decades, quantum hardware is still restricted and computations are fragile [44]. In particular *decoherence*, *noise* and *fidelity* are three of the main limiting factors. Decoherence refers to the loss of coherence (=the separation of the quantum state from its classical state). It is caused by interactions with the environment, and it can make it difficult to maintain the quantum state long enough to perform useful computations. Another hurdle is noise, it summarizes different types of unwanted disturbances in a quantum system that can cause errors in quantum computations; i.e. the control system, and the quantum hardware itself. Controlling and manipulating quantum systems is also a task to be improved; It requires precise control over the parameters of the system, such as the energy levels, and the interactions between the qubits. It requires the ability to read out the state of the qubits, and to perform operations on them with high precision, which is referred to as *fidelity*. According to a recent study [44] current quantum devices can realistically execute quantum programs with  $n$  logical qubits where

$$n \cdot \text{depth} \ll \frac{1}{\text{error} - \text{rate}}. \quad (2.9)$$

and this will hold for the next five to ten years.

**Example 2.7.1.** *Thus a typical quantum device with error-rates of around  $10^{-3}$ , the size of a quantum circuit is restricted to being much smaller than  $\text{depth}=50$ , given a medium sized quantum hardware of 20 qubits.*

Lastly, quantum computing is expensive; both in terms of energy and monetary costs. Some technologies like superconducting qubits and trapped ions need to be operated at low temperatures (mK), for which energy expensive cryostats are necessary. This in turn translates to actual monetary cost [62] - quantum hardware is currently still expensive to build and operate, even though it is difficult to find precise numbers and figures in this regard, since energy efficiency of quantum computing has not been thoroughly addressed yet [63].

### 2.7.2 Connectivity of quantum hardware

As we saw in the previous chapters, quantum technologies differ a lot and so do their connectivity constraints. Connectivity describes the ability of the qubits in the system to interact and communicate with one another. It is commonly represented by a so called *coupling graph*, where the physical qubits form the nodes of the graph and the edges mark the ability to perform an operation. There are different types of connectivity, such as distance-based or programmable connectivity,

where the physical qubit interactions are introduced during the course of an execution. This work focuses on fixed connectivity, which is the most common in all the above mentioned technologies, where the qubits are connected in specific patterns, for example a linear chain or a two-dimensional grid; i.e. nearest neighbour interaction capability. Further, for some technologies it is only possible to apply two-qubit gates in one direction even though two physical qubits are connected on the chip, this is called asymmetric connection. The majority of technologies however, e.g. all currently built superconducting processors, allow symmetric application of gates [8]. For this reason, the work will exclusively focus on the latest symmetric coupling model with nearest neighbour interaction.

## Chapter 3

# Qubit Mapping via QUBO

In the following sections the qubit mapping problem will be defined and how it has been approached by other authors. It will further motivate and introduce this work's method to do so.

### 3.1 The Qubit Mapping Problem in general

Qubit mapping is the process of assigning the logical qubits in a quantum program to the physical qubits on a quantum computer, while adhering to the connectivity constraints (Sec. 2.7.2). It is one of the main tasks of a quantum compiler and the corresponding mathematical problem is proven to be NP-complete [64]. In its most general form the definition reads:

**Definition 3.1.1** (Qubit Mapping Problem). Given a quantum program with  $n$  qubits and a target architecture with  $n'$  qubits, where  $n \leq n'$ , find a function that

1. assigns each qubit in the program to a qubit in the target architecture at each unit time of the execution and
2. inserts the minimum number of transformation operations such that the resulting program can be executed using the connectivity available in the target architecture.

Simply put, the goal is to minimize transformations to the quantum program required to execute the program on a given quantum hardware. The objective is to reduce execution overhead in terms of transformation operations and runtime, as the states of the physical qubits in a quantum computer are fragile and deteriorate over time and this is naturally coupled to the success of their operations as described in Sec. 2.7.1.

**Definition 3.1.2** (Transformation Operation). A transformation operation in quantum computing is a quantum operation, which transfers the state of a set of physical qubits to another set of physical qubits in the quantum architecture.

Some relevant examples are the familiar SWAP-gate, the MOVE-gate [9], teleportation [65] and emulating gates such as Bridge or Reversal [14].

*Remark.* As many other mapping methods, the introduced approach will exclusively focus on using SWAP operations to solve the qubit mapping problem, since it is composed of CX-gates, which is native to the majority of nowadays quantum technologies.

## 3.2 Related Work

In recent years, different approaches to qubit mapping have been explored, ranging from permutation-based search algorithms to machine-learning based solutions [8–11, 13, 14, 66]; They of course vary in their strategy, e.g. heuristic or exact, but also in their optimization metrics, i.e. whether they focus on minimizing the number of transformations or the number of time steps. In addition to hardware connectivity some mapping methods include noise, and error statistics of the hardware [16]. There have been attempts to tackle asymmetric coupling constraints (Sec. 2.7.2) via transformation operations such as Bridges or Reversal e.g. [67]. Most solutions however assume symmetric connectivity and therefore apply exclusively SWAP gates to adapt the quantum program. Two promising heuristic-mapping methods of common use are *SABRE* a SWAP-based Bidirectional heuristic-search algorithm [8] and *qmap*, another search heuristic which uses look-ahead and an empty initial placement (which is then developed in the course of the algorithm) [10]; Both are fully integrated with IBM’s Quantum SDK<sup>2</sup>. A comprehensive overview of relevant mapping methods and their specifications is given in Appendix B. The majority of these approaches start out with an initial placement, i.e. assigning the logical qubits involved in the quantum program to the physical qubits in the hardware. Then the heuristic loops through the sequence of gates of the program and applies the gates to physical qubits according to the constraints of the quantum hardware and inserts transformation operations when needed.

The introduced method aligns itself among these techniques, however, it utilizes in contrast to many of the mentioned works a holistic view of the problem. In other words, many existing works rely on local search strategies, while this study focuses on a holistic view of the qubit mapping problem. This means our formulation is not prone to inaccuracies stemming from too restricted search windows, but captures the quantum program and its requirements in one piece.

---

<sup>2</sup><https://qiskit.org>, 06.01.2023



### 3.3 Motivation to use QUBO for qubit mapping

This thesis introduces a new strategy: instead of iterating over gates, the problem is captured as a whole by a special kind of optimization formulation, which can then be solved with different standard optimization methods (e.g. simulated annealing, tabu search, etc.). This optimization formulation is called quadratic unconstrained binary optimization, or QUBO in short.

QUBO presents an interesting type of optimization formulation for quantum computing, since it cannot only be solved by classical heuristics but also with adiabatic quantum computing [18, 19]. Since the construction of the first adiabatic quantum computer (built by D-Wave Systems Inc. in the early 2000s, also called quantum annealer) a lot of effort has been made to formulate QUBOs for different NP-hard optimization problems, ranging from graph isomorphism [20], Traffic Flow Optimization [21] to training machine-learning models [22]. Even though the computational solving ability of these devices are restricted to small problem sizes still, the fast development of these systems gives hope to expect that more realistic problem sizes can be addressed in the near to mid-term future [23].

Furthermore, QUBO can be formulated in an exact manner, and is then solved with some standard exact or heuristic solving method. This gives rise to two advantages: a) since new and better solving methods are developed every year, the method can always be dynamically adapted with the best currently available algorithm, and b) the exact formulation gives a lower bound to the problem, meaning that if it is solved exactly, it will always give the best possible solution.

Lastly, there have been only rudimentary experiments with using QUBO for qubit mapping [24], which makes it an especially interesting research question, since there is no proof-of-concept yet.

### 3.4 Preliminaries

The following definitions and the concepts of assignments and transformation operations will be used to explain the details of the introduced approach

#### 3.4.1 Definitions

**Definition 3.4.1** (Coupling Graph). The coupling graph is a connected graph that represents the connectivity of the physical qubits in a quantum device, which can in general be symmetric or asymmetric. As previously described in Sec. 2.7.2 most technologies allow for a symmetric application of two-qubit gates, i.e. CX-gates in both directions are possible, which is represented by an undirected graph  $G'(V', E')$ . We will therefore use the symmetric coupling model for this study, as

the majority of other references do, too. The physical qubits make up the set of nodes  $V'$  in the graph and possible interactions in both directions between physical qubits form the set of edges  $E'$ .

**Definition 3.4.2** (Slice). A slice or time slice  $k$  refers to an arbitrary sequence of gates cut from the quantum program; The slices form a partition of the quantum program, where every gate belongs to exactly one slice.

**Example 3.4.1** (Slice). A time-sliced quantum program is given in Figure 3.1.

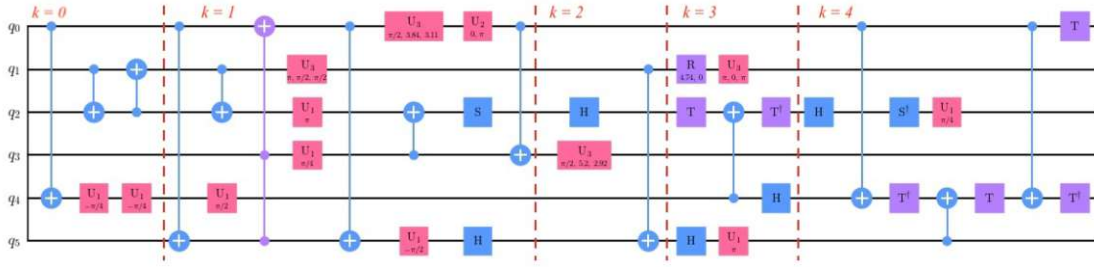


Figure 3.1: Example partition of quantum circuit with five time slices.

**Definition 3.4.3** (Interaction Graph). The interaction graph  $G_k(V, E_k)$  of a time slice  $k$  is in general a directed unconnected graph, which represents the qubits as nodes  $V_k$  of the quantum program and the sequence of two-qubit gates as directed edges  $E_k$ . Since a gate can be executed on a coupling graph edge regardless of its direction, it is sufficient for the mapping problem to perceive it as an undirected graph, depicted in a) of Fig. 3.2.

**Definition 3.4.4** (Assignment or Subgraph Isomorphism). An assignment is a subgraph-isomorphic mapping: it refers to an injective function  $f_k : V \rightarrow V'$  of an interaction graph  $G_k(V, E_k)$  to the coupling graph  $G'(V', E')$  of a time slice  $k$ , i.e.  $f_k$  assigns the set of logical qubits in the interaction graph  $V$  to a subset of the physical qubits in the coupling graph  $V'$ , such that all edges in  $E_k$  are preserved in  $E'$ .

**Definition 3.4.5** (Non-Assignment). A non-assignment is every mapping from the an interaction graph to the coupling graph violating the constraints of a subgraph isomorphism.

**Example 3.4.2** (Assignment to coupling graph). An assignment and a non-assignment is depicted in b) of Fig. 3.2. In case of an assignment all gates happening in the time slice are represented by an edge in the coupling graph.

**Definition 3.4.6** (Initial Placement). The initial placement refers to the assignment of time slice  $k = 0$ .

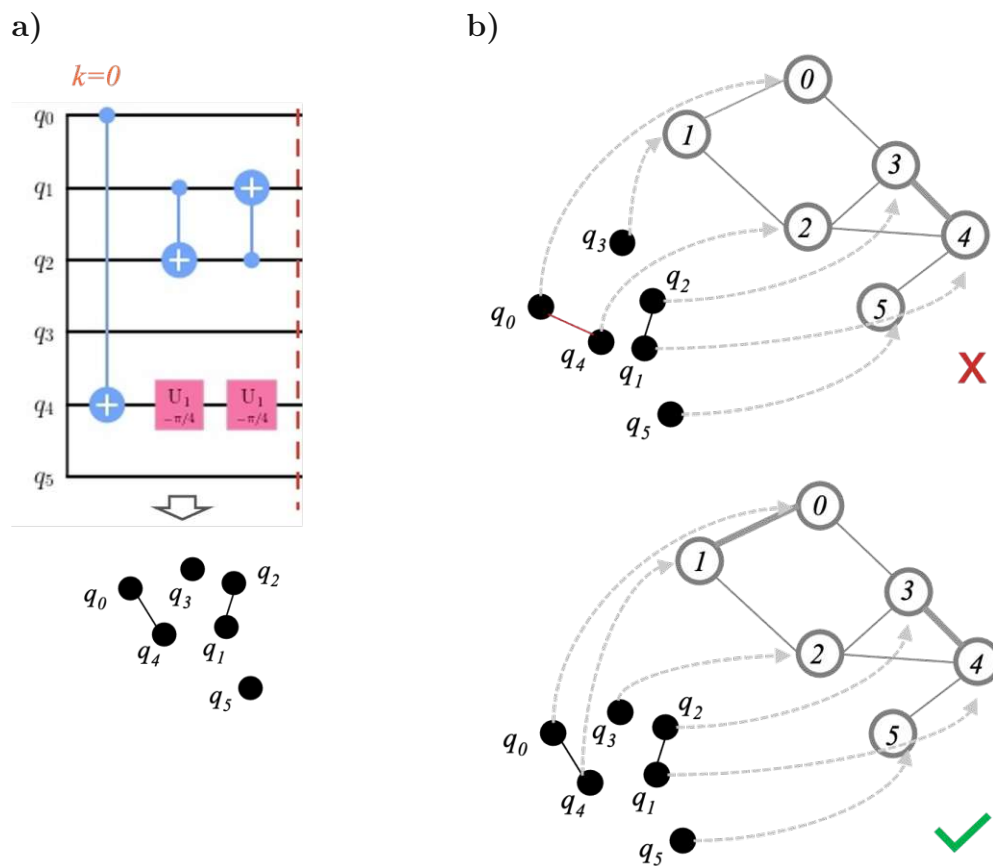


Figure 3.2: **a)** Shows the interaction graph of a quantum program’s time slice  $k = 0$  (Ex. 3.4.1) where qubits are presented as black circles and gates between logical qubits are represented by solid black edges; **b)** pictures a non-assignment on the top and an assignment on the bottom; the interaction graph drawn in black is mapped to the coupling graph in grey/white where solid grey edges mark the connectivity of the coupling graph and the bold edges signify where at least one gate is applied. The top scenario sketches a non-assignment since the edge  $(q_0, q_4)$  has no representation in the coupling graph; the lower scenario pictures an assignment, i.e. all logical qubits and edges in the interaction graph have their representative in the coupling graph.

### 3.4.2 Assignments and Transformation Operations

Given a quantum circuit sliced in time, the objective is to find an assignment for each interaction graph  $G_k(V, E_k)$  to the coupling graph - assuming at present that there always exists at least one assignment for each time slice. It is likely that two assignments of two different time slices are not equivalent; In this case

transformation operations have to be applied. The overall objective is to minimize these transformation operations.

**Example 3.4.3** (Two Different Assignments). *Let us consider two different assignments of two adjacent time slices. Fig. 3.3 depicts two assignments, where in  $k = 0$  the logical qubit  $q_2$  is placed on physical qubit 3, i.e.  $q_2 \rightarrow 3$ . In the adjacent slice  $k = 1$   $q_2 \rightarrow 0$ . Hence,  $q_2$  needs to be moved from its node 3 to node 0 ...*

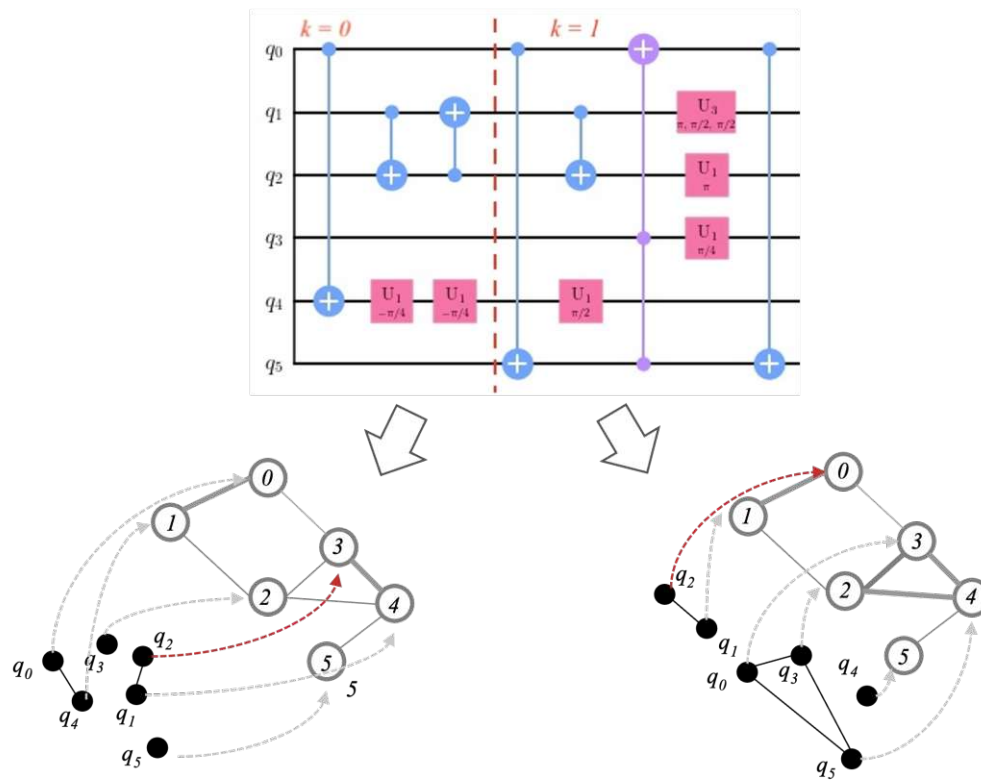


Figure 3.3: In the picture two assignments for time slices  $k = 0$  (left) and  $k = 1$  (right) are plotted. Each assignment is signified by dotted grey arrows pointing from the logical qubits in the interaction graph (black) to the physical qubits in the coupling graph (white/grey). The edges in bold solid grey again signify that it is used by at least one gate. The red dotted arrows outline the situation for  $q_2$ , which is first assigned to node 3 and then to 0 in  $k = 0$ , and  $k = 1$ , respectively.

In general each logical qubit in time slice  $k - 1$  assigned to a node  $i'$  in the coupling graph, which is assigned to a coupling graph node  $j' \neq i'$  in the adjacent time slice  $k$ , needs to be relocated from its coupling graph node  $i'$  in time slice  $k - 1$  to the target coupling graph node  $j'$  in time slice  $k$ . Since we are considering a connected graph, this can always be achieved by a so called *SWAP-chain*.

**Definition 3.4.7** (SWAP-chain). A SWAP-chain is a sequence of one or more SWAP-gates between a logical qubit  $i$  starting out at some location  $i'$  and other logical qubits; in this way qubit  $i$  is moved to its target destination  $j'$ , where  $i'$  and  $j'$  denote different physical qubits in the coupling graph in two adjacent time slices  $k - 1$  and  $k$  respectively. Thus, a SWAP-chain happens alongside of an edge-path in the coupling graph. Performing one or more SWAP-gates according to the needs of a quantum program is termed *routing*.

**Example 3.4.4** (Simple SWAP-chain with one SWAP). *Back to "...  $q_2$  needs to be moved from its node 3 to node 0 ...": The obvious SWAP-chain, which comprises only one SWAP-gate is  $\text{SWAP}(q_0, q_2)$  depicted in Fig. 3.5.*

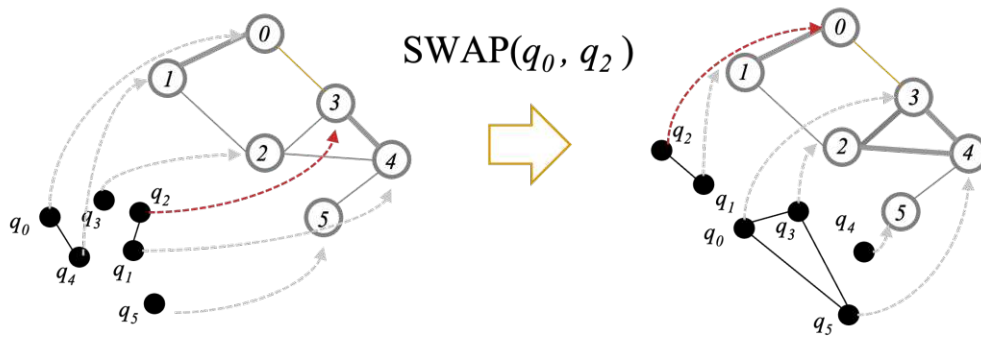


Figure 3.4: In order for the logical qubit  $q_2$  starting out from node 3 in  $k - 1$  to be assigned to node 0 in time slice  $k$ , it has to perform a SWAP with the logical qubit placed on 0, i.e.  $q_0$ . The coupling graph edge  $(0,3)$  alongside of which the SWAP happens is marked in yellow.

**Example 3.4.5** (SWAP-chain with more SWAPs). *A more comprehensive SWAP-chain is required for  $q_1$ , since it is assigned to node 4 in  $k - 1$  and then assigned to node 1 in  $k$  and these nodes are separated by a 3-edge distance. In order to provide for the transition  $q_1 \rightarrow 4$  to  $q_1 \rightarrow 1$  we need to take into account the SWAP that has already happened in Ex. 3.4.4 and thus apply the SWAP-chain:  $\text{SWAP}(q_1, q_0) + \text{SWAP}(q_1, q_3) + \text{SWAP}(q_1, q_4)$ . The coupling graph edges involved in the chain are marked in yellow in Fig. 3.5. After performing this SWAP-chain  $q_0, q_2$  (2 out of 5 qubits) have reached their target nodes. The remaining task is to find SWAP chains for all other qubits until all qubits are placed on their target nodes.*

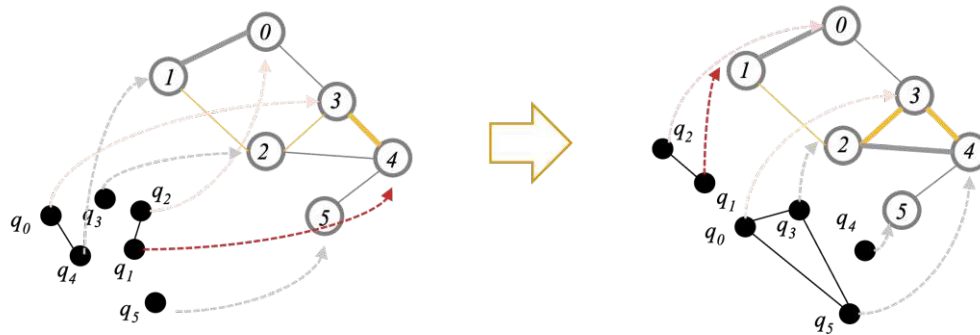


Figure 3.5: After the first simple SWAP (marked in light pink) a more comprehensive SWAP-chain comprising 3 SWAP-gates is required to transfer  $q_1$  to its target location node 1 in the coupling graph. The coupling graph edges involved in the chain are marked in yellow. A possible SWAP-chain reads: SWAP( $q_1, q_0$ ) - SWAP( $q_1, q_3$ ) - SWAP( $q_1, q_4$ ).

*Remark.* A SWAP-chain needs to take into account the changes that have already been made by previous SWAP-chains.

We saw that applying SWAP-chains between time slices yields the necessary transformations between the assignments of time slices, which is equivalent to adding SWAP-gates in between time slices of the quantum program.

**Example 3.4.6** (Add SWAP-gates to quantum program). *The SWAP-chains of our example routing are added between the time slices of the quantum program as depicted in Fig. 3.6.*

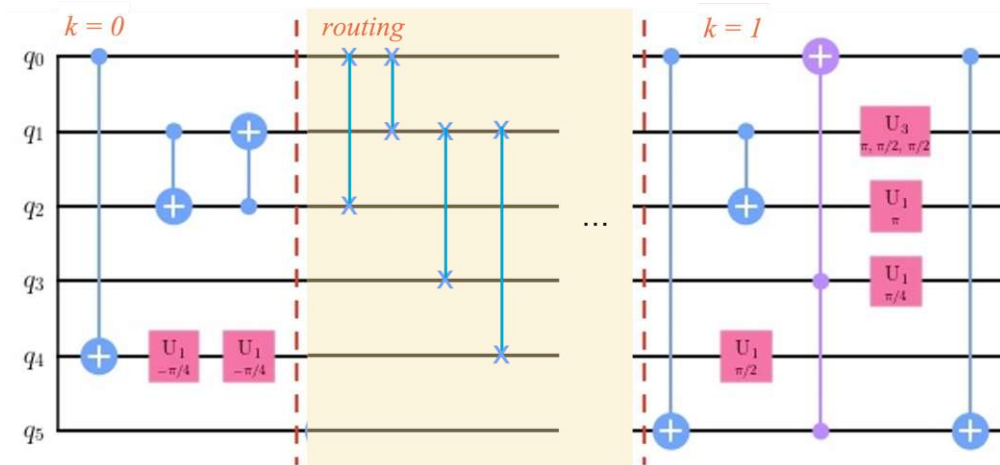


Figure 3.6: SWAP-chains are inserted in the routing section (shaded in yellow) in between the time slices  $k = 0$  and  $k = 1$ . The first SWAP-gate and the three last SWAP-gates correspond to Ex. 3.4.4 and Ex. 3.4.5, respectively. The dots represent the remaining SWAP-chains still required for  $q_3, q_4$  and  $q_5$ .

The corresponding graph-theoretical problem describing the transition from one assignment to another is termed the *token swapping problem* and it has NP-complexity [68].

**Definition 3.4.8** (Token Swapping Problem). Given an undirected connected graph  $G(E, V)$ , where each node has an assigned token, the goal is to swap tokens between adjacent nodes until a desired distribution of tokens is achieved with the minimum number of token swaps.

Luckily, there exists an approximate algorithm solving the token swapping problem, which yields the SWAP-chains necessary with an upper bounded number of SWAPs [69]; we will utilize this algorithm later.

### 3.5 The Cost Function

As outlined in the previous section the problem formulation will consist of two tasks, a) finding an *assignment* or *subgraph isomorphism* for each time slice, while b) solving the *token swapping problem* between all neighboring pairs of time slices.

*Remark.* These concepts are certainly not new and have been used in qubit mapping, as for example in an exhaustive method developed by [14]. These strategies utilized for qubit mapping have however so far to our knowledge never been formulated as a quadratic unconstrained binary optimization problem, which this thesis is all about.

Before delving into the details of the cost function, we give a general definition of quadratic unconstrained binary optimization.

**Definition 3.5.1** (Quadratic unconstrained binary optimization model). The QUBO model is in general a NP-complete optimization problem; the minimizing version which we will use reads

$$x^* = \min_x x^T Q x = \min_x \left( \sum_{i < j} Q_{ij} x_i x_j + \sum_i Q_{ii} x_i \right) \quad (3.1)$$

where  $x$  is a vector of binary decision variables of size  $N$ , i.e.  $x = \{0, 1\}^N$  and  $Q$  is a square matrix of  $N \times N$  real valued constants. The matrix' constants encode the problem, such that the vector  $x^*$  corresponding to the minimum value is the solution to the optimization problem.

In what follows, the components of the developed QUBO formulation for qubit mapping will be thoroughly explored.

### 3.5.1 Find Assignments

Given a quantum program sliced into  $m$  time slices. The objective is to find an assignment or subgraph isomorphism for each time slice  $k$  to the coupling graph  $G'(V', E')$ . In order to encode the problem into a QUBO we need to define the solution array.

**Definition 3.5.2** (Binary Solution Array). All possible assignments for a time slice  $k$  are represented by a binary array  $\mathbf{x}^k \in \{0, 1\}^{n \cdot n'}$ , where  $n = |V|$  denotes the number of logical qubits and  $n' = |V'|$  denotes the number of physical qubits in the coupling graph.

$$\mathbf{x}^k := [x_{0,0}^k, x_{0,1}^k, \dots, x_{0,n'-1}^k, x_{1,0}^k, \dots, x_{1,n'-1}^k, \dots, x_{n-1,n'-1}^k] \quad (3.2)$$

$x_{i,i'}^k = 1$  signifies that logical qubit  $i$  is assigned to the physical qubit  $i'$  in time slice  $k$  and  $x_{i,i'}^k = 0$  signifies that logical qubit  $i$  is *not* assigned to the physical qubit  $i'$  in time slice  $k$ . Thus, the problem size  $N$  grows with the number of logical qubits  $n$ , the number of physical qubits  $n'$  and the number of time slices  $m$ , i.e.  $N = n \cdot n' \cdot m$ .

We will use the following definition to encode an assignment [70]:

**Definition 3.5.3** (Assignment QUBO). The function  $F$  to be minimized consists of two quadratic functions  $J$  and  $P$ ; both functions serve as penalties should the



solution not be a subgraph isomorphism. The sets  $E_k$  and  $E'$  denote the edge sets for the interaction graph and the coupling graph, respectively.

$$\min_{\mathbf{x}^k} F(\mathbf{x}^k) = \min_{\mathbf{x}^k} J(\mathbf{x}^k) + P(\mathbf{x}^k) \quad (3.3)$$

where

$$J(\mathbf{x}^k) := \sum_{i \in V} (1 - \sum_{i' \in V'} x_{ii'}^k)^2 + \sum_{i' \in V'} \left(\frac{1}{2} - \sum_{i \in V} x_{ii'}^k\right)^2 \quad (3.4)$$

$$P(\mathbf{x}^k) := \sum_{(i,j) \in E_k} \sum_{(i',j') \notin E'} x_{ii'}^k x_{jj'}^k \quad (3.5)$$

Summing over all time slices  $k$  results in the overall cost function to find assignments for all time slices.

$$\min_{\mathbf{x}} \sum_k F(\mathbf{x}^k) \quad (3.6)$$

*Proof.* Since all terms of the sum (3.6) only consist of quadratic functions with a minimum value greater or equal to zero, it is at its minimum value iff all terms exhibit their minimum values, i.e.  $J$  and  $P$  exhibit  $\frac{n'}{4}$  and zero, respectively. The first sum of  $J$  is zero, iff  $\mathbf{x}^k$  assigns each logical qubit  $i \in V$  exactly once to a physical qubit  $i' \in V'$ . The second sum of  $J$  exhibits its minimum value of  $\frac{n'}{4}$ , if one or no logical qubit  $i$  is assigned to a physical qubit  $i'$ .  $P$  preserves the edges, meaning it is greater zero if an edge in the interaction graph is mapped to a non-edge in the coupling graph. Thus, these conditions are equivalent with the property that the sum (3.6) penalizes  $\mathbf{x}^k$  should it be a non-assignment.  $\square$

**Example 3.5.1** (Penalties for Non-Assignments). *There are four ways to violate the constraints for an assignment. a) A logical qubit is not mapped to any coupling graph node, b) a logical qubit is mapped more than once to two or more nodes in the coupling graph, c) two or more logical qubits are mapped to one node in the coupling graph and d) an edge in the interaction graph has no representative in the coupling graph, Fig. 3.7.*

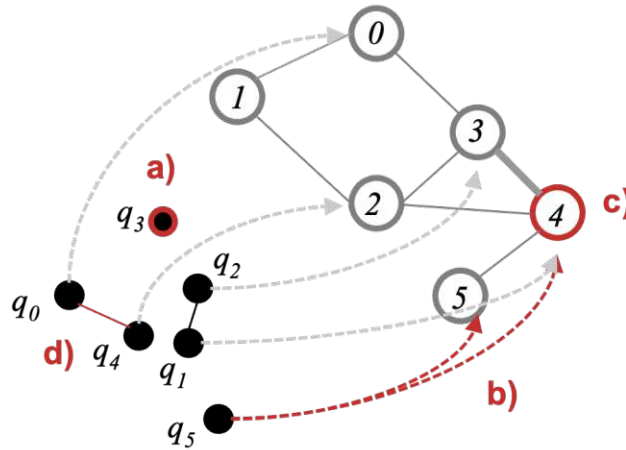


Figure 3.7: Violations of an assignment, where marks a) a logical qubit is not mapped to any coupling graph node, b) a logical qubit which is mapped more than once to the coupling graph, c) two or more logical qubits are mapped to one node in the coupling graph and d) an edge in the interaction graph which has no representative in the coupling graph.

The logical qubits "live" on the same set of physical qubits throughout the course of the program, i.e. in a correct mapping solution all assignments target the same set of qubits.

*Proof.* Given an assignment, where  $i \rightarrow i', j \rightarrow j'$ , the exchange of locations  $i'$  and  $j'$  on the coupling graph between the logical qubits  $i$  and  $j$ , is achieved by a SWAP-chain. Both locations  $i'$  and  $j'$  are necessarily occupied before and after the SWAP-chain by qubits  $i \rightarrow i', j \rightarrow j'$  and  $i \rightarrow j', j \rightarrow i'$ , respectively. Thus, no SWAP-chain can ever lead to a logical qubit being mapped to a coupling graph node outside the initial set of occupied physical qubits.  $\square$

*Remark (Set of Occupied Physical Qubits).* Given a bigger coupling graph than interaction graph  $V < V'$ , there is at least one coupling graph node not occupied by any logical qubit. Since SWAP-gates take place between logical qubits, the free coupling graph node will always stay free over the course of all time slices, as presented in Fig. 3.8.

**Definition 3.5.4 (Initial Set and Free Set).** Given one or more assignments, the initial set  $V'_{init}$  comprises all occupied physical qubits in the coupling graph, where naturally  $|V'_{init}| = |V|$ , while the free set  $V'_f$  determines the complementary set of nodes of the coupling graph, i.e.  $V'_f = V' \setminus V'_{init}$ .

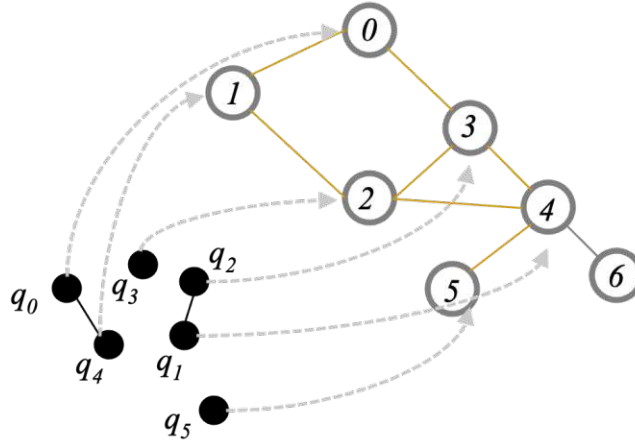


Figure 3.8: All pairs of logical qubits mapped to a pair of nodes in the coupling graph with a connecting edge (marked in yellow) can perform a SWAP. No SWAP involves a free coupling graph node 6, thus the latter stays free in all assignments.

The goal is to ensure that the initial set is preserved over all assignments. The function  $S$  will serve as penalization function for every logical qubit  $i$  being mapped to a physical qubit  $i' \in V'_f$ .

**Definition 3.5.5** (Cost function  $S$ ). The quadratic term  $S$  ensures preservation of the initial placement, it reads

$$\min_{\mathbf{x}} S(\mathbf{x}^k, \mathbf{y}) \quad (3.7)$$

$$S(\mathbf{x}^k, \mathbf{y}) := (n' - n - \sum_{i' \in V'} y_{i'})^2 + \sum_{i \in V} \sum_{i' \in V'} x_{i,i'}^k y_{i'} \quad (3.8)$$

where  $\mathbf{y} \in \{0, 1\}^{n'}$  are  $n'$  additional binary variables, signifying the occupied and free set:  $y_{i'} = 0$  or  $y_{i'} = 1$  mark that physical qubit  $i'$  is either in the initial set or in the free set, respectively. Since these variables are chosen once for all time slices, the first part of  $S$  ensures that there are exactly  $(n' - n)$  qubits in the free set for all time slices. The second sum then penalizes every logical qubit mapped to one of the free set nodes  $V'_f$ .

Summing over all time slices  $k$  results in the overall cost function for qubit placements, which reads

$$\min_{\mathbf{x}} \sum_k S(\mathbf{x}^k, \mathbf{y}) \quad (3.9)$$

*Proof.* Since all components of the sum only consists of quadratic terms, it is zero if and only if all components  $S = 0$ . Further, since for all slices  $k$ , both terms are

quadratic terms,  $S = 0$  if and only if both terms are equal to zero. The first term is zero, iff  $y$  consists of exactly  $n$  zero elements and  $(n' - n)$  non-zero elements. Fixing the second term of  $S$  to an arbitrary logical qubit  $i$ , the term is greater than zero, if and only if qubit  $i$  is placed on a physical qubit  $i'$  where  $y_{i'} = 1$ . Since this naturally holds for all other qubits  $i$  and terms of the sum, these conditions with the previously set definitions are equivalent with the property that penalization is incurred, if either the initial set held more or less than  $n$  physical qubits or if a logical qubit is placed on a physical qubit outside the initial set.  $\square$

**Example 3.5.2** (Violating  $S$ ). *Given are two assignments  $\mathbf{x}^0$  and  $\mathbf{x}^1$  involving two logical qubits, and a coupling graph with three nodes, outlined in Fig. 3.9.*

$$\mathbf{x}^0 = \left[ \underbrace{100}_{i=0}; \underbrace{010}_{i=1} \right] \quad (3.10)$$

$$\mathbf{x}^1 = \left[ \underbrace{100}_{i=0}; \underbrace{001}_{i=1} \right] \quad (3.11)$$

Regardless of what  $S$  decides for the variables  $\mathbf{y}$  it will result in a violation; The only options satisfying the first sum of  $S$  are  $\mathbf{y} = [010]$ ,  $\mathbf{y} = [100]$  or  $\mathbf{y} = [001]$  (hence one qubit  $i'$  being in free set  $|V'_f| = 1$ ). Let us write  $S$  explicitly for option  $\mathbf{y} = [001]$ , i.e.  $V'_f = 2$

$$S(\mathbf{x}^0) + S(\mathbf{x}^1) = (3 - 2 - (0 + 0 + 1))^2 \quad (3.12)$$

$$+ \underbrace{1 \cdot 0 + 0 \cdot 0 + 0 \cdot 1}_{i=0, k=0} + \underbrace{0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1}_{i=1, k=0} \quad (3.13)$$

$$+ \underbrace{1 \cdot 0 + 0 \cdot 0 + 0 \cdot 1}_{i=0, k=1} + \underbrace{0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1}_{i=1, k=1} \quad (3.14)$$

$$= 1 \quad (3.15)$$

It is left to the reader to try it with the other options;  $S(\mathbf{x}^0) + S(\mathbf{x}^1) > 0$ , which is equivalent to the solution vectors targeting different sets of physical qubits.

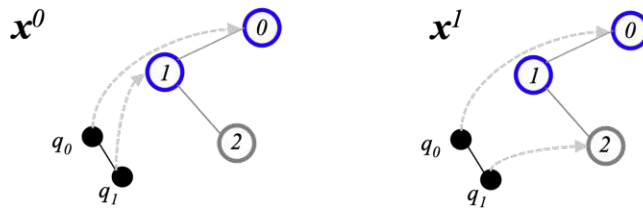


Figure 3.9: Sketch of the two assignments of Eqs. 3.10 and 3.11, where the blue bounded nodes in the coupling graph mark that they in the initial set  $\mathbf{y} = [001]$ .

Combining the introduced cost functions  $F$  Eq. 3.6 and  $S$  Eq. 3.9 yields an assignment solution for all the time slices  $m$  in the quantum program. We thus write the assignment part  $H_a$  of the cost function

$$\min_{\mathbf{x}} H_a(\mathbf{x}) \quad (3.16)$$

where

$$H_a(\mathbf{x}) := \sum_{k=1}^m (F(\mathbf{x}^k) + S(\mathbf{x}^k, \mathbf{y})) - \frac{m \cdot n'}{4}. \quad (3.17)$$

We subtract the theoretical minimum value of  $\min_{\mathbf{x}} \sum_k F(\mathbf{x}^k) = \frac{m \cdot n'}{4}$  in order to have an overall minimum value of zero, which does not change the findings of the cost function and is more convenient to remember.

### 3.5.2 Minimizing Transformation Operations

Lastly, all pairs of two adjacent assignments should reduce the overall number of transformations. As described in the previous Sec. 3.4.2 the transition from one assignment  $\mathbf{x}^{k-1}$  to another  $\mathbf{x}^k$  is achieved via SWAP-chains and the corresponding problem is termed token swapping. Luckily, there always exists a solution for two assignments which cover the same node set of a connected graph, i.e. the initial set in our case as discussed by [69]. Further, the same authors developed an approximate algorithm, which solves the token swapping problem in polynomial time with a bounded number of SWAPs;

**Definition 3.5.6** (Swap bounds approximate algorithm [69]). The number of swaps  $n_s$  necessary for a placement transition is proven to be bounded by

$$\frac{L}{2} \leq n_s \leq 2L. \quad (3.18)$$

The bounds are determined by the so called distance measure  $L$ .

**Definition 3.5.7** (Distance measure  $L$ ). The distance measure  $L$  is the sum over all shortest distances  $d_{i'j'}$ , i.e. the smallest connected path of edges between nodes  $i'$  and  $j'$  on the coupling graph considering two assignments  $\mathbf{x}^{k-1}$  and  $\mathbf{x}^k$ .

$$L(\mathbf{x}^{k-1}, \mathbf{x}^k) := \sum_i \sum_{i' \neq j'} d_{i'j'} x_{i,i'}^{k-1} x_{i,j'}^k. \quad (3.19)$$

**Example 3.5.3.** Let us again assume the following two assignments  $\mathbf{x}^0$  and  $\mathbf{x}^1$  involving two logical qubits, and a coupling graph with three nodes, outlined in Fig.

3.10.

$$\mathbf{x}^0 = \left[ \underbrace{100}_{i=0}; \underbrace{010}_{i=1} \right] \quad (3.20)$$

$$\mathbf{x}^1 = \left[ \underbrace{010}_{i=0}; \underbrace{100}_{i=1} \right] \quad (3.21)$$

Hence  $L = d_{01}x_{00}^0x_{01}^1 + d_{10}x_{11}^0x_{10}^1 = 2$ . The obvious solution is the  $SWAP(q_0, q_1)$ , i.e. one transformation counted by the lower bound  $L/2 = 1$ .

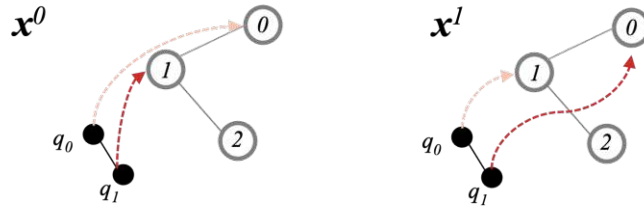


Figure 3.10: Sketch of the two assignments of Eqs. 3.20 and 3.21, where the edge in between the target coupling graph nodes (where the colored arrows point to) already indicates how many SWAPs will be necessary, i.e. one SWAP.

Since  $L$  has already the desired quadratic form, it will promptly serve as the last part of the cost function, which is responsible for minimizing SWAPs and thus term this part  $H_s$ .

$$\min_{\mathbf{x}} H_s(\mathbf{x}) \quad (3.22)$$

where

$$H_s(\mathbf{x}) := \sum_k L(\mathbf{x}^k, \mathbf{x}^{k-1}) \quad (3.23)$$

### 3.5.3 The Final Form

Combining Eqs. 3.17 and 3.23 yields the final cost function  $H$  for the qubit mapping problem

$$\min_{\mathbf{x}} H(\mathbf{x}) \quad (3.24)$$

where

$$H(\mathbf{x}) := H_a(\mathbf{x}) + \lambda \cdot H_s(\mathbf{x}). \quad (3.25)$$

with  $\lambda > 0$  referred to as *SWAP-penalization multiplier*, which is necessary to scale the SWAP cost  $H_s$ . In general we want to penalize SWAPs as much as possible which means we favor a high value of  $\lambda$ . However, according to the proofs for F (Eq. 3.6) and S (Eq. 3.9),  $H_s$  has to be at its minimum value zero in order to represent an assignment. This means any values greater than zero are equivalent to at least one wrong assignment, and this in turn means the circuit cannot be executed. Thus, the mapping solution should always prefer a minimum value of  $H_a$  before a minimum value of  $H_s$ .

**Example 3.5.4.** *Let us assume for a solution  $\mathbf{x}_1$ , the components of the cost function yield  $H_a(\mathbf{x}_1) = 1$  and  $H_s(\mathbf{x}_1) = 1$ . For another solution  $\mathbf{x}_2$ , the cost function results in  $H_a(\mathbf{x}_2) = 0$  and  $H_s(\mathbf{x}_2) = 3$ . In total the cost functions for  $\mathbf{x}_1$  and  $\mathbf{x}_2$  yield 2 and 3, respectively. Even though the solution  $\mathbf{x}_1$  yields obviously a wrong assignment and  $\mathbf{x}_2$  yields a correct assignment, the solver would still decide that  $\mathbf{x}_1$  to be the minimum solution.*

In order to avoid a scenario described in Ex. 3.5.4, i.e. guarantee favoring assignments over SWAP minimization, we want to enforce

$$\lambda \cdot H_s < 1. \quad (3.26)$$

In theory, this could be achieved by choosing  $\lambda < \lambda_{max}$  with

$$\lambda_{max} := \frac{1}{\max H_s} \quad (3.27)$$

hence

$$\lambda \cdot H_s < \lambda_{max} \cdot H_s = \frac{1}{\max H_s} \cdot H_s \leq 1. \quad (3.28)$$

The  $\max H_s$  is however not a priori known, since we do not know beforehand which subset of physical qubits are chosen and hence we do not know which elements of the distance matrix will be used.

Summa summarum: we cannot set an upper bound for  $\lambda$ . We however should still choose  $\lambda$  reasonably small, such that

$$\lambda \cdot H_s \approx 1. \quad (3.29)$$

As the  $H_s$  grows with the size of the quantum program, as well as the size of the coupling graph,  $\lambda$  is defined as follows

**Definition 3.5.8.** (SWAP-penalization multiplier  $\lambda$ )

$$\lambda := \frac{f}{n \cdot n_g}$$

with the number of qubits  $n$  and two-qubit gates  $n_g$ , and a scaling factor  $f > 0$ .

**Example 3.5.5.** Choosing  $f = 1$ , we assume that there are less than  $n \cdot n_g$  SWAPs expected, i.e.  $\lambda H_s < \frac{1}{nn_g} \max H_s = \frac{1}{nn_g} nn_g = 1$ . This equals a scenario where only one two-qubit gate is in every time slice and we would have to apply a SWAP to all  $n$  qubit gates for each pair of time slices. Choosing  $f = 10$  or  $f = 100$  equals expecting  $10\times$  or  $100\times$  less SWAPs, respectively, which is more realistic.

### 3.5.4 On the Problem Size and Choice of Solver

The size of the binary solution vector  $N = m \cdot n \cdot n'$  grows with the number of slices  $m$ , the number of logical qubits  $n$  and the number of physical qubits  $n'$ . In the maximum case, the number of slices  $m$  is equivalent to the number of two-qubit gates of the quantum program, hence a slice holds then one two-qubit gate. Thus, the binary vector scales cubically with both the size of the circuit ( $n$  and  $m$ ) and the quantum hardware ( $n'$ ). Rapidly increasing solution sizes quickly result in matrices with dimensions of more than tens to hundreds of thousands (e.g.  $N = 10^4$  solving for a quantum circuit with  $n = 10$  qubits on a  $n' = 10$  qubit hardware with  $m = 100$  time slices; which is realistic considering the maximum case when a circuit with  $m$  gates also is sliced into  $m$  parts).

Studies of exact QUBO solvers typically use instances with hundreds up to thousands of variables [71, 72]. Thus, the problem sizes we want to consider for fall in the regime where heuristics are more applicable: the goal is to find near-optimal solutions within reasonable time limits. For this purpose we use the dwave-neal [73] simulated annealer [74] as our dedicated solver heuristic, as it comes with the open source package of D-Wave Systems and can compete with commercial QUBO solvers [75].

## 3.6 Slicing with a Genetic Algorithm

In Sec. 3.4.2 we assumed that there exists at least one assignment for each time slice. This is in general not provided by an arbitrary slicing. In order to ensure a slicing where for each time slice an assignment exists, a genetic algorithm is implemented according to literature recommendations [76], which improves on the slices' time points in the quantum program.

**Definition 3.6.1** (Genetic Algorithms [76]). Genetic Algorithms are optimization techniques where the solution to a cost function is encoded as arrays of bits or character strings, called *chromosomes*. These chromosomes are manipulated through genetic operations, such as *crossover* and *mutation*, and selected according to their *fitness* to optimize their associated cost. The process involves encoding the objectives, defining a fitness function, creating a population of individuals, and evaluating their fitness over multiple generations.



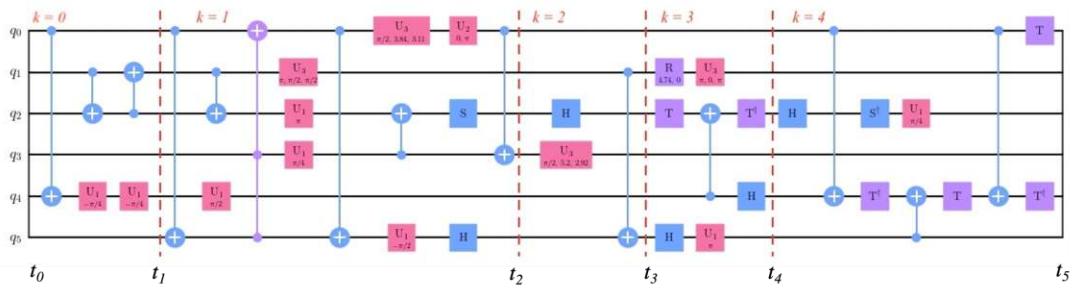


Figure 3.11: Sliced quantum program with associated time points  $\mathbf{t} = [0, 3, 9, 10, 11, 12, 15]$

Let us define the components of the GA used in more detail:

**Definition 3.6.2** (Chromosome). A chromosome  $\mathbf{t}$  is an array of arbitrary ascending circuit time points. We are only interested in two-qubit gates and we assume for the time count that it is possible to execute one two-qubit gate at a time. Hence, the number of time points of the program is equivalent to the number of two-qubit gates in the circuit. In order to be consistent each chromosome  $\mathbf{t} = [t_0, t_1, \dots, t_{m+1}]$  starts with  $t_0 = 0$  and ends with  $t_{m+1} = n_g + 1$ , where  $m$  denotes the number of slices and  $n_g$  the number of two-qubit gates. Each slice  $k$  thus has two confining time points  $[t_k, t_{k+1}]$ , which contains at least one two-qubit gate. An example of a sliced circuit with marked time points is shown in Fig. 3.11.

**Definition 3.6.3** (Population). A population is a list of  $n_p$  chromosomes.

In the subsequent section, these definitions will be used to explain the code implementation to introduce the used genetic operations and formalise the algorithm as pseudocode.

## Chapter 4

# Implementation

The introduced qubit mapping method is implemented in Python 3.8 and publicly available on Github [17]. In the following, an overview of the modules, their functionality and respective interaction will be explained.

## 4.1 Overview Modules

The implementation comprises the following modules

- `CouplingGraph.py` contains the class `CGraph` and several lists of edges for different hardware systems; as the name suggests, it is responsible for the generation of the target coupling graph.
- `InteractionGraph.py` comprises the class `IGraph`, and holds the functionality to import a quantum program, to slice the quantum program into random time slices and generate the corresponding interaction graphs.
- `QUBOmodel.py` holds the class `QUBO` which provides the functionality to build and solve the quadratic cost function from a sliced quantum program and the target coupling graph. Further, it contains the functions `CHECKMAPPING` and `COUNTSWAPS` to verify a mapping-solution and to count its swaps, respectively.
- `GeneticAlgorithm.py` comprises the genetic algorithm, with the genetic operations `MUTATION`, `CROSSOVER` and `CREATE` to generate and transform chromosomes. The module is applied to improve the slicing while ensuring a valid mapping.

## 4.2 Functions

In what follows, definitions of class functions and additional functions are given.

### 4.2.1 Fitness

The fitness value of a chromosome  $\mathbf{t}$  is an important concept in a genetic algorithm. In our mapping method, it is defined as follows

**Definition 4.2.1** (Fitness function). The fitness function  $F(\mathbf{t})$  serves as a measure of how well a random slicing of the circuit performs compared to another random slicing. The time-point arrays form the input, which are then used to generate the respective slice-interaction graphs. These graph properties build the QUBO model. Subsequently, a built-in simulated annealer of the `dwave-neal` library is called to solve the QUBO model  $H$  (Eq. 3.25). The result of  $F(\mathbf{t})$  is thus the minimum value found by simulated annealing for the QUBO model for a chromosome  $\mathbf{t}$ . Pseud-code listings in Alg. 1.

---

#### Algorithm 1: Fitness in QUBOmodel.py

---

```

function F( $\mathbf{t}$ ):
    QUBO <quadratic model>  $\leftarrow$  build model from time points in
    chromosome  $\mathbf{t}$  with corresponding objects defined in InteractionGraph.py
    and CouplingGraph.py.
    min <int>, x <bool array>  $\leftarrow$  call neal.SimulatedAnnealing(QUBO).
    return min, x
end function

```

---

*Remark.* Since we are interested in the mapping solution of the QUBO, the Fitness function returns both the fitness value as well as the solution array.

### 4.2.2 Validation and SWAP Count

In order to ensure a valid mapping, the assignment part of the cost function  $H_a$  Eq. 3.17 needs to be the optimal value. This can easily be checked by inserting the binary solution array into  $H_a$ , listed in Alg. 2. Further in order to calculate the upper and lower bound of SWAP-gates necessary using a mapping solution, we calculate the second part of the cost function  $H_s$  Eq. 3.23 as listed in Alg. 3.

---

**Algorithm 2:** Validation in QUBOmodel.py

---

```

function CHECKMAPPING(x):
    ismapping <bool>  $\leftarrow$  evaluate  $H_a(\mathbf{x}) \stackrel{?}{=} 0$ 
    return ismapping
end function

```

---



---

**Algorithm 3:** Number of SWAP-gates in QUBOmodel.py

---

```

function COUNTSWAPS(x):
    min nswaps <int>  $\leftarrow 1/2 \cdot \sum_k L(\mathbf{x}^k, \mathbf{x}^{k+1})$ 
    max nswaps <int>  $\leftarrow 2 \cdot \sum_k L(\mathbf{x}^k, \mathbf{x}^{k+1})$ 
    return min nswaps, max nswaps
end function

```

---

### 4.2.3 Genetic Operations

The genetic operations CREATION, CROSSOVER and MUTATION are defined as follows:

---

**Algorithm 4:** Creation in GeneticAlgorithm.py

---

```

function CREATION( ):
     $\mathbf{t}_{inner}$  <int list>  $\leftarrow$  select a random set of elements  $\in [1, n_g]$ 
     $\mathbf{t}$  <int list>  $\leftarrow [0] + \text{sort}(\mathbf{t}_{inner}) + [n_g + 1]$ 
    return  $\mathbf{t}$ 
end function

```

---

---

**Algorithm 5:** Crossover in GeneticAlgorithm.py

---

```

function CROSSOVER( $\mathbf{t}_c$ ,  $\mathbf{t}_d$ ):
     $\mathbf{t} \leftarrow$  combination of left half of parent  $\mathbf{t}_c$  and right half of parent  $\mathbf{t}_d$ .
     $\mathbf{t} \leftarrow \text{unique}(\mathbf{t})$  Note: creates unique set in case of duplicates caused by
    previous step
    return  $\mathbf{t}$ 
end function

```

---



---

**Algorithm 6:** Mutation in GeneticAlgorithm.py

---

```

function MUTATION( $\mathbf{t}$ ):
     $\mathbf{t}_{inner}$  <int list>  $\leftarrow$  select a random set of elements of  $\mathbf{t}$ 

    for  $t_i$  in  $\mathbf{t}_{inner}$  do
         $r = \text{random}(0,1)$ 
        if  $r < 0.5$  then
            |  $t_i = t_i + 1$ 
        else
            |  $t_i = t_i - 1$ 
        end
    end
     $\mathbf{t}$  <int list>  $\leftarrow [0] + \text{sort}(\mathbf{t}_{inner}) + [n_g + 1]$ 
     $\mathbf{t} \leftarrow \text{unique}(\mathbf{t})$  Note: creates unique set in case of duplicates caused by
    previous step
    return  $\mathbf{t}$ 
end function

```

---

### 4.3 Program Flow

1. Create initial population, i.e. generation  $g = 0$ , of  $n_p$  chromosomes  $[\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_{n_p}]$ , and evaluate their individual fitness values  $[F(\mathbf{t}_1), F(\mathbf{t}_2), \dots, F(\mathbf{t}_{n_p})]$  defined in Alg. 1.
2. Build population of the next generation  $g = 1$  with genetic operations:
  - The fittest  $p\%$  chromosomes of the initial population are chosen as parents.
  - Further  $c\%$  chromosomes of the initial population are created by crossover of two randomly selected parents with the genetic operation CROSSOVER Alg. 5.
  - Another  $m\%$  chromosomes undergo mutation with the genetic operation MUTATION Alg. 6.
  - The remaining  $r\%$  chromosomes are created anew with the genetic operation CREATE Alg. 4,

where  $p, c, m, r \in \mathbb{R}^+$  are adjustable and naturally  $p + c + m + r = 100$ .

3. The generation is complete, the fitness  $F$  of all chromosomes is evaluated and sorted accordingly.
4. Repeat 2.-3. steps  $g_{max}$  times, where  $g_{max} \in \mathbb{N}^+$  is adjustable.
5. Select the mapping solution of the best chromosome in the  $g_{max}$ th generation and
  - check if it is a valid mapping by calling the operation CHECKMAPPING Alg. 2
  - count the number of SWAP-gates necessary in the optimal and maximum case via the function COUNTSWAPS Alg. 3.
6. Return the **mapping solution** ( $x$ ), the **validation outcome** (ismapping) and respective **SWAP counts** (min nswaps, max nswaps).

A schematic flow diagram is shown in Figure 4.1.

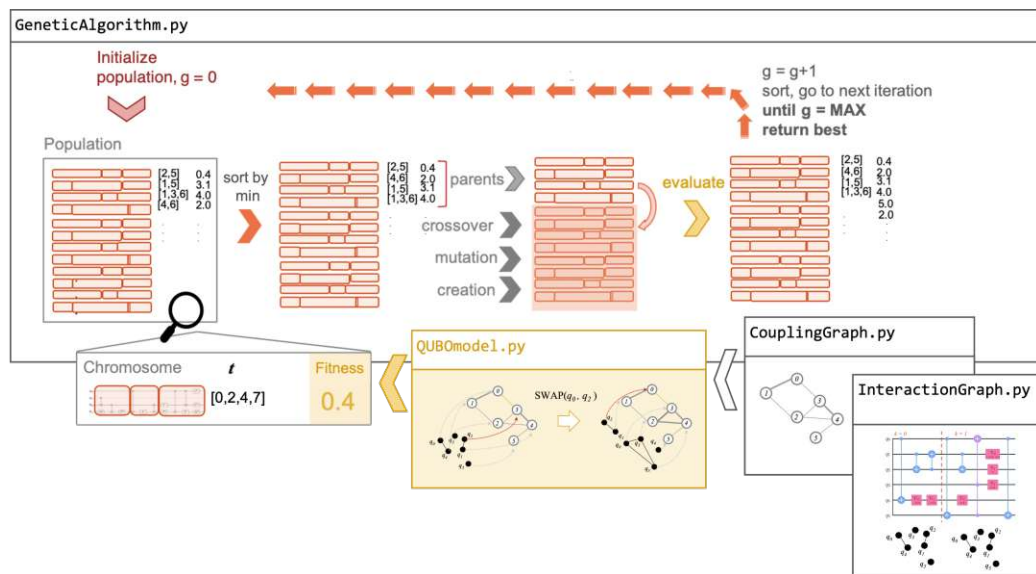


Figure 4.1: Optimization flow of the implemented qubit mapping method. The central module `GeneticAlgorithm.py` comprises the genetic algorithm which makes use of `InteractionGraph.py` and `CouplingGraph.py` and `QUBOmodel.py`. In `GeneticAlgorithm.py` the chromosomes are created, sorted according to their fitness and improved over a set maximum number of generations  $g_{max}$ . The fitness is calculated via `QUBOmodel.py`. This module makes use of `InteractionGraph.py` and `CouplingGraph.py` to solve the cost function  $H$  for a chromosome  $t$ .

## 4.4 Additional Functions

Once a solution is found, the functions `LISTMAPPING` and `ANIMATEMAPPING` can be used to generate a more comprehensible translation of the binary solution array.

- `LISTMAPPING` lists the assignments of two-qubit gates to the coupling graph, it can also be used as test function; however as it is equipped with boolean functions which check for valid assignments, an example is depicted in Figure 4.2.
- An animation created with `ANIMATEMAPPING`, shows the coupling graph with its initial-set, and the course of the execution with one two-qubit gate per frame. A screenshot is depicted in Figure 4.3.



is mapping = False							
	slice	CX-gates	coupling-graph	# placed q1	# placed q2	edge exists in coupling-graph	set is correct
0	0	(2, 0)	(10, 6)	1	1	True	True
1	0	(4, 3)	(1, 7)	1	1	True	True
2	0	(4, 1)	(1, 2)	1	1	True	True
3	0	(3, 1)	(7, 2)	1	1	True	True
4	1	(4, 3)	(1, 7)	1	1	True	True
5	1	(4, 2)	(1, 2)	1	1	True	True
6	1	(4, 0)	(1, 6)	1	1	True	True
7	1	(2, 0)	(2, 6)	1	1	True	True
8	2	(1, 4)	(1, -1)	1	0	False	False
9	2	(1, 3)	(1, 2)	1	1	True	False
10	2	(1, 0)	(1, 6)	1	1	True	False
11	2	(4, 3)	(-1, 2)	0	1	False	False
12	2	(4, 2)	(-1, 7)	0	1	False	False
13	2	(4, 0)	(-1, 6)	0	1	False	False
...							
37	6	(0, 2)	(2, 7)	1	1	True	True
38	6	(3, 2)	(6, 7)	1	1	True	True

final energy  
2.2200000000000273

check: if both logical qubit have exactly one physical qubit in the coupling graph  
check: does the edge exist in the coupling graph?  
check: is the set covering n physical qubits is it equivalent to the first slice?

In order to form an accurate mapping all of those entries have to be True / 1

Figure 4.2: Mapping solution translated to a list generated by LISTMAPPING. On the left we can see the solution translated to slice, CX-gate and where on the coupling graph it is executed. The number -1 in a tuple signifies, that the qubit is not assigned. Further the last two columns signify, if the edge exists in the coupling graph and if the qubits of this edge are within the initial set, respectively.

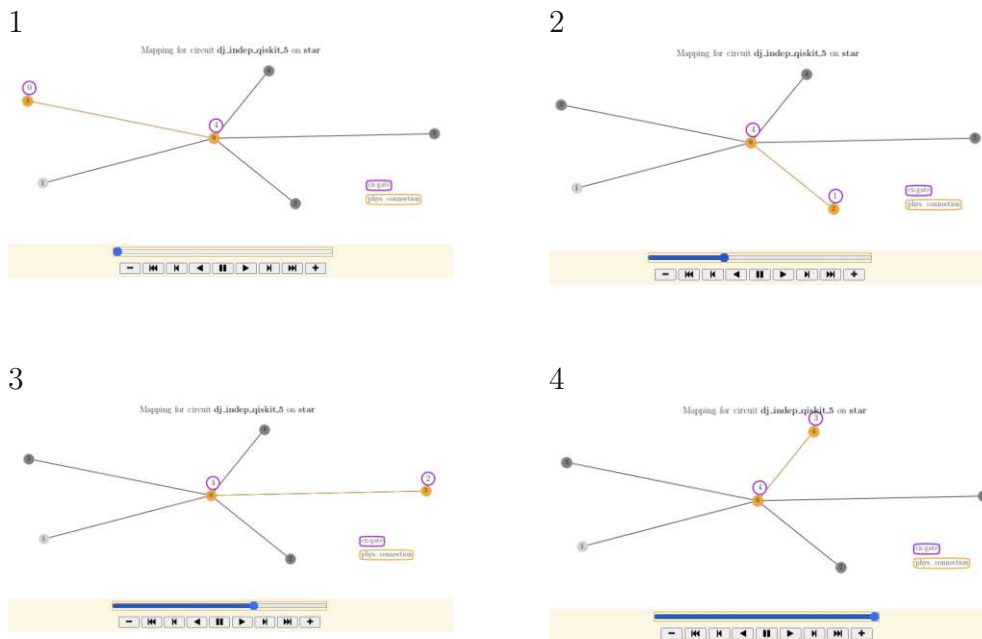


Figure 4.3: Example frames 1-4 generated by ANIMATEMAPPING.

## Chapter 5

# Evaluation

In what follows the experimental setup to comprehensively test the introduced method and the results will be discussed.

## 5.1 Experimental Setup

**System.** All tests were run on an Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz with 256 GB of RAM and 16 cores running Debian GNU/Linux 11.

**Quantum Hardware.** A selection of benchmark quantum programs is mapped to the `ibmqx20` coupling graph depicted in 5.1 with 20 physical qubits connected via 43 edges.

**Parameter settings.** The parameters  $n_p$  (population size) and  $g_{max}$  (max number of generations) are chosen such that one execution takes less than 3 minutes for the largest benchmark,  $n_p = 20$  and  $g_{max} = 15$ . Note, that the parameter settings have a large impact on the performance and runtime of the method. The penalization factor  $\lambda = \frac{f}{n \cdot n_g}$  is dynamically set for each circuit depending on the number of qubits  $n$  and the number of two-qubit gates  $n_g$  with an additional scaling factor  $f$ , in order to dynamically adjust the SWAP-penalization  $\lambda \cdot H_s \approx 1$  as discussed in Sec. 3.5.3.

**Comparison.** The introduced method is compared to four qubit mapping methods, *qmap*, *Qiskit basic*, *Qiskit stochastic* and *Qiskit SABRE*<sup>1</sup>, where for the Qiskit methods, the trivial built-in method for the initial placement is chosen. All of them serve as reference mapping methods in many recent publications; a detailed overview of recently developed mapping methods, their comparing references and other specifications can be found in the Appendix Table B.1.

<sup>1</sup>Qiskit integration of SABRE [77]

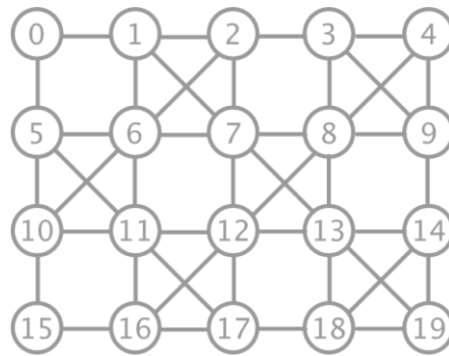


Figure 5.1: IBM Q20 Tokyo coupling graph

**Experiments.** Two experiments are conducted to demonstrate significant difference in the mapping outcomes due to different scaling factors  $f$ ; in the first experiment the scaling factor is set  $f = 10$  and  $f = 100$  in the second as discussed in Ex. 3.5.5. Each given data point in the following is the mean value of 10 runs.

**Performance.** Similar to most other mapping methods listed in Tab. B.1, the performance metrics of interest are the SWAP count and the execution time.

## 5.2 Benchmark selection

Three commonly evaluated benchmark sets of quantum programs were chosen for the mapping task, MQT Bench [78], RevLib [79] and QUEKO [80]; they have been used by many recently developed mapping methods Tab. B.1. In this study we are mostly interested in quantum programs, which have a realistic chance to run on a near-term quantum device, i.e quantum programs which comply with the limits of the inequality constraint (2.9). Applying this filter to the named benchmark sets yields 261 quantum programs, which are within the thresholds of maximum 20 qubits and 50 two-qubit gates.

*Remark.* All circuits are decomposed into the native gate set of the chosen hardware, thus there is only one type of two-qubit gate apparent in all quantum programs, the CX-gate.

The respective distributions of benchmark programs is shown in Figure 5.2.

In order to know, which benchmarks can be located where at the previous plots, Fig. 5.3 shows the number of qubits involved in the benchmark circuits over the number of CX-gates.

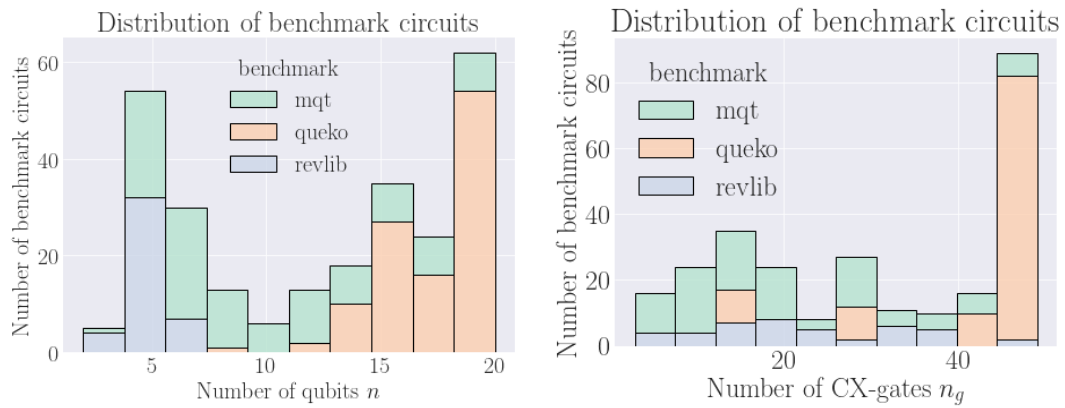


Figure 5.2: The graphs on the left and right display the distribution of benchmark circuits over the number of qubits they comprise, and the distribution of benchmark circuits over the number of CX-gates they comprise, respectively. In total, mqt, queko and revlib make up 51%, 18% and 31%, respectively.

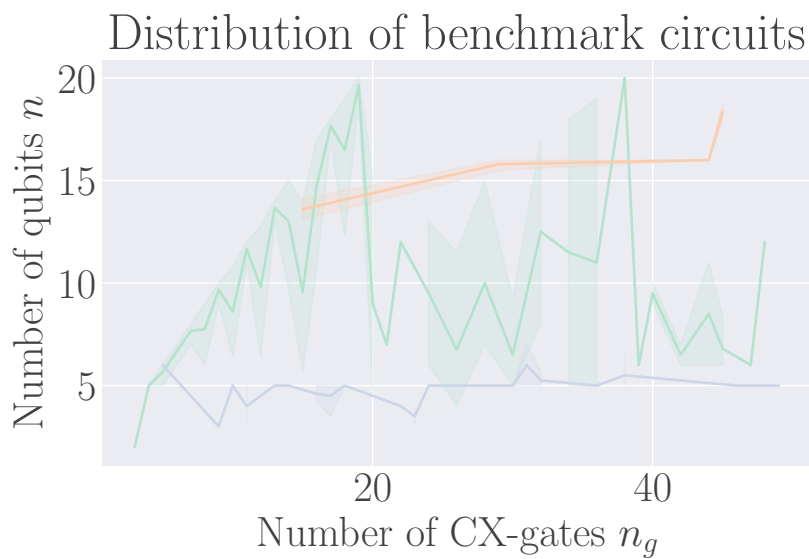


Figure 5.3: Number of qubits involved in the benchmark circuits over the number of CX-gates.

## 5.3 Results by SWAP Count

### 5.3.1 Experiment 1

In the first experiment, where we set the scaling factor  $f = 10$ , the method could successfully find assignments for 114 benchmark circuits, i.e. 44% of the total number of benchmark circuits evaluated.

Fig. 5.4 shows the overall SWAP-count (mean and standard deviation) grouped by the number of qubits  $n$  in a circuit and the three reference mappings of Qiskit (basic, stochastic and SABRE) in boxplots. Further the plots in Fig. 5.5 display the SWAP-count difference between the QUBO method and the Qiskit methods. All of the graphs show clearly that the upper bound of SWAPs for the QUBO method cannot compete with the Qiskit methods, hence it results in worse outcomes for almost all benchmark circuits and the difference grows in favor of the Qiskit methods with the number of qubits. In the worst case, the benchmark circuit with 19 qubits result for the Qiskit methods in 110 to 120 SWAPs less than in the QUBO mapping. In the best case, the upper bound is about the same or a little bit better than outcomes of Qiskit basic and sabre for small sized circuits (2-5 qubits). However, if the approximate algorithm could always find the best solution (which it cannot guarantee of course), the reference method stochastic and basic perform worse than the introduced method by on average 4 and 8 additional SWAPs, respectively. Only the SABRE method could outperform the minimum bound in about 70% of the evaluations.

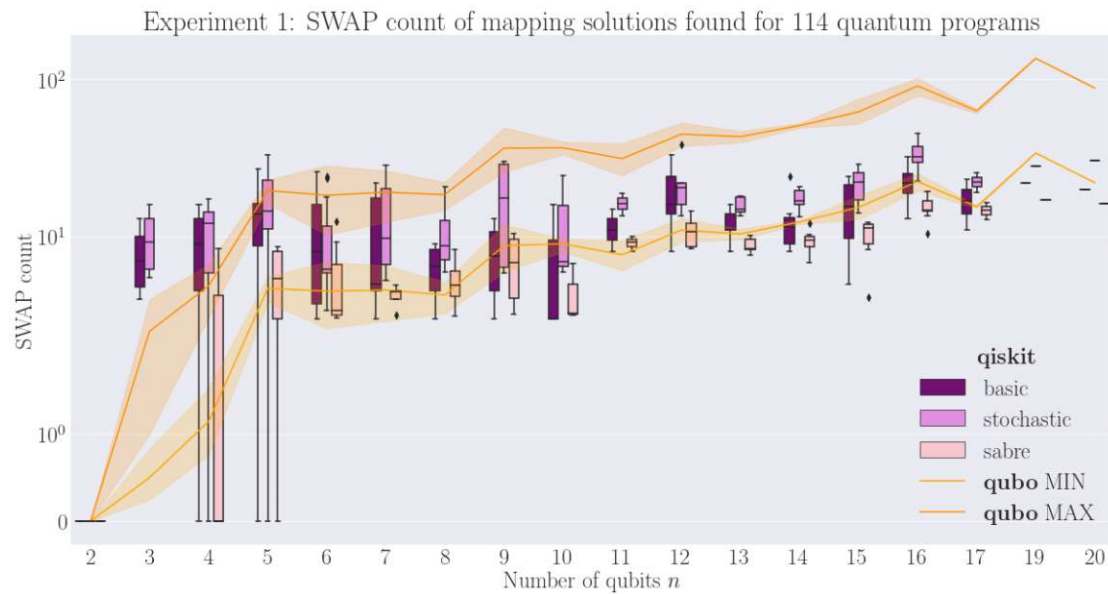


Figure 5.4: Number of additional SWAPs resulted from QUBO method (orange and yellow) and Qiskit basic, stochastic and SABRE mapping (boxplots) over the number of logical qubits  $n$ , in which the 114 successful mappings are grouped in.

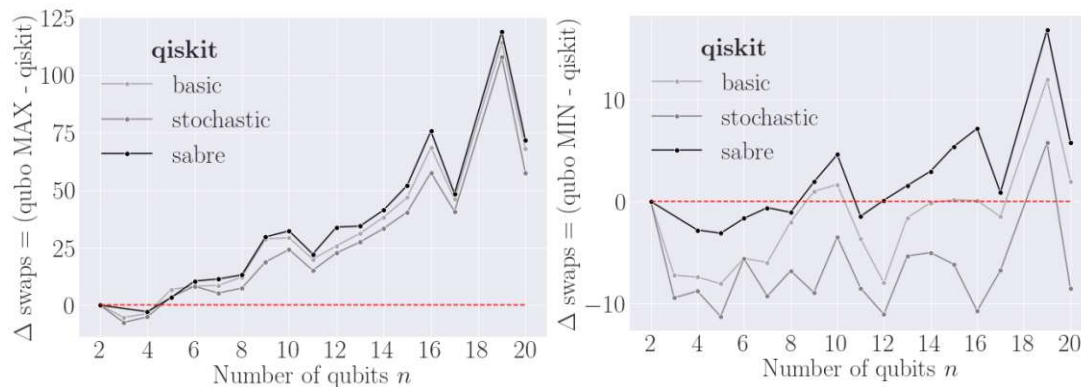


Figure 5.5: Left: Difference of average SWAP counts between QUBO MAX and Qiskit methods. Right: Difference of average SWAP count between QUBO MIN and Qiskit methods over the number of qubits of the benchmark set; all data points below the dashed zero-line signify a better outcome of the QUBO method.

When we look at the experiment from the perspective of CX-gates, we can see in the plots of Fig. 5.6 and Fig. 5.7 the overall same performance outcome. However, while the difference is clearly growing in favor of the Qiskit methods with the number of qubits (plot on the left in Fig. 5.5), this is not the case for

the number of CX-gates. This implies, the number of gates does not have a worse effect on the QUBO method compared to the reference methods. When we look at the minimum bound on the right of Fig. 5.5 and Fig. 5.7, we can see that the difference in SWAP-count does not depend on the number of qubits; however, the higher the number of CX-gates, the difference grows in favor of the QUBO method.

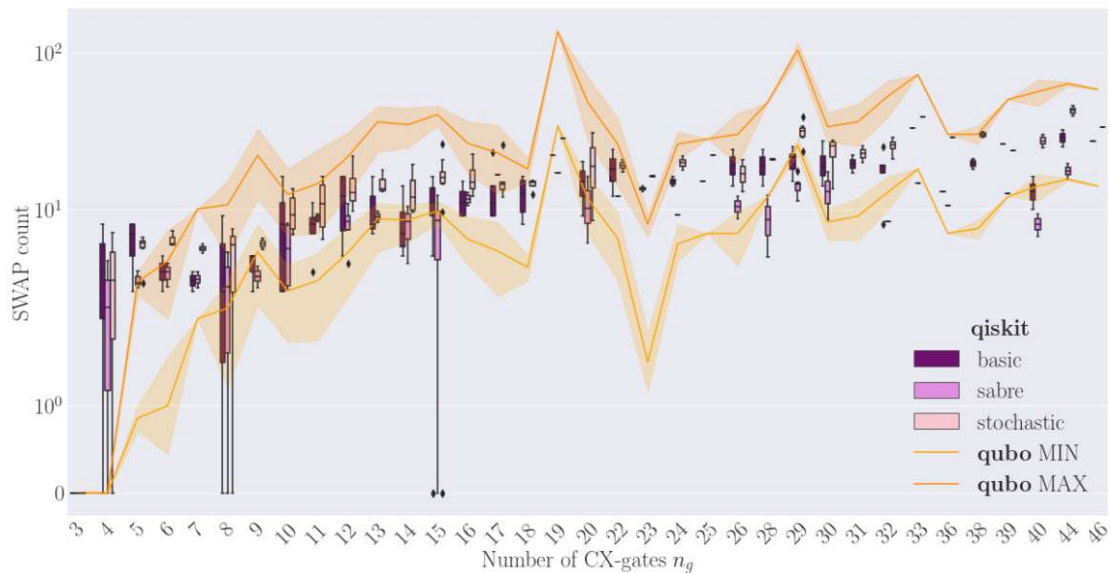


Figure 5.6: Number of additional SWAPs resulted from QUBO method (orange and yellow) and Qiskit basic, stochastic and SABRE mapping (boxplots) over the number of CX-gates  $n_g$ , in which the 114 successful mappings are grouped.

The results in figures for the second experiment are listed in the Appendix Sec. C.1.

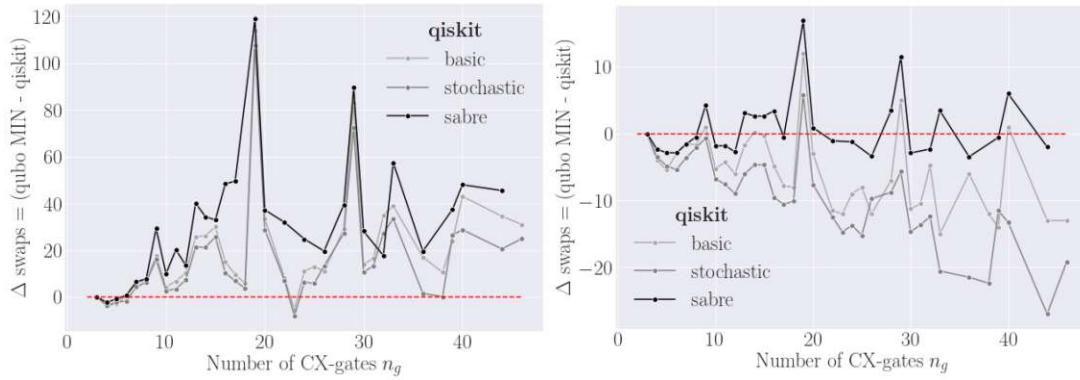


Figure 5.7: Left: Difference of average SWAP counts between QUBO MAX and Qiskit methods. Right: Difference of average SWAP count between QUBO MIN and Qiskit methods over the number of CX-gates of the benchmark set; all data points below the dashed zero-line signify a better outcome of the QUBO method.

### 5.3.2 Experiment 2

The second experiment, where  $f = 100$ , the introduced method found mapping solutions for 62 benchmark circuits, i.e. 24% of the total benchmark set. In this experiment an additional fourth reference method *qmap* [10] is evaluated for circuits up to 12 qubits (for unknown reasons, *qmap* could not compile the QUEKO benchmark set, which makes up all circuits with 13 and 14 qubits in the set).

Again, grouped by the number of qubits  $n$  in a circuit, Figure 5.8 shows the overall results in mean and standard deviation of the QUBO SWAP counts, compared to the four reference mappings, again Qiskit (basic, stochastic and SABRE) in boxplots and additionally *qmap* marked in blue. Further, Figures in 5.9 display the respective difference between the number of SWAPs of the QUBO method and the reference methods. The plots show that in almost all evaluated cases the introduced QUBO method yields better results for all circuit sizes. The worst performer for the Qiskit method were the group with  $n = 12$  qubits. It took the methods on average 26 SWAPs, while the QUBO maximum bound lies at 8 SWAPs. Even more striking is the result for the largest circuit size in terms of qubits: the average SWAP count for the Qiskit method is 15 SWAPs, while the QUBO method finds the optimal solution with zero SWAPs. Overall the maximum bound for the QUBO method yields on average 11 SWAPs less than its counterparts, it is further able to find the optimal solution in 75% of the evaluated circuits, while the reference methods could find it in only 16% (Qiskit) and 40% (*qmap*) of the cases.



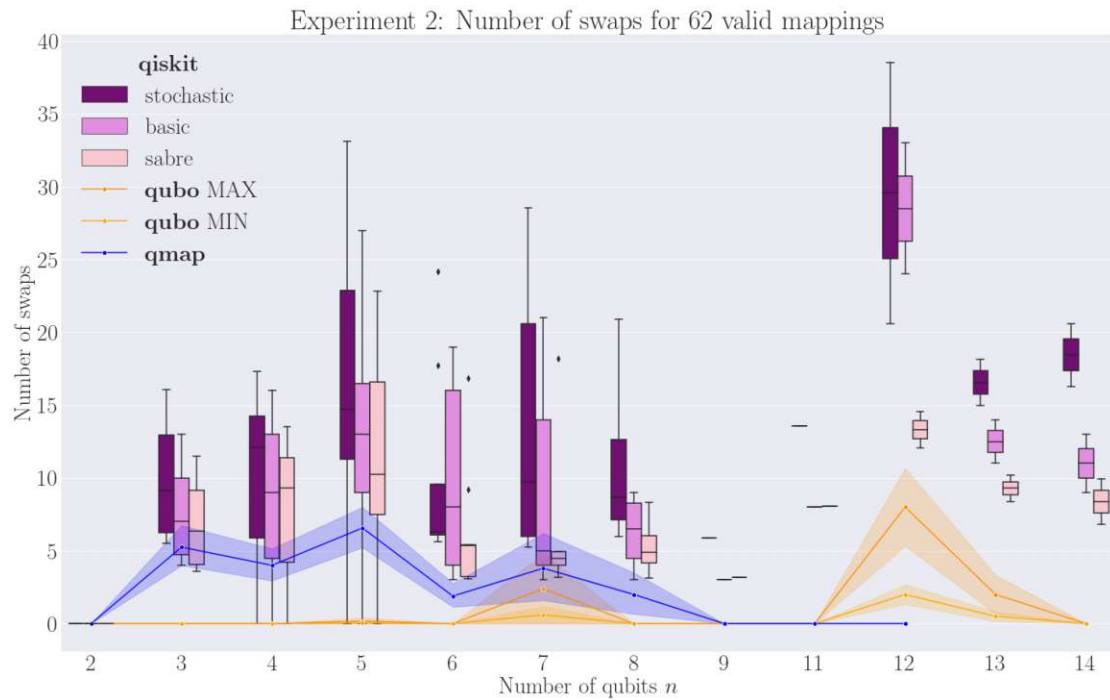


Figure 5.8: The number of SWAPs from the QUBO method plotted as orange line-plot, qmap as the blue lineplot and Qiskit basic, stochastic and SABRE mapping displayed as boxplots over the 62 valid mappings grouped in number of logical qubits  $n$  in the circuit.

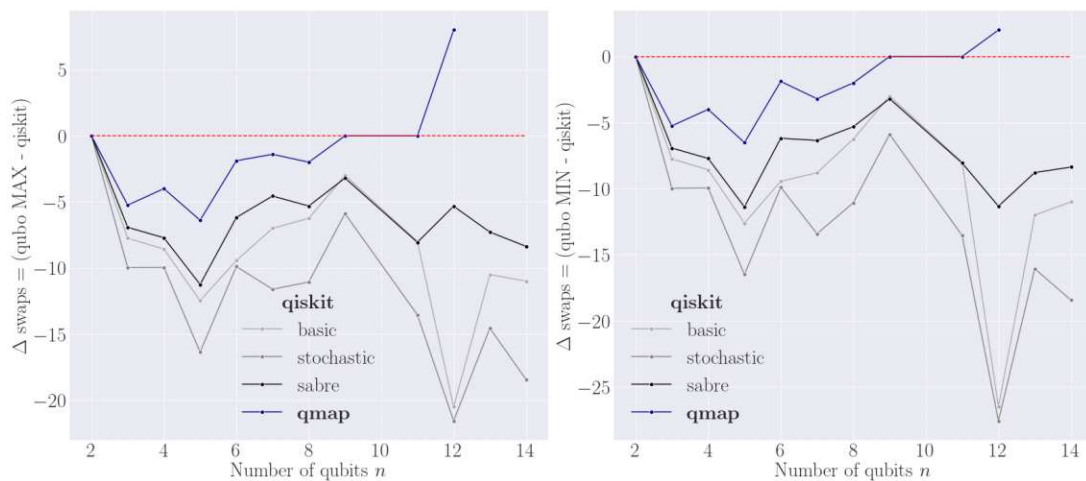


Figure 5.9: Left: Difference of average SWAP count between QUBO MAX and Qiskit methods. Right: Difference of average SWAP count between QUBO MIN and Qiskit methods; all data points below the dashed zero-line signify a better outcome for the introduced QUBO method.

Examining the results from the perspective of the number of CX-gates in the circuits gives an even brighter picture. Fig. 5.10 shows the overall results in mean and standard deviation of the QUBO SWAP counts, compared to the four reference mappings, again Qiskit (basic, stochastic and SABRE) in boxplots and additionally qmap marked in blue; however this time the benchmark programs are grouped in the number of CX-gates they contain. Further the plots in Fig. 5.11 display the respective difference between the number of SWAPs of the QUBO method and the reference methods. The results show clearly that with the numbers of gates the difference in SWAP-count is in favor of the introduced QUBO method for both the minimum and the maximum bound case. With the CX-gate grouping, the best outcome is also the largest circuit in terms of CX-gates: 19 to 30 SWAPs less are used by QUBO method's upper bound compared to the reference methods.

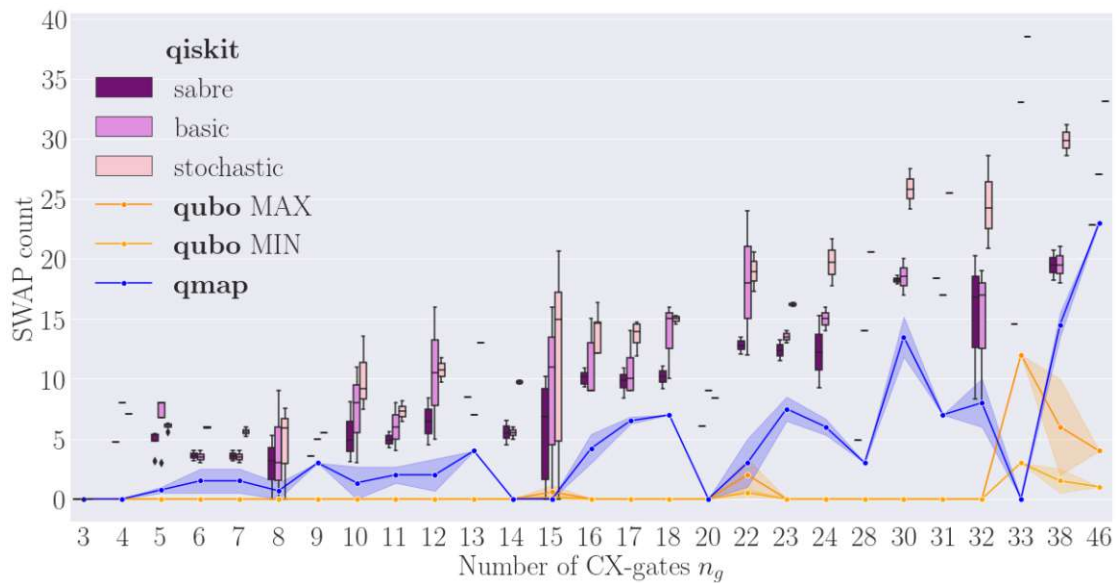


Figure 5.10: The number of SWAPs from the QUBO method plotted as orange line-plot, qmap as the blue lineplot and Qiskit basic, stochastic and SABRE mapping displayed as boxplots over the 62 valid mappings grouped in number of CX-gates  $n_g$  in the circuit.

The results in figures for the second experiment are listed in the Appendix Sec. C.2.

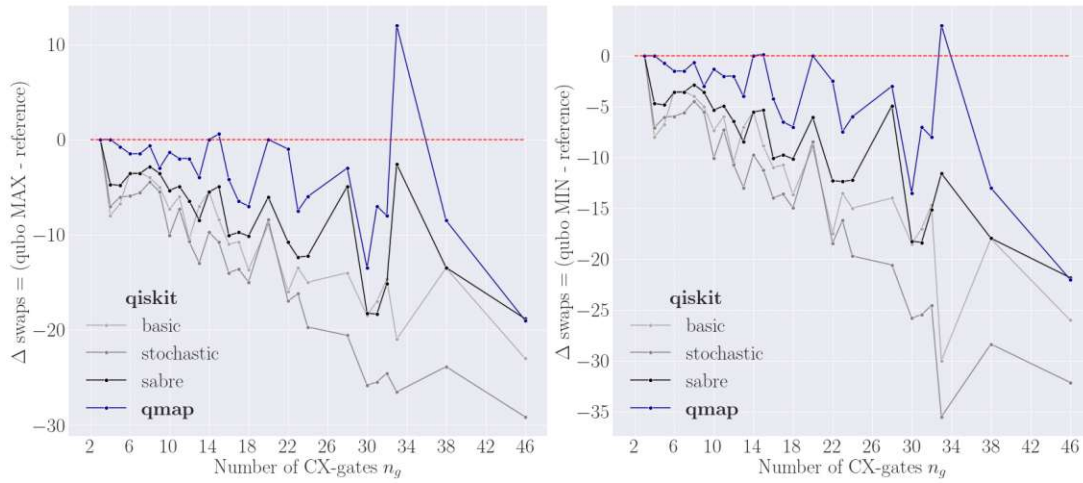


Figure 5.11: Left: Difference of average SWAP count between QUBO MAX and reference methods grouped by the number of CX-gates in a circuit. Right: Difference of average SWAP count between QUBO MIN and reference methods.; all data points below the dashed zero-line signify a better outcome for the introduced QUBO method.

## 5.4 Results by Execution Time

Turning to the second performance measure, we average over both experiments (since changing  $\lambda$  does not have an impact on the runtime) and display the execution times for the introduced QUBO method as well as for the reference methods in Figures 5.12 and 5.13. The plots clearly show the significant better performance of the reference methods. While both for Qiskit and qmap the overall average runtime stays below 0.05 seconds, the QUBO mapping runs 31 seconds on average, i.e. almost 620 times longer. In the maximum case for the largest circuit, the QUBO method executes 146 seconds, while Qiskit and qmap only need 0.03 seconds and 0.05 seconds to find a solution, respectively. Further, while the reference method stay constant over the considered range of qubits, the qubo method's execution time grows significantly with the number of qubits (about  $n^{1.5}$ ). Please refer to the detailed results in the appendix Sec. C.2.

The reference methods thus have a stable and much better runtime, 700 times on average; as such, this aspect of the algorithm presented exhibits a weakness in this regard. However, due to material constraints discussed in [Sec. 2.7.1](#), execution time on a quantum computer is expected to remain much more expensive than execution time on classical computer in the near future (which we know, is the main motivation to think about proper qubit mapping in the first place). At present, the consensus is still that it is better to spend several seconds on classical hardware

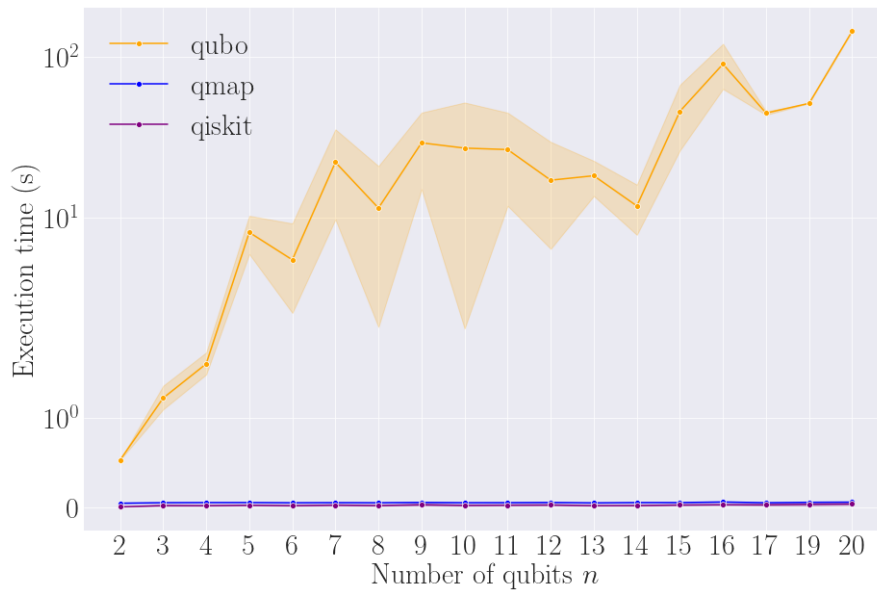


Figure 5.12: The execution time for the chosen benchmark set for the QUBO method in orange, and the reference methods qmap and Qiskit in blue and purple, respectively.

than to use even milliseconds more on a quantum computer [14]. For example, if one takes the trapped-ion technology specifications from Tab. 2.5, executing five CX-gates requires about one millisecond. Indeed, taking into account the results of the second experiment in this thesis, at least 11 CX-gates on average can be spared using 15 seconds on average more classical runtime; therefore, one can consider the method still as a relevant improvement.

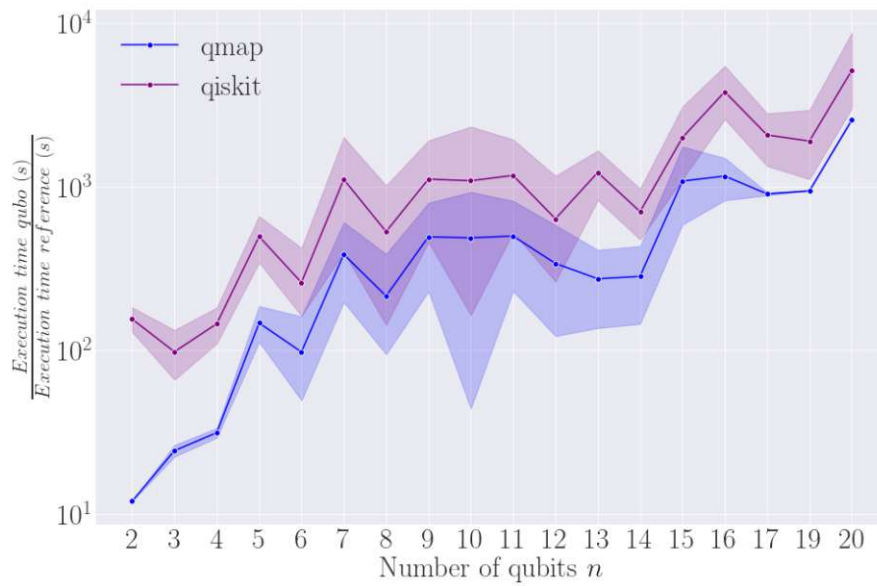


Figure 5.13: The execution time fraction for the chosen benchmark set; QUBO execution time is divided by reference methods' execution time qmap and Qiskit in blue and purple, respectively. It answers the question "How much longer does the QUBO method run compared to the reference methods?" - "[data point] times longer on average."

## Chapter 6

# Conclusion

This thesis has introduced a new method for solving the qubit mapping problem, based on a QUBO formulation incorporating the concepts of subgraph isomorphism and token swapping. The mapping involves dividing the circuit into smaller pieces, and is later improved upon with a genetic algorithm. The naive approach of putting only one CX-gate in each slice, for which there always exists a direct mapping to the coupling graph, the QUBO is an exact formulation, for which a solution always exists. Solving this problem in an exact manner gives the best possible solution, i.e. it represents a lower bound to the problem. In practice the QUBO is solved heuristically (classically with simulated annealing in the present work, or even on a quantum annealer). This is beneficial, since new and better heuristics are developed continuously; such that the best solving strategy available can be promptly deployed, since the QUBO itself is not dependent on the method used to solve it. Solving the QUBO is equivalent to finding the parameter matrix's minimum binary eigenvector, which represents the solution and in this case the mapping for each slice.

The results by execution time reflect the polynomial growth of the problem size  $N = n \cdot n' \cdot m$ , which is not desirable. Other implemented mapping algorithms have a stable and much better runtime (700 times on average for the references compared to in the experiments); However, as already mentioned earlier due to material constraints, execution time on a quantum computer is expected to remain much more expensive than execution time on classical computer in the near future. It is still thus the consensus that it is better to spend several seconds on classical hardware than to use even milliseconds more on a quantum computer [14]. Therefore, taking into account the significant reduction in SWAP-count outlined in Sec. 5.3.2, one can consider the method as a relevant improvement, despite lower runtimes showcased by other reference methods.

In contrast to most other mapping methods, however, the QUBO method introduced is not able to guarantee a mapping solution; hence, the search strategy is not exact, and therefore finding the minimum of the cost function cannot be

deterministic, even with no penalization for swaps, i.e.  $\lambda = 0$ . The success rate, i.e. how often the method finds a mapping solution for a given problem divided by the number of trials, is vitally dependent on the penalization multiplier  $\lambda$ , as we saw in the experiments in Sec. 5.3.2. A factor of 10 difference in the scaling factor  $f$  diminished the success rate of finding a mapping solution from 44% to 24%. Further investigations have to be made in order to improve  $\lambda$ , such that the success rate reaches an acceptable level (i.e. over 95%).

Taking stock of the advantages and disadvantages for this QUBO approach for qubit mapping, one can expect that this method exhibits potential for future qubit mapping tasks; hence it already demonstrates promising results for found mapping solutions, which are significantly better than what state-of-the-art methods achieve. Even though the success rate of finding a solution and runtime are still hurdles to overcome, there is sufficient reason to hope that matrix solving heuristics will eventually lead to a higher success rate and less runtime.

## Chapter 7

# Acknowledgements

I am grateful for the significant amount of support and assistance that I have received throughout the writing of this thesis. Firstly, I express my gratitude to Professor Sebastian Feld, my supervisor at Delft University of Technology, whose expertise was invaluable in formulating the research questions and methodology. His continuous feedback pushed me to sharpen my thinking and broaden my understanding. Secondly, I would like to thank my supervisor Professor Nysret Musliu from the Technical University of Vienna for his consistent support. I am grateful that I could conduct this research project due to the mutual consent of the Delft University of Technology and the Technical University of Vienna. Additionally, I received great guidance from my colleagues Jonas Nüklein, Medina Bandic and Matthew Steinberg, who provided me with indispensable insights needed to successfully complete my master thesis. Lastly, I am thankful to my parents for their unwavering support and confidence in me and my abilities throughout my years of study.



## Appendix A

# Quantum Fourier Transform

The quantum version of the Fourier transformation takes a quantum state  $|x\rangle$  of  $N$  qubits and transforms it into another basis, the Fourier basis  $|y\rangle$ , which holds the information of the states frequency components. We follow the calculations of [27] and write two general quantum states, where one is the Fourier transform of the other

$$|x\rangle = \sum_{n=0}^{N-1} x_n |n\rangle \quad |y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle \quad (\text{A.1})$$

$$\text{where } y_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{i2\pi nk/N}. \quad (\text{A.2})$$

Note, that Eq. A.2 is just the formula for the classical inverse Fourier transform. To keep things (relatively) simple, we set  $N = 2^n$ , where  $n \in \mathbb{N}_+$  and assume  $|x\rangle$  is a basis state; then

$$y_k = \frac{1}{\sqrt{N}} e^{i2\pi xk/N} \quad \text{where } x \in [0, N]. \quad (\text{A.3})$$

The goal is to find a practical unitary transformation  $U$ , yielding

$$U |x\rangle = |y\rangle. \quad (\text{A.4})$$

We insert Eq. A.3 into Eq. A.1, which gives

$$U |x\rangle = \sum_{k=0}^{N-1} y_k |k\rangle \quad (\text{A.5})$$

$$= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{i2\pi xk/N} |k\rangle. \quad (\text{A.6})$$

The next crucial step is to expand  $|k\rangle$  into its binomial basis; which takes the coefficients  $k$  as binomial numbers.

$$\sum_{k=0}^{N-1} |k\rangle \longrightarrow \sum_{k_1=0}^1 \dots \sum_{k_N=0}^1 |k_1 k_2 \dots k_N\rangle \quad (\text{A.7})$$

$$k \longrightarrow k_1 2^{n-1} + k_2 2^{n-2} + \dots + k_n 2^0. \quad (\text{A.8})$$

Inserting the above expansion into Eq. A.6 then reads

$$U |x\rangle = \frac{1}{\sqrt{N}} \sum_{k_1=0}^1 \dots \sum_{k_N=0}^1 e^{i2\pi x/N \sum_l k_l 2^{n-l}} |k_1 k_2 \dots k_N\rangle \quad (\text{A.9})$$

$$= \frac{1}{\sqrt{N}} \prod_{l=1}^n \sum_{k_l=0}^1 e^{i2\pi x k_l 2^{n-l}/N} |k_1 k_2 \dots k_N\rangle \quad (\text{A.10})$$

$$= \frac{1}{\sqrt{2^n}} \prod_{l=1}^n \sum_{k_l=0}^1 e^{i2\pi x k_l 2^{-l}} |k_1 k_2 \dots k_N\rangle. \quad (\text{A.11})$$

Hence  $|k_1 k_2 \dots k_n\rangle$  are basis states, i.e.  $k_l = 1$ , then  $k_i = 0$  ( $i \neq l$ ), we can rewrite Eq. A.11 to

$$= \frac{1}{\sqrt{2^n}} (|0\rangle + e^{i2\pi x 2^{-1}} |1\rangle) \otimes (|0\rangle + e^{i2\pi x 2^{-2}} |1\rangle) \otimes \dots \otimes (|0\rangle + e^{i2\pi x 2^{-n}} |1\rangle). \quad (\text{A.12})$$

From here, we can already determine the circuit. Eq. A.12 can be computed with Hadamarts and controlled rotations.

## Appendix B

# Related Work

In the following Table B.1 a comprehensive overview of recently developed mapping methods is given, it includes their acronym (if given), strategy, benchmarks used for evaluation, quantum hardware used for evaluation, programming language, performance metric and references the authors compared their method to.

Reference	Name	Strategy	Benchmark	Quantum Hardware	Language	Metric	Comparison
[11] 2020, pre-print	DQN	Heuristic, Reinforcement Learning	MQT, RevLib ca 150 samples 5-50 qubits 30-40 cxgates	5 different, incl. IBM Tokyo	Python	depth	qiskit stochastic, tket
[9] 2022, IEEE	Qmap	Heuristic, search algorithm & ILP for init mapping	MQT, QUEKO ca 50 samples 3-16 qubits 5 to 64,283 gates (w/ 2-30% cx-gates)	Surface 17, IBM Tokyo	Python	#swaps	-
[8] 2019 ACM	SABRE	Heuristic, look-ahead, w/ heuristic cost function	RevLib ca 30 samples 5-11 qubits 21-34 000 cxgates	IBM Tokyo	Python	#swaps, runtime	qmap
[15] 2019 IEEE		Exact, Satisfiability problem	RevLib 2-8 qubits 4-20 cx-gates	IBM QX 4	C++	#swaps, runtime	SABRE
[81] 2019 Conference paper	tket	Heuristic, slicing, look-ahead	MQT ca 160 samples 6-16 qubits 7-7mio gates (? cxgates)	5 different, incl. IBM Tokyo	Python	#swaps	SABRE, qmap
[82] 2019 ACM	BMT	Heuristic, Subgraph isomorphism w/ greedy search, Token swapping w/ approximate algorithm	RevLib, Quipper, ao. ca 160 samples	IBM Tokyo, IBM QX3	C++	depth, runtime, memory	Qiskit, SABRE, qmap
[77] 2021 IEEE	SAHS	Heuristic, subgraph isomorphism, depth search	RevLib, MQTT ca 30 samples 5-16 qubits 11-15.000 cxgates	3 different, including Tokyo	Python	#swaps, runtime	SABRE, qmap
[10] 2019 IEEE	qmap	Heuristic, slicing, A*, look-ahead	RevLib, MQTT ca50 samples 2-16 qubits 20-513 000 gates	4 different IBM Q	C++	#swaps, runtime	Qiskit

Table B.1: Specifications of a selection of relevant mapping methods.

## Appendix C

# Results

### C.1 Experiment 1

# qubits $n$	QUBO	SWAP-count	time (s)	Qiskit	SWAP-count		Qiskit	time (s)	
	MIN	MAX		basic	sabre		stochastic	basic	
2	0.00	0.00	0.53	0.00	0.00	0.00	0.00	0.00	0.00
3	0.50	2.50	1.28	7.75	NaN	9.98	0.02	NaN	0.03
4	1.14	4.86	1.56	8.57	2.81	9.96	0.02	0.01	0.03
5	4.69	19.62	6.96	12.78	4.99	16.00	0.02	0.01	0.04
6	4.50	18.33	5.32	10.08	5.52	10.13	0.02	0.01	0.02
7	4.57	19.14	14.81	10.57	4.22	13.88	0.02	0.01	0.04
8	4.25	18.50	11.43	6.25	5.32	11.07	0.02	0.01	0.02
9	8.75	36.50	29.19	7.75	6.83	17.71	0.03	0.01	0.03
10	9.00	36.67	27.06	7.33	4.39	12.49	0.03	0.01	0.02
11	7.67	31.33	26.50	11.33	9.13	16.19	0.02	0.01	0.03
12	11.00	44.80	17.06	19.00	10.88	22.11	0.03	0.01	0.03
13	10.40	43.20	18.19	12.00	8.84	15.77	0.02	0.01	0.02
14	12.38	50.75	11.78	12.50	9.41	17.40	0.02	0.01	0.03
15	15.33	62.00	45.96	15.17	9.93	21.52	0.03	0.01	0.03
16	22.40	90.80	90.67	22.30	15.22	33.17	0.03	0.01	0.04
17	15.50	63.00	44.79	17.00	14.63	22.28	0.03	0.01	0.04
19	34.00	136.00	51.54	22.00	17.12	28.21	0.05	0.02	0.03
20	22.00	88.00	145.57	20.00	16.21	30.56	0.05	0.02	0.04

Table C.1: Mean values of SWAP-count and execution time calculated over the set of benchmark circuits grouped by the number of qubits  $n$ .

# CX-gates $n_g$	QUBO	SWAP-count	time (s)	Qiskit	SWAP-count	stochastic	Qiskit	time (s)	stochastic
	MIN	MAX		basic	sabre		basic	sabre	
3	0.00	0.00	0.53	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	1.39	4.00	2.36	3.53	0.01	0.01	0.01
5	0.86	3.43	1.38	6.29	3.52	5.69	0.01	0.01	0.01
6	1.00	4.67	3.01	4.00	3.89	6.38	0.01	0.01	0.02
7	2.00	10.00	1.39	3.50	3.57	5.60	0.01	0.01	0.02
8	2.33	10.67	2.26	4.00	2.83	4.47	0.01	0.01	0.01
9	5.33	22.00	1.52	4.33	3.74	6.00	0.01	0.01	0.02
10	3.00	12.40	4.49	8.20	5.57	9.78	0.02	0.01	0.02
11	3.50	14.50	1.75	7.75	8.80	11.08	0.02	0.01	0.02
12	5.25	21.50	2.38	11.25	7.99	14.24	0.03	0.01	0.03
13	8.67	36.00	11.20	10.33	8.93	14.70	0.02	0.01	0.03
14	8.50	34.50	8.83	8.25	7.71	13.18	0.02	0.01	0.03
15	9.71	40.00	9.93	10.07	7.08	14.38	0.02	0.01	0.02
16	6.43	26.29	12.41	11.29	11.61	15.94	0.02	0.01	0.03
17	5.40	22.80	11.26	13.20	16.51	15.99	0.02	0.01	0.04
18	4.25	18.00	2.79	12.25	NaN	14.32	0.02	NaN	0.04
19	34.00	136.00	51.54	22.00	17.12	28.21	0.05	0.02	0.03
20	11.67	48.00	70.86	14.67	10.78	19.27	0.03	0.01	0.04
22	6.50	26.00	6.67	18.00	12.09	18.95	0.03	0.02	0.05
23	1.50	8.00	1.68	13.50	NaN	16.21	0.03	NaN	0.04
24	6.00	26.00	15.73	15.00	9.23	19.70	0.03	0.01	0.04
25	7.00	28.00	6.09	15.00	NaN	22.21	0.02	NaN	0.06
26	7.00	30.00	18.55	19.00	10.40	16.73	0.04	0.01	0.04
28	12.00	48.00	74.24	19.00	8.57	20.78	0.04	0.01	0.04
29	25.50	103.50	45.85	20.50	14.02	31.20	0.03	0.01	0.03
30	8.25	33.50	38.30	19.50	12.90	22.94	0.03	0.02	0.06
31	9.00	36.00	25.33	19.50	NaN	22.66	0.03	NaN	0.06
32	12.80	52.40	16.20	17.60	8.31	25.19	0.03	0.02	0.06
33	18.00	72.00	65.26	33.00	14.57	38.53	0.04	0.02	0.04
36	7.00	30.00	65.96	13.00	10.46	28.47	0.04	0.01	0.03
38	7.50	30.00	42.25	19.50	NaN	29.87	0.03	NaN	0.08
39	12.00	50.00	35.26	26.00	12.57	23.51	0.04	0.02	0.05
40	14.00	56.00	61.34	13.00	8.04	27.27	0.05	0.02	0.05
44	15.50	63.00	246.07	28.50	17.50	42.46	0.04	0.02	0.04
46	14.00	58.00	10.95	27.00	NaN	33.14	0.04	NaN	0.07

Table C.2: Mean values of SWAP-count and execution time calculated over the set of benchmark circuits grouped by the number of CX-gates  $n_g$ .

## C.2 Experiment 2

# qubits $n$	QUBO	SWAP-count	time (s)	qmap	time(s)	Qiskit	SWAP-count			Qiskit	time (s)	
	MIN	MAX				basic	sabre	stochastic	basic	sabre	stochastic	
2.0	0.00	0.00	0.61	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3.0	0.00	0.00	1.32	5.25	0.05	7.75	6.93	9.98	0.02	0.01	0.03	0.03
4.0	0.00	0.00	1.57	4.00	0.05	8.57	7.72	9.96	0.02	0.01	0.03	0.03
5.0	0.04	0.17	26.92	6.57	0.05	12.65	11.42	16.54	0.02	0.01	0.04	0.04
6.0	0.00	0.00	9.46	1.89	0.05	9.44	6.18	9.89	0.02	0.01	0.02	0.02
7.0	0.60	2.40	10.54	3.80	0.05	9.40	6.96	14.01	0.02	0.01	0.03	0.03
8.0	0.00	0.00	16.10	2.00	0.05	6.25	5.32	11.07	0.02	0.01	0.02	0.02
9.0	0.00	0.00	12.74	0.00	0.04	3.00	3.20	5.89	0.02	0.01	0.01	0.01
11.0	0.00	0.00	5.22	0.00	0.04	8.00	8.06	13.55	0.01	0.01	0.02	0.02
12.0	2.00	8.00	52.75	0.00	0.06	28.50	13.33	29.56	0.04	0.02	0.05	0.05
13.0	0.50	2.00	8.53	NaN	NaN	12.50	9.28	16.56	0.02	0.01	0.02	0.02
14.0	0.00	0.00	14.27	NaN	NaN	11.00	8.37	18.44	0.01	0.01	0.02	0.02

Table C.3: Mean values of SWAP-count and execution time calculated over the set of benchmark circuits grouped by the number of qubits  $n$ .

# qubits $n$	QUBO	SWAP-count	time (s)	qmap	time(s)	Qiskit	SWAP-count			Qiskit	time (s)	
	MIN	MAX				basic	sabre	stochastic	basic	sabre	stochastic	
3.0	0.00	0.00	0.61	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4.0	0.00	0.00	0.68	0.00	0.05	8.00	4.71	7.06	0.01	0.01	0.01	0.01
5.0	0.00	0.00	1.18	0.75	0.04	6.75	4.82	6.04	0.01	0.01	0.01	0.01
6.0	0.00	0.00	1.23	1.50	0.04	3.50	3.58	5.95	0.01	0.01	0.02	0.02
7.0	0.00	0.00	14.59	1.50	0.05	3.50	3.57	5.60	0.01	0.01	0.02	0.02
8.0	0.00	0.00	8.32	0.67	0.05	4.00	2.83	4.47	0.01	0.01	0.01	0.01
9.0	0.00	0.00	1.12	3.00	0.05	5.00	3.57	5.50	0.01	0.01	0.02	0.02
10.0	0.00	0.00	2.63	1.33	0.05	7.33	5.34	10.08	0.01	0.01	0.02	0.02
11.0	0.00	0.00	1.17	2.00	0.05	6.00	4.92	7.30	0.02	0.01	0.02	0.02
12.0	0.00	0.00	1.59	2.00	0.05	10.50	6.46	10.73	0.03	0.01	0.03	0.03
13.0	0.00	0.00	1.80	4.00	0.05	7.00	8.46	12.98	0.01	0.01	0.03	0.03
14.0	0.00	0.00	17.68	0.00	0.05	5.50	5.52	9.75	0.02	0.01	0.03	0.03
15.0	0.14	0.57	7.24	0.00	0.05	9.00	5.50	11.37	0.01	0.01	0.02	0.02
16.0	0.00	0.00	2.20	4.20	0.05	11.00	10.09	14.00	0.02	0.01	0.03	0.03
17.0	0.00	0.00	28.43	6.50	0.05	10.75	9.76	13.61	0.02	0.01	0.04	0.04
18.0	0.00	0.00	40.42	7.00	0.05	13.67	10.14	14.98	0.02	0.01	0.04	0.04
20.0	0.00	0.00	1.97	0.00	0.05	9.00	6.06	8.42	0.02	0.01	0.02	0.02
22.0	0.50	2.00	20.37	3.00	0.05	18.00	12.80	18.95	0.03	0.01	0.05	0.05
23.0	0.00	0.00	1.81	7.50	0.05	13.50	12.37	16.21	0.03	0.01	0.04	0.04
24.0	0.00	0.00	26.32	6.00	0.05	15.00	12.24	19.70	0.03	0.01	0.04	0.04
28.0	0.00	0.00	18.53	3.00	0.05	14.00	4.92	20.58	0.03	0.01	0.03	0.03
30.0	0.00	0.00	3.10	13.50	0.06	18.50	18.24	25.82	0.02	0.01	0.06	0.06
31.0	0.00	0.00	60.96	7.00	0.06	17.00	18.36	25.47	0.03	0.01	0.06	0.06
32.0	0.00	0.00	53.23	8.00	0.06	14.67	15.14	24.57	0.03	0.01	0.06	0.06
33.0	3.00	12.00	66.69	0.00	0.06	33.00	14.57	38.53	0.04	0.02	0.04	0.04
38.0	1.50	6.00	50.41	14.50	0.06	19.50	19.45	29.87	0.03	0.01	0.08	0.08
46.0	1.00	4.00	84.57	23.00	0.07	27.00	22.83	33.14	0.04	0.01	0.07	0.07

Table C.4: Mean values of SWAP-count and execution time calculated over the set of benchmark circuits grouped by the number of CX-gates  $n_g$ .

# Bibliography

- [1] R. P. Feynman, “Simulating physics with computers,” *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467–488, 1982. [Online]. Available: <https://doi.org/10.1007/BF02650179>
- [2] D. Deutsch and R. Penrose, “Quantum theory, the church&#x2013;turing principle and the universal quantum computer,” *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985. [Online]. Available: <https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1985.0070>
- [3] P. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.
- [4] J. Preskill, “Quantum computing 40 years later,” 2021. [Online]. Available: <https://arxiv.org/abs/2106.10522>
- [5] D. A. Lidar and T. A. Brun, *Quantum error correction*. Cambridge university press, 2013.
- [6] P. W. Shor, “Scheme for reducing decoherence in quantum computer memory,” *Physical review A*, vol. 52, no. 4, p. R2493, 1995.
- [7] M. A. Steinberg, S. Feld, C. G. Almudever, M. Marthaler, and J.-M. Reiner, “Topological-Graph Dependencies and Scaling Properties of a Heuristic Qubit-Assignment Algorithm,” *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–14, 2022, conference Name: IEEE Transactions on Quantum Engineering.
- [8] G. Li, Y. Ding, and Y. Xie, “Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices,” May 2019, arXiv:1809.02573 [quant-ph]. [Online]. Available: <http://arxiv.org/abs/1809.02573>

- [9] L. Lao, H. van Someren, I. Ashraf, and C. G. Almudever, “Timing and resource-aware mapping of quantum circuits to superconducting processors,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 2, pp. 359–371, Feb. 2022, arXiv:1908.04226 [quant-ph]. [Online]. Available: <http://arxiv.org/abs/1908.04226>
- [10] A. Zulehner, A. Paler, and R. Wille, “An efficient methodology for mapping quantum circuits to the ibm qx architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2019.
- [11] M. G. Pozzi, S. J. Herbert, A. Sengupta, and R. D. Mullins, “Using Reinforcement Learning to Perform Qubit Routing in Quantum Compilers,” Jul. 2020, arXiv:2007.15957 [quant-ph]. [Online]. Available: <http://arxiv.org/abs/2007.15957>
- [12] A. Sinha, U. Azad, and H. Singh, “Qubit Routing Using Graph Neural Network Aided Monte Carlo Tree Search,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, pp. 9935–9943, Jun. 2022, number: 9. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/21231>
- [13] R. Wille, A. Lye, and R. Drechsler, “Exact Reordering of Circuit Lines for Nearest Neighbor Quantum Architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1818–1831, Dec. 2014. [Online]. Available: <https://ieeexplore.ieee.org/document/6951856>
- [14] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, “Qubit allocation as a combination of subgraph isomorphism and token swapping,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–29, Oct. 2019. [Online]. Available: <https://dl.acm.org/doi/10.1145/3360546>
- [15] R. Wille, L. Burgholzer, and A. Zulehner, “Mapping Quantum Circuits to IBM QX Architectures Using the Minimal Number of SWAP and H Operations,” Jul. 2019, arXiv:1907.02026 [quant-ph]. [Online]. Available: <http://arxiv.org/abs/1907.02026>
- [16] M. Bandic, H. Zarein, E. Alarcon, and C. G. Almudever, “On structured design space exploration for mapping of quantum algorithms,” in *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, 2020, pp. 1–6.



- [17] L. Prielinger, “Github qubit mapping.” [Online]. Available: <https://github.com/Luisenden/map-quantum-circuits-to-multi-core>
- [18] W. van Dam, M. Mosca, and U. Vazirani, “How powerful is adiabatic quantum computation?” in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, 2001, pp. 279–287.
- [19] N. G. Dickson, M. W. Johnson, M. H. Amin, R. Harris, F. Altomare, A. J. Berkley, P. Bunyk, J. Cai, E. M. Chapple, P. Chavez, F. Cioata, T. Cirip, P. deBuen, M. Drew-Brook, C. Enderud, S. Gildert, F. Hamze, J. P. Hilton, E. Hoskinson, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Lanting, T. Mahon, R. Neufeld, T. Oh, I. Perminov, C. Petroff, A. Przybysz, C. Rich, P. Spear, A. Tcaciuc, M. C. Thom, E. Tolkacheva, S. Uchaikin, J. Wang, A. B. Wilson, Z. Merali, and G. Rose, “Thermally assisted quantum annealing of a 16-qubit problem,” *Nat. Commun.*, vol. 4, p. 1903, 05 2013. [Online]. Available: <http://dx.doi.org/10.1038/ncomms2920>
- [20] C. S. Calude, M. J. Dinneen, and R. Hua, “Qubo formulations for the graph isomorphism problem and related problems,” *Theoretical Computer Science*, vol. 701, pp. 54–69, 2017, at the intersection of computer science with biology, chemistry and physics - In Memory of Solomon Marcus. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0304397517304590>
- [21] F. Neukart, G. Compostella, C. Seidel, D. von Dollen, S. Yarkoni, and B. Parney, “Traffic flow optimization using a quantum annealer,” *Frontiers in ICT*, vol. 4, 2017. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fict.2017.00029>
- [22] P. Date, D. Arthur, and L. Pusey-Nazzaro, “QUBO formulations for training machine learning models,” *Scientific Reports*, vol. 11, no. 1, p. 10029, May 2021, number: 1 Publisher: Nature Publishing Group. [Online]. Available: <https://www.nature.com/articles/s41598-021-89461-4>
- [23] S. Yarkoni, E. Raponi, T. Bäck, and S. Schmitt, “Quantum annealing for industry applications: introduction and review,” *Reports on Progress in Physics*, vol. 85, no. 10, p. 104001, Oct. 2022. [Online]. Available: <https://iopscience.iop.org/article/10.1088/1361-6633/ac8c54>
- [24] B. Dury and O. D. Matteo, “A qubo formulation for qubit allocation,” *arXiv: Quantum Physics*, 2020.
- [25] M. A. Nielsen and I. Chuang, “Quantum computation and quantum information,” 2002.

- [26] Bloch-kugel. [Online]. Available: <https://de.wikipedia.org/wiki/Bloch-Kugel>
- [27] V. Kasirajan, *Fundamentals of quantum computing*. Springer, 2021.
- [28] “RSA (cryptosystem),” Jan. 2023, page Version ID: 1134809193. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=RSA\\_\(cryptosystem\)&oldid=1134809193](https://en.wikipedia.org/w/index.php?title=RSA_(cryptosystem)&oldid=1134809193)
- [29] “A Resource Estimation Framework for Quantum Attacks Against Cryptographic Functions - Recent Developments.” [Online]. Available: <https://globalriskinstitute.org/publication/a-resource-estimation-framework-for-quantum-attacks-against-cryptographic-functions-recent->
- [30] M. Mosca and M. Piani, “2021 quantum threat timeline report,” *Global Risk Institute, Toronto, ON*, 2022.
- [31] D. R. Musk, “A comparison of quantum and traditional fourier transform computations,” *Computing in Science & Engineering*, vol. 22, no. 6, pp. 103–110, 2020.
- [32] M. Saeedi, R. Wille, and R. Drechsler, “Synthesis of quantum circuits for linear nearest neighbor architectures,” *Quantum Information Processing*, vol. 10, pp. 355–377, 2011.
- [33] M. Smelyanskiy, N. P. Sawaya, and A. Aspuru-Guzik, “qhipster: The quantum high performance software testing environment,” *arXiv preprint arXiv:1601.07195*, 2016.
- [34] “NVIDIA cuQuantum,” Apr. 2021. [Online]. Available: <https://developer.nvidia.com/cuquantum-sdk>
- [35] N. Khammassi, I. Ashraf, J. Someren, R. Nane, A. Krol, M. A. Rol, L. Lao, K. Bertels, and C. G. Almudever, “Openql: A portable quantum programming framework for quantum accelerators,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 1, pp. 1–24, 2021.
- [36] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, “A software methodology for compiling quantum programs,” *Quantum Science and Technology*, vol. 3, no. 2, p. 020501, 2018.
- [37] A. Cross, “The ibm q experience and qiskit open-source quantum computing software,” in *APS March meeting abstracts*, vol. 2018, 2018, pp. L58–003.

- [38] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, “Q# enabling scalable quantum computing and development with a high-level dsl,” in *Proceedings of the real world domain specific languages workshop 2018*, 2018, pp. 1–10.
- [39] “Cirq | Google Quantum AI.” [Online]. Available: <https://quantumai.google/cirq>
- [40] N. Schetakakis, D. Aghamalyan, P. Griffin, and M. Boguslavsky, “Review of some existing qml frameworks and novel hybrid classical–quantum neural networks realising binary classification for the noisy datasets,” *Scientific Reports*, vol. 12, no. 1, pp. 1–12, 2022.
- [41] S. Resch and U. R. Karpuzcu, “Quantum computing: an overview across the system stack,” *arXiv preprint arXiv:1905.07240*, 2019.
- [42] M. A. Serrano, J. A. Cruz-Lemus, R. Perez-Castillo, and M. Piattini, “Quantum software components and platforms: Overview and quality assessment,” *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–31, 2022.
- [43] J. Preskill, “Quantum Computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, Aug. 2018. [Online]. Available: <https://doi.org/10.22331/q-2018-08-06-79>
- [44] F. Leymann and J. Barzen, “The bitter truth about gate-based quantum algorithms in the NISQ era,” *Quantum Science and Technology*, vol. 5, no. 4, p. 044007, Oct. 2020. [Online]. Available: <https://iopscience.iop.org/article/10.1088/2058-9565/abae7d>
- [45] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, “Superconducting qubits: Current state of play,” *Annual Review of Condensed Matter Physics*, vol. 11, pp. 369–395, 2020.
- [46] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
- [47] A. Myerson, D. Szwer, S. Webster, D. Allcock, M. Curtis, G. Imreh, J. Sherman, D. Stacey, A. Steane, and D. Lucas, “High-fidelity readout of trapped-ion qubits,” *Physical Review Letters*, vol. 100, no. 20, p. 200502, 2008.

- [48] B. Trauzettel, D. V. Bulaev, D. Loss, and G. Burkard, “Spin qubits in graphene quantum dots,” *Nature Physics*, vol. 3, no. 3, pp. 192–196, 2007.
- [49] M. Ruf, N. H. Wan, H. Choi, D. Englund, and R. Hanson, “Quantum networks based on color centers in diamond,” *Journal of Applied Physics*, vol. 130, no. 7, p. 070901, 2021.
- [50] S. Boixo, T. F. Rønnow, S. V. Isakov, Z. Wang, D. Wecker, D. A. Lidar, J. M. Martinis, and M. Troyer, “Evidence for quantum annealing with more than one hundred qubits,” *Nature Physics*, vol. 10, no. 3, p. 218, 2014.
- [51] P. Ball, “Major Quantum Computing Strategy Suffers Serious Setbacks | Quanta Magazine.” [Online]. Available: <https://www.quantamagazine.org/major-quantum-computing-strategy-suffers-serious-setbacks-20210929/>
- [52] “Quantum computing with superconducting qubits — PennyLane.” [Online]. Available: [https://pennylane.ai/qml/demos/tutorial\\_sc\\_qubits.html](https://pennylane.ai/qml/demos/tutorial_sc_qubits.html)
- [53] I. Pogorelov, T. Feldker, C. D. Marciniak, L. Postler, G. Jacob, O. Kriegelsteiner, V. Podlesnic, M. Meth, V. Negnevitsky, M. Stadler, B. Höfer, C. Wächter, K. Lakhmanskiy, R. Blatt, P. Schindler, and T. Monz, “Compact ion-trap quantum computing demonstrator,” *PRX Quantum*, vol. 2, p. 020343, Jun 2021. [Online]. Available: <https://link.aps.org/doi/10.1103/PRXQuantum.2.020343>
- [54] Y.-C. Liu, Y.-C. Dzens, and C.-C. Ting, “Nitrogen vacancy-centered diamond qubit: The fabrication, design, and application in quantum computing,” *IEEE Nanotechnology Magazine*, vol. 16, no. 4, pp. 37–43, 2022.
- [55] A. D. King, S. Suzuki, J. Raymond, A. Zucca, T. Lanting, F. Altomare, A. J. Berkley, S. Ejtemaee, E. Hoskinson, S. Huang *et al.*, “Coherent quantum annealing in a programmable 2000-qubit ising chain,” *arXiv preprint arXiv:2202.05847*, 2022.
- [56] S. H. Sack and M. Serbyn, “Quantum annealing initialization of the quantum approximate optimization algorithm,” *Quantum*, vol. 5, p. 491, Jul. 2021. [Online]. Available: <https://doi.org/10.22331/q-2021-07-01-491>
- [57] J. L. O’Brien, A. Furusawa, and J. Vučković, “Photonic quantum technologies,” *Nature Photonics*, vol. 3, no. 12, pp. 687–695, 2009.
- [58] J. J. Pla, K. Y. Tan, J. P. Dehollain, W. H. Lim, J. J. Morton, D. N. Jamieson, A. S. Dzurak, and A. Morello, “A single-atom electron spin qubit in silicon,” *Nature*, vol. 489, no. 7417, pp. 541–545, 2012.

- [59] D. Schrader, I. Dotsenko, M. Khudaverdyan, Y. Miroshnychenko, A. Rauschenbeutel, and D. Meschede, “Neutral atom quantum register,” *Physical Review Letters*, vol. 93, no. 15, p. 150501, 2004.
- [60] G. Kurizki, P. Bertet, Y. Kubo, K. Mølmer, D. Petrosyan, P. Rabl, and J. Schmiedmayer, “Quantum technologies with hybrid systems,” *Proceedings of the National Academy of Sciences*, vol. 112, no. 13, pp. 3866–3873, 2015.
- [61] C. P. Koch, U. Boscain, T. Calarco, G. Dirr, S. Filipp, S. J. Glaser, R. Kosloff, S. Montangero, T. Schulte-Herbrüggen, D. Sugny *et al.*, “Quantum optimal control in quantum technologies. strategic report on current status, visions and goals for research in europe,” *EPJ Quantum Technology*, vol. 9, no. 1, p. 19, 2022.
- [62] “World Economic Forum report - State of Quantum Computing: Buildin.” [Online]. Available: <https://www.quantumforbusiness.eu/insights/world-economic-forum-report-state-of-quantum-computing-building-a-quantum-economy>
- [63] A. Auffeves, “Quantum technologies need a quantum energy initiative,” *PRX Quantum*, vol. 3, no. 2, p. 020101, 2022.
- [64] A. Botea, A. Kishimoto, and R. Marinescu, “On the complexity of quantum circuit compilation,” in *Eleventh annual symposium on combinatorial search*, 2018.
- [65] D. Gottesman and I. L. Chuang, “Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations,” *Nature*, vol. 402, no. 6760, pp. 390–393, 1999.
- [66] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, “On the qubit routing problem,” p. 32 pages, 2019, arXiv:1902.08091 [quant-ph]. [Online]. Available: <http://arxiv.org/abs/1902.08091>
- [67] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, “Qubit allocation,” in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. Vienna Austria: ACM, Feb. 2018, pp. 113–125. [Online]. Available: <https://dl.acm.org/doi/10.1145/3168822>
- [68] É. Bonnet, T. Miltzow, and P. Rzażewski, “Complexity of token swapping and its variants,” *Algorithmica*, vol. 80, no. 9, pp. 2656–2682, 2018.
- [69] K. Yamanaka, E. D. Demaine, T. Ito, J. Kawahara, M. Kiyomi, Y. Okamoto, T. Saitoh, A. Suzuki, K. Uchizawa, and T. Uno,

- “Swapping labeled tokens on graphs,” *Theoretical Computer Science*, vol. 586, pp. 81–94, 2015, fun with Algorithms. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0304397515001656>
- [70] N. Yoshimura, M. Tawada, S. Tanaka, J. Arai, S. Yagi, H. Uchiyama, and N. Togawa, “Efficient ising model mapping for induced subgraph isomorphism problems using ising machines,” in *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, 2019, pp. 227–232.
- [71] D. Rehfeldt, T. Koch, and Y. Shinano, “Faster exact solution of sparse max-cut and qubo problems,” *Mathematical Programming Computation*, pp. 1–26, 2023.
- [72] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, “The unconstrained binary quadratic programming problem: a survey,” *Journal of combinatorial optimization*, vol. 28, pp. 58–81, 2014.
- [73] “D-Wave Systems | The Practical Quantum Computing Company.” [Online]. Available: <https://www.dwavesys.com/>
- [74] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, “Optimization by simulated annealing,” *science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [75] H. Oshiyama and M. Ohzeki, “Benchmark of quantum-inspired heuristic solvers for quadratic unconstrained binary optimization,” *Scientific reports*, vol. 12, no. 1, pp. 1–10, 2022.
- [76] X.-S. Yang, *Nature-inspired optimization algorithms*. Academic Press, 2020.
- [77] S. Li, X. Zhou, and Y. Feng, “Qubit Mapping Based on Subgraph Isomorphism and Filtered Depth-Limited Search,” *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1777–1788, Nov. 2021, conference Name: IEEE Transactions on Computers.
- [78] N. Quetschlich, L. Burgholzer, and R. Wille, “MQT Bench: Benchmarking software and design automation tools for quantum computing,” 2022, MQT Bench is available at <https://www.cda.cit.tum.de/mqtbench/>.
- [79] D. Große, R. Wille, G. W. Dueck, and R. Drechsler, “Exact synthesis of elementary quantum gate circuits for reversible functions with don’t cares,” in *38th International Symposium on Multiple Valued Logic (ismvl 2008)*. IEEE, 2008, pp. 214–219.

- [80] B. Tan and J. Cong, “Optimal Layout Synthesis for Quantum Computing,” in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Nov. 2020, pp. 1–9, iSSN: 1558-2434.
- [81] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, “ $\text{\$}\langle\text{\$}$  : A Retargetable Compiler for NISQ Devices,” *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, Jan. 2021, arXiv:2003.10611 [quant-ph]. [Online]. Available: <http://arxiv.org/abs/2003.10611>
- [82] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. a. Pereira, “Qubit allocation as a combination of subgraph isomorphism and token swapping,” *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, oct 2019. [Online]. Available: <https://doi.org/10.1145/3360546>