

INSTITUT FÜR INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



Master's Thesis

**Investigating
Evolving Ansatz VQE Algorithms
for Job Shop Scheduling**

Daniel Alexander Leidreiter



Master's Thesis

Investigating Evolving Ansatz VQE Algorithms for Job Shop Scheduling

Daniel Alexander Leidreiter

Aufgabensteller: Prof. Dr. Dieter Kranzlmüller
Betreuer: Korbinian Staudacher
Xiao-Ting Michelle To
Justyna Zawalska (AGH University)
Sven Prüfer (DLR)
Abgabetermin: 12. Juni 2024

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

Neuried, den 12. Juni 2024

A handwritten signature in black ink, appearing to read "O. Leidreiter". The signature is fluid and cursive, with a long horizontal stroke at the end.

.....
(Unterschrift des Kandidaten)

Abstract

The job shop scheduling problem (JSSP) is an important and much-researched combinatorial optimisation problem that is NP-complete for more than two machines [GJS76] [LKB77]. Many algorithms for solving it exactly and approximately exist. Recently, some research has been done on using variational quantum algorithms (VQA) like QAOA or VQE to approximately solve the JSSP [ARF⁺22] [KPS⁺23]. The hope is that such VQAs might be able to find better solutions with less computational effort than classical optimisation heuristics. In VQAs, a quantum circuit whose behaviour is controlled by parameter values is used to create a quantum state. Measuring the expectation value of that quantum state with respect to the problem Hamiltonian yields its quality with respect to the optimisation problem. A classical optimisation algorithm is then used within VQAs in an iterative loop to minimise the expectation value by adjusting the parameter values. This procedure is repeated until a good solution to the optimisation problem is found. The parameterised quantum circuit used within a VQA algorithm is usually referred to as its ansatz. It needs to be designed to provide a good amount of expressibility while avoiding barren plateaus [CAB⁺21] [BCLK⁺22]. This is difficult and has led researchers to investigate adaptive VQA methods, which, in addition to optimising the parameter values, also optimise the structure of the ansatz [BCLK⁺22] [TCC⁺22]. Amongst these are evolving ansatz algorithms like EVQE [RHP⁺19], MoG-VQE [CSU⁺20], and QNEAT [GTMS23], which use evolutionary algorithms to achieve this goal. They have been shown to provide promising results and to be remarkably noise-resistant, while using much smaller ansatz circuits than typical for VQAs. These advantages might make them interesting for solving complex optimisation problems like the JSSP. Yet, it remains unclear how such methods scale to problems of increasing size and complexity [TCC⁺22]. To alleviate this gap in current research, this thesis investigates the scaling of evolving ansatz variational quantum eigensolvers for the JSSP and compares it to the scaling of QAOA and VQE. To this end, the average solution quality and the average number of expectation value evaluations needed for the algorithms to converge are measured over multiple random JSSP problem instances for each problem size. This entails the design of a method for generating random JSSP problem instances and their translation into a Hamiltonian usable by the VQA algorithms. To limit the impact of bad parameter choices on the performance of the VQA algorithms, hyperparameter optimisation is applied before benchmarking.

Contents

1. Introduction	1
2. Foundations	3
2.1. The Job Shop Scheduling Problem	3
2.2. Gate-based Quantum Computing	4
2.2.1. Qubits and Quantum Registers	5
2.2.2. Quantum Gates and Entanglement	7
2.2.3. Measurement	9
2.3. Variational Quantum Algorithms	11
2.3.1. Parameterised Quantum Circuits	12
2.3.2. Hamiltonians	13
2.3.3. Discrete Variable Encodings	15
2.3.4. The Variational Quantum Eigensolver (VQE)	17
2.3.5. The Quantum Approximate Optimisation Algorithm (QAOA)	18
2.4. Classical Optimisation Algorithms	19
2.4.1. Optimisation Problems	19
2.4.2. Optimisation Algorithms	20
3. Related Work	23
4. The Evolutionary VQE Algorithm	31
4.1. Overview	31
4.2. Population	31
4.2.1. Individual Genomes	33
4.2.2. Random Gene Generation	34
4.2.3. Population Initialisation	36
4.2.4. Speciation and Population Diversity	36
4.3. Evaluation and Selection	37
4.3.1. Last Layer Optimisation	37
4.3.2. Fitness Score	37
4.3.3. Selection of fit individuals	38
4.4. Variation	39
4.4.1. Parameter Search	39
4.4.2. Topological Search	39
4.4.3. Layer Removal	39
4.5. Termination Criteria	40
5. Encoding the JSSP as a Hamiltonian	41
5.1. Definitions	42
5.2. Variables and Variable Encoding	42

5.3. Penalties	43
5.3.1. Encoding Penalties	43
5.3.2. Precedence Penalties	44
5.3.3. Overlap Penalties	44
5.3.4. Invalid Penalty Interactions	45
5.4. Optimisation Goal	45
5.4.1. Makespan Minimisation	46
5.4.2. Early Start for all Operations	46
5.5. Resulting Energy Landscape	46
6. EVQE - Issues and Improvements	49
6.1. Individual Initialisation	49
6.2. Selection Pressure	52
7. Methodology	55
7.1. Objectives	55
7.2. Performance Metrics	55
7.3. Random Problem Instance Generation	59
7.3.1. Generating a Single JSSP Instance	59
7.3.2. Generating Datasets of JSSP Instances	59
7.4. Manual Algorithm Configuration Decisions	61
7.5. Automatic Algorithm Configuration	63
7.5.1. SMAC3	63
7.5.2. Hyperparameters	64
7.5.3. Configuring SMAC3	68
7.6. Benchmarking Procedure	69
7.6.1. Real Quantum Hardware	70
8. Results	73
8.1. Hyperparameter Optimisation	73
8.1.1. QAOA Pareto Optimal Configurations	73
8.1.2. VQE Pareto Optimal Configurations	74
8.1.3. EVQE Pareto Optimal Configurations	75
8.2. Noiseless Simulation	77
8.2.1. Termination Behaviour	77
8.2.2. Solution Quality and Success Rate	79
8.2.3. Convergence Speed	81
8.2.4. Ansatz Complexity	83
8.2.5. Comparison of EVQE Population Sizes	84
8.3. Noisy Simulation	85
8.4. Real Quantum Hardware	86
8.5. Discussion	86
9. Conclusion and Future Work	91
A. Additional Figures	93
A.1. Example QAOA Ansatz	93

A.2. Hyperparameter Optimisation Training Instances	94
A.3. 12 Qubit Benchmarking Instances	95
A.4. 15 Qubit Benchmarking Instances	96
A.5. 18 Qubit Benchmarking Instances	97
A.6. 21 Qubit Benchmarking Instances	98
B. Hyperparameter Values	99
B.1. QAOA Hyperparameter Values	99
B.2. VQE Hyperparameter Values	100
B.3. EVQE Hyperparameter Values	101
List of Figures	103
Bibliography	111

1. Introduction

Motivation

The modern world relies heavily on the efficient scheduling of tasks for many applications. These range from applications in manufacturing and supply chain management to applications in railway transportation, healthcare, and many more [XSRH22]. With such a wide array of applications, advances in scheduling optimisation algorithms may bring serious economic benefits. As a result, scheduling optimisation has been an important issue in research for many decades. One commonly researched scheduling problem therein is the Job Shop Scheduling Problem (JSSP). Many classical algorithms for solving the JSSP and its many variants, both exactly and approximately, exist [ZDZ⁺19].

With the increasing accessibility of real quantum computing hardware in recent years, another area of research regarding the JSSP has been focused on quantum algorithms. It is hoped that such quantum algorithms may increase the solution quality or the solving speed when compared to classical algorithms. Due to the noisy and error-prone nature of the current quantum computing hardware, much of this research has been focused on variational quantum algorithms (VQAs) like the Variational Quantum Eigensolver (VQE) or the Quantum Approximate Optimisation Algorithm (QAOA). They use a classical computer to iteratively optimise the rotation angle parameters of a quantum circuit so that the expectation value of the quantum state the quantum circuit produces is minimised with respect to the problem Hamiltonian (a more detailed explanation of VQAs can be found in Section 2.3). The way in which this quantum circuit is designed depends on the specific algorithm and hugely impacts its performance.

This motivates VQA algorithms, which use tools from machine learning to find good quantum circuit designs [TCC⁺22]. Among these are VQE algorithms, which use evolutionary algorithms to evolve a quantum circuit structure during the optimisation process [RHP⁺19] [CSU⁺20] [GTMS23]. In this thesis, these algorithms will be referred to as evolving ansatz VQE (EA-VQE) as an umbrella term¹. While EA-VQE algorithms have shown promising results so far, it is not yet clear how the additional effort to learn circuit designs affects the algorithm's scaling to larger problem sizes, especially when compared to VQAs with fixed quantum circuits.

Research Questions

To alleviate this research gap, this thesis investigates the following Research Question (RQ) and Sub Questions (SQ):

RQ: How do evolving ansatz VQE algorithms scale in terms of computational effort and solution quality for job shop scheduling problems?

SQ: How does this scaling compare to QAOA and VQE?

¹No such umbrella term seems to exist yet.

Contributions

To answer the main research question, we investigate the EVQE algorithm [RHP⁺19] as a representative of EA-VQE algorithms in general. We evaluate EVQE’s scaling by benchmarking EVQE multiple times over various JSSP instances of increasing sizes. This enables us to investigate how EVQE’s average performance scales over these increasing problem sizes. In particular, we observe two main performance metrics. The first metric is the computational effort needed by EVQE, which can be quantified by the number of times the expectation value of the quantum state produced by the ansatz circuit is evaluated with respect to the problem Hamiltonian. The second metric is the quality of the result found by EVQE, which can be quantified by the likelihood of measuring valid or optimal solutions from EVQE’s best quantum state. As a frame of reference that allows us to compare the scaling results for EVQE to more established VQA algorithms, we run the same benchmarks for both the QAOA and VQE algorithms.

As a prerequisite for these benchmarks, we implement EVQE in the open-source Python library QUEASARS²³ and implement improvements for EVQE where necessary. We also implement a mapping for JSSP instances to the Ising Hamiltonian problem formulation, which is needed by all the algorithms we investigate. Finally, we implement a random problem instance generation method for JSSP instances, which enables us to generate a dataset of problem instances for our benchmarks. Preceding the benchmarks, we then tune the hyperparameters for all VQA algorithms that we investigate.

Structure

The theoretical foundations with respect to the JSSP, quantum computing hardware, VQAs, and classical optimisation algorithms are explained in Chapter 2. Afterwards, the related work in current research is examined in Chapter 3, and the EVQE algorithm is explained in detail in Chapter 4. The encoding of the JSSP to an Ising Hamiltonian is detailed in Chapter 5, and improvements to EVQE are proposed in Chapter 6. Given the groundwork of the previous chapters, the methodology for our benchmarks is outlined in Chapter 7, and the acquired results are presented and discussed in Chapter 8. Finally, we give our conclusion and outline opportunities for future work in Chapter 9.

Acknowledgements

The QUEASARS Python library was developed as a part of the Quantum Mission Planning Challenges (QMPC) project at the Mission Planning Group of the German Space Operation Center (GSOC) within the German Aerospace Center (DLR). The QMPC project is funded by the Quantum Computing Initiative, which in turn is funded by the federal ministry for economic affairs and climate action (BMWK).

We gratefully acknowledge the computational and data resources provided through the joint high-performance data analytics (HPDA) project “terabyte” of the German Aerospace Center (DLR) and the Leibniz Supercomputing Center (LRZ).

²QUEASARS is licenced under the Apache Licence 2.0.

³The QUEASARS GitHub repository can be found at: <https://github.com/DLR-RB/QUEASARS>

2. Foundations

In this chapter, we explain the theoretical foundations that are needed for understanding this thesis. In Section 2.1, we explain the exact formulation of the JSSP. Following that, we explain the basics of gate-based quantum computing in Section 2.2. We then go on to describe how VQAs leverage gate-based quantum computers to solve optimisation problems in Section 2.3. Finally, since VQAs employ classical optimisation algorithms, we give an overview of classical optimisation algorithms in Section 2.4.

2.1. The Job Shop Scheduling Problem

A job shop is a type of manufacturing process in which small batches of products are produced to the customer's requirements. The products often have to pass through multiple manufacturing steps on machines in some specific order to finish their production [Bur17]. Scheduling the production steps for each job has the potential to significantly reduce costs and increase the throughput of a job shop. This task is also referred to as job shop scheduling [RJ16].

With some abstractions, this task yields the JSSP. In it, a set of jobs $\{J_1, \dots, J_n\}$ needs to be scheduled on a set of machines $\{M_1, \dots, M_m\}$. Each such job j consists of an ordered sequence of operations $[O_{j,1}, \dots, O_{j,k_j}]$. Each operation in turn must be executed on a specific machine without interruption for a specific processing duration $p_{j,k}$. The goal of the JSSP is to find a schedule in which no machine processes more than one operation at any time, the order of the operations in each job is respected, and the time to finish all jobs (makespan) is minimised [XSRH22].

As an example, take the JSSP instance with three jobs and three machines in Figure 2.1.

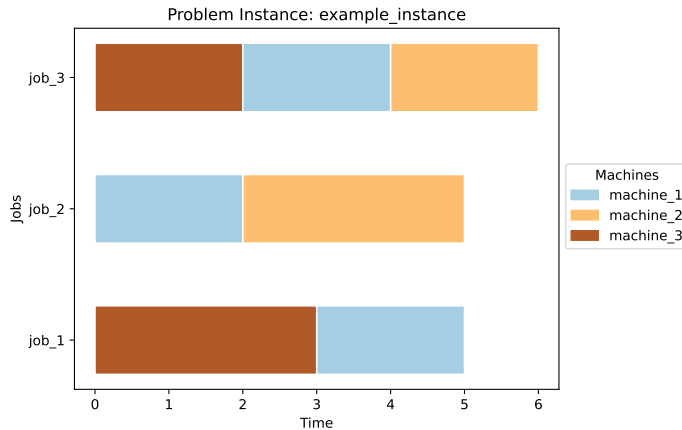


Figure 2.1.: This figure shows a visualisation of the operation order in each job for an example JSSP instance, with 3 jobs on 3 machines.

2. Foundations

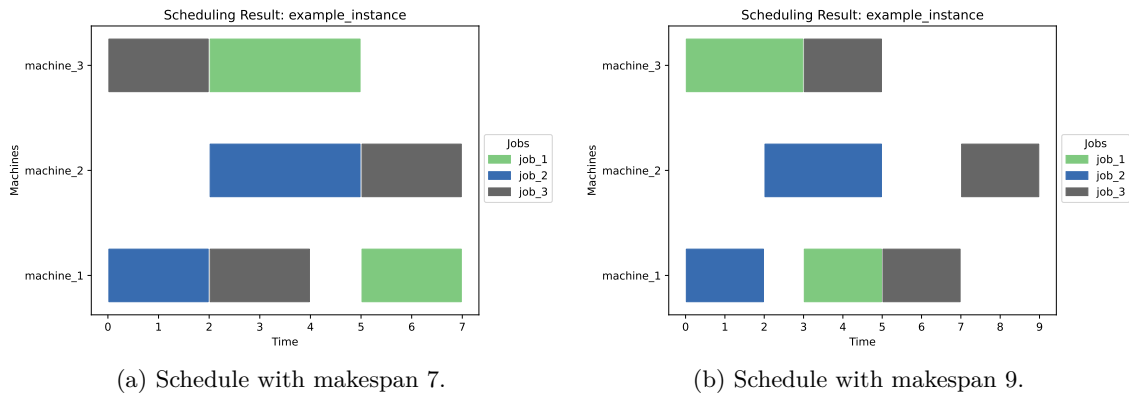


Figure 2.2.: This figure shows a comparison of valid scheduling results with different makespans for the JSSP instance shown in Figure 2.1. The schedule shown in (a) is optimal. The schedule shown in (b) is suboptimal.

This figure visualises the structure of the unsolved scheduling problem by showing the operation order for each job horizontally from left to right. The colours of the operations indicate on which machine they have to be processed, while the length of each operation’s bar shows its processing duration.

A valid solution to the example problem instance can be found with a minimum makespan of 7 (see Figure 2.2a), but other valid schedules with a worse makespan exist (see Figure 2.2b). Notice that these figures, in contrast to Figure 2.1, display how the operations are scheduled to the machines in a valid solution. In particular, these figures show, from left to right, the schedule with which operations are processed on each machine, where the colour of the operations indicates the job they belong to.

Going forward, the two types of plots shown here can be distinguished based on their titles. Any plot titled “Problem Instance” with the jobs placed on the y-axis will show a visualisation of an unsolved problem instance, whereas any plot titled “Scheduling Result” with the machines placed on the y-axis will show a visualisation of a valid solution to a JSSP instance. This distinction is further emphasised by the two plot types using different colour schemes.

Since the JSSP’s inception, it has been applied to many real-world problems (e.g., semiconductor manufacturing, automobile manufacturing, textile production), and many extensions of the JSSP have been investigated to widen its applicability [XSRH22]. For simplicity’s sake, this thesis will be limited to the basic JSSP variant outlined before, as it in itself is already NP-complete when applied to more than two machines [GJS76].

2.2. Gate-based Quantum Computing

Quantum computing takes advantage of quantum effects, like entanglement, and the ability of quantum systems to be in a superposition of states to gain speed-ups for certain problems. Multiple approaches to quantum computing exist, but this thesis will limit itself to gate-based quantum computing.

On gate-based quantum computers, quantum algorithms are expressed as quantum cir-

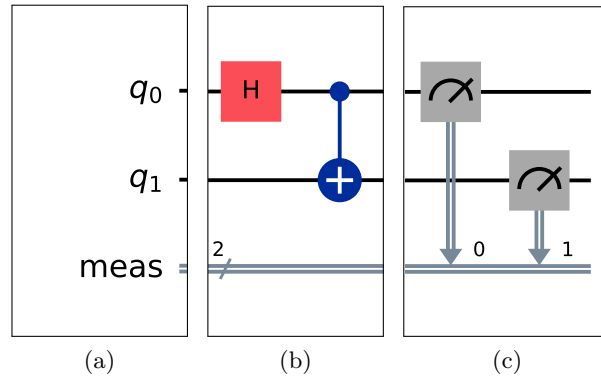


Figure 2.3.: This figure shows a quantum circuit consisting of a quantum register of two qubits (a), a classical register to store results (a), quantum gates (b), and measurement instructions (c).

circuits. They consist of a register of quantum bits in some initial state (Figure 2.3a), a classical register for storing measurements (Figure 2.3a), an ordered collection of quantum gates (Figure 2.3b), and some measurement instructions (Figure 2.3c), which store their results in the classical register. The following subsection will explain each of these components of a quantum circuit to enable a closer discussion of algorithms for gate-based quantum computers. These explanations are based on the literature by Homeister [Hom22] and Zygelman [Zyg18].

2.2.1. Qubits and Quantum Registers

Qubits

Each quantum register consists of some number of quantum bits, which are also referred to as *qubits*. The state $|x\rangle$ of a single qubit can be described as a linear combination of the basis states $|0\rangle$ and $|1\rangle$:

$$|x\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle \quad (\alpha, \beta \in \mathbb{C}) \quad (2.1)$$

This linear combination can also be written as a two-dimensional complex vector:

$$|x\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle = \alpha \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad (\alpha, \beta \in \mathbb{C}) \quad (2.2)$$

Only such linear combinations in which the so-called *amplitudes*, α and β , fulfil the following condition represent valid quantum states:

$$|\alpha|^2 + |\beta|^2 = 1 \quad (2.3)$$

This means that, in contrast to bits on a classical computer, which can only exhibit either the 0 or 1 state, qubits can exhibit states that are in between $|0\rangle$ and $|1\rangle$. In such states, the qubit is said to be in *superposition*.

When the state of the qubit is measured, the superposition cannot be directly observed. Instead, the superposition collapses. If the measurement was done in the Z-basis, either the state $|0\rangle$ or the state $|1\rangle$ will be measured with the probabilities $|\alpha|^2$ and $|\beta|^2$, respectively.

2. Foundations

Disregarding an overall phase factor, a qubit's state $|x\rangle$ can also be represented in terms of two angles, θ and ϕ :

$$|x\rangle = \cos\left(\frac{\theta}{2}\right) \cdot |0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) \cdot |1\rangle \quad (0 \leq \theta \leq \pi, 0 \leq \phi \leq 2\pi) \quad (2.4)$$

This enables the state of a single qubit to be visually represented as a point on the surface of a sphere with a radius of one (the so-called Bloch sphere). In this representation, the coordinates of a qubit's state are expressed with θ as the polar angle and ϕ as the azimuth angle in a spherical coordinate system. The North Pole of the Bloch sphere represents the state $|0\rangle$, while its South Pole represents the state $|1\rangle$. A visualisation of the state $|0\rangle$ on the Bloch sphere can be seen in Figure 2.4.

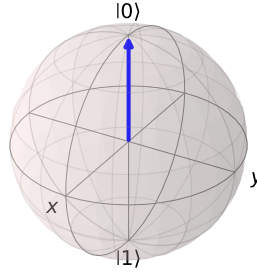


Figure 2.4.: This figure shows a Bloch sphere representation of the state $|0\rangle$.

Quantum Registers

Quantum registers combine multiple qubits into one quantum system. The state $|R\rangle$ of a quantum register, which consists of n qubits, can be described as the linear combination of $N = 2^n$ basis states. They arise from all possible combinations of the two basis states $|0\rangle, |1\rangle$ for all n qubits. These basis states can be denoted as a bitstring or as a decimal if the bitstring is interpreted as a binary number (see Equation 2.5). Note that in the bitstring representation, the order of the bits in the bitstring is inverse to the order of the qubits, and therefore, the right-most bit refers to the state of the first qubit.

$$\begin{aligned} |R\rangle &= \alpha_{00\dots 0} \cdot |00\dots 0\rangle + \dots + \alpha_{11\dots 1} \cdot |11\dots 1\rangle \\ &= \alpha_0 \cdot |0\rangle + \dots + \alpha_{N-1} \cdot |N-1\rangle \end{aligned} \quad (2.5)$$

This linear combination can also be written as an n -dimensional complex vector:

$$\begin{aligned} |R\rangle &= \alpha_0 \cdot |0\rangle + \dots + \alpha_{N-1} \cdot |N-1\rangle \\ &= \alpha_0 \cdot \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \dots + \alpha_{N-1} \cdot \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{N-1} \end{pmatrix} \end{aligned} \quad (2.6)$$

When measuring the state of the quantum register, again, no superposition is observed. Instead, when measuring in the Z-basis, the state $|k\rangle$ ($0 \leq k \leq N - 1$, $k \in \mathbb{N}$) is observed with probability $|\alpha_k|^2$. It also holds for any quantum register's state that:

$$|\alpha_0|^2 + \dots + |\alpha_{N-1}|^2 = 1 \tag{2.7}$$

2.2.2. Quantum Gates and Entanglement

Quantum gates are operations used to change the state of a quantum system. Any quantum gate that operates on n qubits can be described as a matrix M of size $N \times N$, where $N = 2^n$. The resulting state $|x'\rangle$ from applying a quantum gate described by M to the state $|x\rangle$ can be calculated as a matrix vector product:

$$|x'\rangle = M \cdot |x\rangle \tag{2.8}$$

For any valid quantum gate, its matrix M must be unitary, which means that its adjoint M^\dagger must be equal to its inverse M^{-1} . This ensures that quantum gates are always reversible and that no quantum information is lost. Additionally, it also ensures that the sum of all measurement probabilities does not change and thus always remains one.

Single-Qubit Gates

Quantum gates acting on a single qubit can be understood as rotations of the quantum state on the Bloch sphere. Take, for instance, the Pauli-X gate, which can be understood as applying a rotation of π around the x-axis of the Bloch sphere:

$$q \text{ --- } \boxed{X} \text{ ---} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.9}$$

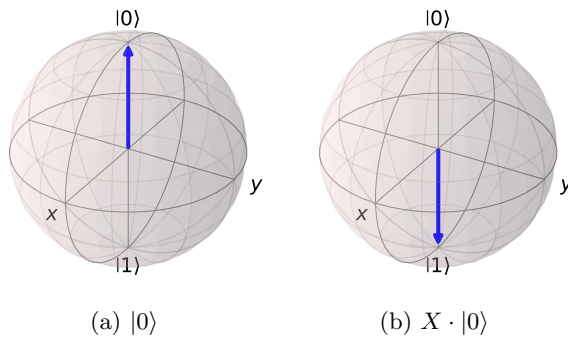


Figure 2.5.: This figure shows a qubit's state before (a) and after (b) applying a Pauli-X gate.

Similarly, the Pauli-Y and Pauli-Z gates apply a rotation of π around the y and z-axis of the Bloch sphere respectively.

Another important single-qubit quantum gate is the Hadamard gate. It can be understood as applying a rotation of π around the z-axis and then a rotation of $\frac{\pi}{2}$ around the y-axis of

2. Foundations

the Bloch sphere. From the basis states $|0\rangle$ or $|1\rangle$, the Hadamard gate changes the qubit's state to a superposition in which both $|0\rangle$ and $|1\rangle$ are equally likely to be observed:

$$q \text{ --- } \boxed{\text{H}} \text{ ---} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (2.10)$$

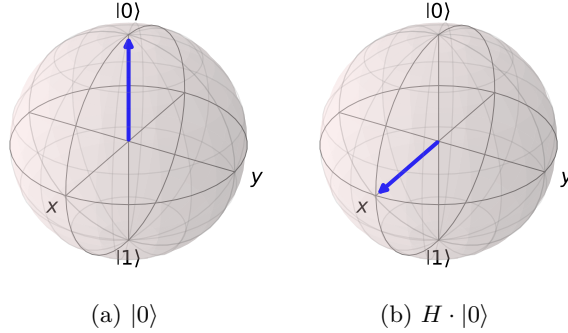


Figure 2.6.: Qubit state before (a) and after (b) applying a Hadamard gate.

One of the most general single-qubit gates is the U3 gate. It can be used to apply any of the previously explained single-qubit gates. It takes three angles, $\theta, \phi, \lambda \in [0, 2\pi]$, and allows an arbitrary rotation around an arbitrary axis on the Bloch sphere. θ specifies the magnitude of the rotation, whereas ϕ and λ specify the rotation axis:

$$q \text{ --- } \boxed{\text{U}}_{\theta, \phi, \lambda} \text{ ---} \quad U3(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda} \sin(\frac{\theta}{2}) \\ e^{i\phi} \sin(\frac{\theta}{2}) & e^{i(\phi+\lambda)} \cos(\frac{\theta}{2}) \end{pmatrix} \quad (2.11)$$

Multi-Qubit Gates

To apply single-qubit quantum gates to a quantum register consisting of n qubits, n single-qubit gates represented by the matrices M_1, \dots, M_n can be combined to a multi-qubit gate represented by the matrix M using the tensor product. This has the effect of individually applying the single-qubit gates to the qubits of the respective index:

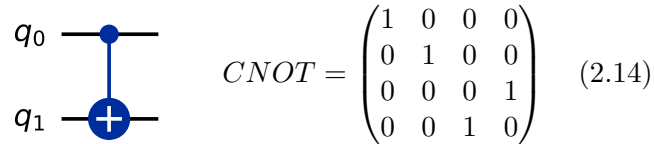
$$M = M_n \otimes \dots \otimes M_1 \quad (2.12)$$

If no gate should be applied for some qubits, the single-qubit identity gate can be used as part of the tensor product:

$$q \text{ --- } \boxed{\text{I}} \text{ ---} \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (2.13)$$

Multi-qubit gates, which are constructed in this way, affect the individual qubits without interaction. Other multi-qubit gates cannot be constructed from single-qubit gates. They enable state interactions between the qubits. Among these gates are controlled quantum

gates. They typically act on two qubits, with one being the control and the other the controlled qubit. They apply a state change to the controlled qubit only if the control qubit is in state $|1\rangle$. If the control qubit is in superposition, only the amplitudes of the basis states in which the control qubit is $|1\rangle$ are affected. In all cases, no state change is applied to the control qubit. The most well-known controlled gate is the controlled Pauli-X gate (also known as the CNOT gate), but any single qubit gate can be controlled in such a way:



$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.14)$$

To demonstrate its effect, see the $CNOT$ gate being applied to a two-qubit register, in which the first qubit is in superposition and the second qubit is in state $|0\rangle$:

$$CNOT \cdot \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle \right) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad (2.15)$$

Before applying the $CNOT$, the state of the second qubit is independent of the state of the first qubit. After applying the $CNOT$, the state of the second qubit depends on the state of the first qubit. Such quantum states in which the states of the individual qubits are not independent are also called entangled. This phenomenon is also referred to as *entanglement*.

Quantum Circuit Depth

If a quantum gate follows another quantum gate on the same qubit, the final state of that qubit depends on the original state of the qubit and the two quantum gates applied to it. Since each quantum gate is error-prone to a certain degree, errors over long sequences of quantum gates can add up. Therefore, an important metric for quantum circuits is the quantum circuit depth. The quantum circuit depth is the longest sequential chain of quantum gates on which the final state of any qubit in the quantum circuit depends.

Take, for instance, the quantum circuit in Figure 2.3. The final state of the zeroth qubit only depends on the Hadamard gate, as the control part of the CNOT gate does not change the zeroth qubit's state. For the first qubit, its final state depends on the Hadamard gate and CNOT gate in sequence, as the effect of the CNOT gate is controlled by the state of the zeroth qubit after the Hadamard gate has been applied. Therefore, the quantum circuit in Figure 2.3 has a quantum circuit depth of 2, as the longest chain of dependence is two quantum gates long.

2.2.3. Measurement

When measuring in the Z-basis, the state $|k\rangle$ of a qubit or quantum register can be observed with the probability $|\alpha_k|^2$. The measurement has the effect of collapsing the superposition to the state $|k\rangle$. Therefore, after the state $|k\rangle$ was measured, the amplitude α_k is one, and all other amplitudes are 0. This means that a subsequent measurement will certainly return $|k\rangle$ again.

2. Foundations

Since the result of the measurement is probabilistic, multiple measurements are needed to get statistically relevant results. Due to the fact that the measurement collapses the superposition, the quantum state needs to be prepared anew for each measurement. How many measurements (also referred to as shots) are needed depends on the required certainty of the results and the noise of the quantum hardware. The results of such repeated measurements can be presented as a probability distribution over the observed states. Take, for instance, the quantum circuit in Figure 2.3, which creates an equal superposition of the states $|00\rangle$ and $|11\rangle$. A simulated measurement of that circuit with 500 shots yields a probability distribution in which these states are nearly equally likely to be observed (see Figure 2.7).

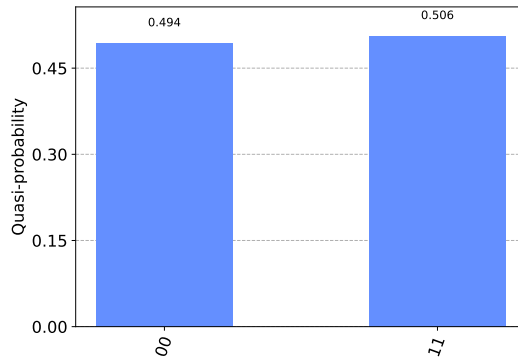


Figure 2.7.: This figure shows the simulated measurement results (noiseless, 500 shots) of the circuit in Figure 2.3.

Going forward, in this thesis, quantum circuit measurement will always refer to the process of retrieving a probability distribution of states using multiple shots.

Observables

So far, the measurement process has only been described for measurements in the Z-basis. In general, the single-shot measurement process can be described using an observable. An observable is an operator described by a hermitian matrix. It specifies some measurable aspect of a quantum system. Measuring the quantum system with respect to an observable returns some eigenvalue λ_i of the observable and collapses the quantum state to the corresponding eigenstate $|e_i\rangle$. The measurement of a single qubit with respect to the basis states $\{|0\rangle, |1\rangle\}$ can, for instance, be described by the Pauli-Z observable (see Equation 2.16), whose eigenstates correspond to these basis states and whose eigenvalues are 1 for the eigenstate $|0\rangle$ and -1 for the eigenstate $|1\rangle$.

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (2.16)$$

Other common single qubit observables are the Pauli-X observable σ_x (see Equation 2.17), which allows measurement with respect to the basis states $\{|+\rangle, |-\rangle\}$ (see Equation 2.18), and the Pauli-Y observable σ_y (see Equation 2.19), which allows measurement with respect to the basis states $\{|i\rangle, |-i\rangle\}$ (see Equation 2.20).

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (2.17)$$

$$|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle), \quad |-\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \quad (2.18)$$

$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad (2.19)$$

$$|i\rangle = \frac{1}{\sqrt{2}} (|0\rangle + i|1\rangle), \quad |-i\rangle = \frac{1}{\sqrt{2}} (|0\rangle - i|1\rangle) \quad (2.20)$$

When measuring with respect to an observable, the resulting eigenvalue is probabilistic. If multiple shots are used, an expectation value over the observed eigenvalues can be formed. For a known state $|\psi\rangle$ and an observable O , this eigenvalue can also be calculated as:

$$\langle\psi|O|\psi\rangle \quad (2.21)$$

2.3. Variational Quantum Algorithms

Classical optimisation algorithms typically evaluate a finite number of solutions from the search space during an optimisation step. Quantum optimisation algorithms, on the other hand, can use quantum superpositions to their advantage. An example of this is Grover's algorithm [Gro96], which first creates an equal superposition over all solutions and then uses repeated sequences of quantum operations to amplify the amplitudes (and thus the measurement probability) of good solutions [Hom22]. The issue is that this process requires a lot of quantum gates [CS20] [JNRV20], each of which is noisy and error-prone to a certain degree on current quantum computers. This issue is common for complex quantum algorithms that are purely executed on quantum hardware. Due to their generally large quantum circuits, the quantum noise adds up quickly, which inhibits their usefulness on current quantum hardware [WK20].

While such complex quantum algorithms are out of reach for current quantum hardware, researchers still endeavour to make use of the current hardware's capabilities [BCLK⁺22]. This leads to the idea of variational quantum algorithms (VQAs). These quantum algorithms delegate a part of the computation to a classical computer, which allows their quantum circuits to be much shorter. This is done in an iterative loop (see Figure 2.8), in which the quantum computer is used to create a quantum state $|\psi(\theta)\rangle$ based on some parameter values θ . That state is then measured, and a classical computer is used to evaluate and improve the quality of that quantum state by changing the parameter values, based on an objective function O [BCLK⁺22]. For classical optimisation problems, this can intuitively be thought of as the quantum computer creating a superposition of possible solutions to the optimisation problem. This superposition is then iteratively improved by the classical optimiser so that solutions of increasing quality are more likely to be measured.

The following subsections explain how parameterised quantum circuits are used to create quantum states $|\psi(\theta)\rangle$ based on parameter values θ and how Hamiltonians can be used to measure and evaluate the quality of a quantum state with regard to an optimisation problem. After that, the cornerstone variational algorithms, Variational Quantum Eigensolver (VQE) and Quantum Approximate Optimisation Algorithm (QAOA), are explained.

2. Foundations

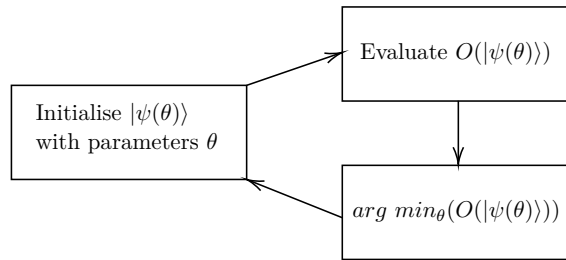


Figure 2.8.: This figure shows the general workflow of VQA algorithms, which is an iterative loop. In it, a quantum state $|\psi(\theta)\rangle$ is created based on the parameter values θ . This state is then evaluated using objective function O . Based on the resulting objective value, the classical optimiser improves the parameter values θ .

2.3.1. Parameterised Quantum Circuits

In the previous sections on quantum computing, it was outlined that quantum gates can be understood as rotations of the qubits' states. Some rotation gates, like the $U3$ gate, provide parameters that allow the user to specify the extent of the rotations that will be applied. A parameterised quantum circuit is a quantum circuit that contains many such parameterised gates, whose rotation angles are not yet predetermined.

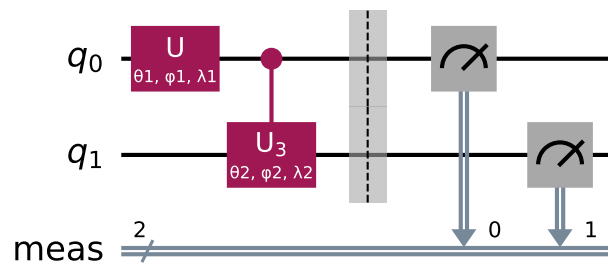


Figure 2.9.: This figure shows a parameterised version of the circuit in Figure 2.3 that uses $U3$ gates.

To execute such a parameterised circuit, all rotation angles need to be populated, with the resulting quantum state varying greatly based on the chosen parameter values. In the context of VQAs, such a parameterised quantum circuit is called the VQA's ansatz [CAB⁺21]. It spans the search space of possible quantum states for the classical computer to optimise over. Take, as an example, the parameterised quantum circuit in Figure 2.9. It is a parameterised version of the quantum circuit in Figure 2.3, where the H gate has been replaced by a $U3$ gate and the $CNOT$ gate has been replaced by a controlled $U3$ gate. For some specific parameter values, these gates act exactly like the H and $CNOT$ gates, retrieving the same result (see Figure 2.10a) as the non parameterised circuit (see Figure 2.7). A different result emerges when changing only two parameter values (see Figure 2.10b).

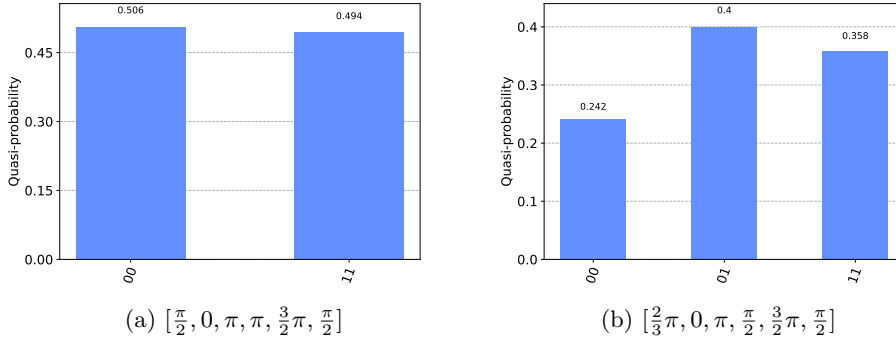


Figure 2.10.: This figure shows the simulated measurement results (noiseless, 500 shots) of the circuit in Figure 2.9 for different parameter values. The parameter values that were applied are shown below the individual figures in the order $[\theta_1, \phi_1, \lambda_1, \theta_2, \phi_2, \lambda_2]$.

2.3.2. Hamiltonians

A Hamiltonian H is an observable operator that describes the energy of a physical system. Measuring its expectation value $\langle \psi | H | \psi \rangle$ for a quantum state $|\psi\rangle$ yields the energy of that quantum state [BCLK⁺22]. Hamiltonians can also be used to encode optimisation problems so that states with a minimum energy correspond to the solution of the optimisation problem [Luc14]. One way to compose Hamiltonians is as a linear combination of Pauli strings \hat{P} . Pauli strings are observables that consist of a tensor product of single-qubit Pauli observables $\{\sigma_x, \sigma_y, \sigma_z\}$ and single qubit identity operators [BCLK⁺22].

$$H = \sum_{k=1}^M c_k \cdot \hat{P}_k, \quad (c_k \in \mathbb{C}) \quad (2.22)$$

$$\hat{P} = \otimes_{j=1}^n \sigma_j, \quad (\sigma \in \{\sigma_x, \sigma_y, \sigma_z, I\}) \quad (2.23)$$

Given a Hamiltonian composed of a linear combination of Pauli strings, its expectation value can be retrieved as a linear combination of the expectation value of its composing Pauli strings [BCLK⁺22].

$$\langle \psi | H | \psi \rangle = \sum_{k=1}^M c_k \cdot \langle \psi | \hat{P}_k | \psi \rangle \quad (2.24)$$

A Pauli string that is commonly used is the Z_i Pauli string. It consists of only one Pauli-Z observable and identity observables (see Equation 2.25). For an n -qubit quantum register, its eigenvalues are -1 for eigenstates in which the i -th qubit is one. Conversely, for eigenstates in which the i -th qubit is zero, its eigenvalues are $+1$.

$$Z_i = I_n \otimes \cdots \otimes \sigma_{z_i} \otimes \cdots \otimes I_1 \quad (2.25)$$

Since Hamiltonians describe the energy of physical systems, it seems natural to use models of physical systems to encode optimisation problems as Hamiltonians. One such model is the

2. Foundations

Ising model, which was introduced in the context of ferromagnetism [Hom22]. In general, it can model physical systems consisting of n elements s_k , which can be in two discrete states $s_i \in \{-1, 1\}$, with the energy of that system $E(s)$ being based on the pairwise interaction of the elements and some external factors. The energy of such Ising systems can be formalised with Equation 2.26, where $J_{i,j}$ describes the interaction between the two elements s_i and s_j and h_i describes an external factor acting on s_i [Hom22].

$$E(s) = \sum_{i < j \leq n} J_{i,j} \cdot s_i \cdot s_j + \sum_{i=1}^n h_i \cdot s_i, \quad (s_k \in \{-1, 1\}) \quad (2.26)$$

To encode the Ising model as a Hamiltonian, one can use Pauli strings Z_i to replace the variables s_i from Equation 2.26 ($s_i = -Z_i$). This results in Equation 2.27, which yields the Hamiltonian describing the energy of the Ising system. Its eigenstates and eigenvalues match the states and energy values of the system, as given by Equation 2.26.

$$H_{ising} = \sum_{i < j \leq n} J_{i,j} \cdot Z_i \cdot Z_j + \sum_{i=1}^n h_i \cdot -Z_i \quad (2.27)$$

Crucially, any quadratic unconstrained binary optimisation (QUBO) problem (in which functions in the form shown in Equation 2.28 are optimised) is easily transformed into the form of the Ising model, by defining $x_i = \frac{s_i+1}{2}$ [Hom22]. Therefore, any QUBO problem can be expressed as an Ising Hamiltonian.

$$f(x) = \sum_{i,j} J_{i,j} \cdot x_i \cdot x_j + \sum_i h_i \cdot x_i + c, \quad (x_k \in \{0, 1\}) \quad (2.28)$$

Going forward, in this thesis, Hamiltonians for optimisation problems will be described in the QUBO form, with the mapping from the QUBO to the Ising Hamiltonian being automatically implied as described in this subsection.

Example: From QUBO to Hamiltonian

This section provides an example to illustrate the QUBO to Ising Hamiltonian conversion process. The QUBO in this example is a minimisation problem with two binary variables (see Equation 2.29). Its minimum is located at $x_0 = 0, x_1 = 1$.

$$\min_{x_0, x_1 \in \{0, 1\}} f(x) = x_0 \cdot x_1 - 2 \cdot x_1 \quad (2.29)$$

The next step is to substitute the variables $x_i \in \{0, 1\}$ with Ising variables $s_i \in \{-1, 1\}$ by defining $x_i = \frac{s_i+1}{2}$. The result of this step can be seen in Equation 2.30.

$$E(s) = \frac{s_1 + 1}{2} \cdot \frac{s_0 + 1}{2} - 2 \cdot \frac{s_1 + 1}{2} \quad (2.30)$$

With the QUBO minimisation task formulated as an Ising model, the next step is to convert the Ising model to a Hamiltonian by substituting the Ising variables with Pauli Z_i strings ($s_i = -Z_i$). Note that the 1s added to the Ising variables also need to be replaced by Pauli identity strings I , which consist only of identities. The result of that conversion is a Hamiltonian written as a combination of its constituting Pauli strings (see Equation 2.31).

This formulation is sufficient for variational algorithms, as the expectation value of the Hamiltonian can be calculated based on the expectation of its constituting Pauli strings.

$$H = \frac{-Z_0 + I}{2} \cdot \frac{-Z_1 + I}{2} - 2 \cdot \frac{-Z_1 + I}{2} \quad (2.31)$$

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \quad (2.32)$$

Given Equation 2.31, one can also calculate the matrix representation of that Hamiltonian. This calculation is done here only for explanatory purposes and is normally not done in VQAs. The resulting matrix for the example Hamiltonian (see Equation 2.32) is a diagonal matrix, which allows the direct read-out of its eigenstates and eigenvalues from the values on its diagonal (highlighted in blue). To be exact, the n -th value on the diagonal, starting from zero for the left-most value, is then the eigenvalue for the eigenstate $|n\rangle$ in decimal notation. For example, -2 is the eigenvalue for the state $|2\rangle$, which, when written in binary, is the state $|10\rangle$. From the eigenstates, the state of the qubits, and thus the state of the binary variables, can be determined by then reading the bits of the binary state in inverse order. Therefore, in the state $|10\rangle$, the variable x_0 is 0 and the variable x_1 is 1. The read-out of all values on the Hamiltonian's diagonal then yields Table 2.11. One can then plug the variable assignments from Table 2.11 into the QUBO function in Equation 2.29, which shows that the Hamiltonian's eigenvalues match the results of the QUBO function exactly.

Eigenstate	Corresponding Variable Assignments	Eigenvalue
$ 00\rangle$	$x_0 = 0, x_1 = 0$	0
$ 01\rangle$	$x_0 = 1, x_1 = 0$	0
$ 10\rangle$	$x_0 = 0, x_1 = 1$	-2
$ 11\rangle$	$x_0 = 1, x_1 = 1$	-1

Figure 2.11.: This table shows the eigenstates and eigenvalues of the example Hamiltonian (Equation 2.32) with the corresponding QUBO variable assignments.

2.3.3. Discrete Variable Encodings

A common way to express an optimisation problem as an Ising Hamiltonian is to formalise it as a QUBO first. This is not easily doable for all optimisation problems. A common issue is that many optimisation problems need discrete variables, which may represent more than two possible values. An example of this are integer variables, like the start time of the operations within the JSSP. These discrete variables cannot be represented using only one binary QUBO variable. Instead, discrete variables need to be expressed using a collection of binary QUBO variables each. Methods that achieve this are referred to as *encodings*.

Encoding a discrete variable onto a group of binary variables works by assigning each value of the discrete variable a bit-value combination for the group of QUBO variables. If these QUBO variables then take values matching an assigned value combination, this counts as the discrete variable holding the value corresponding to that combination.

2. Foundations

Let us now refer to an encoded discrete variable consisting of n QUBO variables x_i as y . To be able to work with the variable y in the QUBO, the value it takes needs to be accessible with at most a quadratic term. In the following, this term will be referred to as $c(y, v)$, which is 1 if y contains the value v and 0 otherwise (see Equation 2.33).

$$c(y, v) = \begin{cases} 1 & \text{if } y = v \\ 0 & \text{if } y \neq v \end{cases} \quad (2.33)$$

If using n QUBO variables to encode the k choices offered by a discrete variable, it is the case that if $k < 2^n$, not all state combinations of the QUBO variables represent valid values of the encoded discrete variable. Such states are invalid encoding states. In these states, the function $c(y, v)$ is not well-defined. In the QUBO formulation, such states need to be penalised with a penalty term $p(y)$ to avoid the minimisation procedure falling into such states. It should be zero if the variable is in a valid state and greater than zero otherwise:

$$p(y) = \begin{cases} 0 & \text{if } y \text{ is valid} \\ > 0 & \text{if } y \text{ is invalid} \end{cases} \quad (2.34)$$

In the following, the one-hot encoding and the domain wall encoding are explained, based on the work of Plewa et al. [PSR21].

One-Hot Encoding

In the one-hot encoding, a discrete variable y with k possible values is encoded in k binary variables, so that if y takes its i -th value, only x_i is 1 and all other binary variables that are part of y are zero. An example of the one-hot encoding for a discrete variable $y \in \{0, 1, 2\}$ can be seen in Table 2.12.

$x_0x_1x_2$	y
100	0
010	1
001	2

Figure 2.12.: This table demonstrates the one-hot encoding for a variable $y \in \{0, 1, 2\}$.

In the one-hot encoding, checking whether the variable y takes the i -th value v_i is achieved simply by checking the variable x_i :

$$c_{oh}(y, v_i) = x_i \quad (2.35)$$

For this encoding, each state in which more than one binary variable is one is invalid. This can be penalised with the following penalty term:

$$p_{oh}(y) = \left(1 - \sum_i x_i\right)^2 \quad (2.36)$$

Domain Wall Encoding

In the domain wall encoding, a discrete variable y , which can take k different values, is encoded in $k - 1$ binary variables. In this encoding, the i -th value of the variable is encoded

by the first i variables x_0, \dots, x_{i-1} being 1 and the other variables being 0. This encoding is named after the so-called domain wall, which refers to the space between bits of differing values, since in this encoding, the place of the domain wall encodes the variable value. An example of the domain wall encoding for a discrete variable $y \in \{0, 1, 2, 3\}$ can be seen in Table 2.13. In the example, for the states $y = 0$ and $y = 3$, there is still a domain wall because, in this encoding, the binary variables are preceded by an imagined bit, which is always 1, and followed by an imagined bit, which is always 0.

$x_0x_1x_2$	y
1 0000	0
11 000	1
111 00	2
1111 0	3

Figure 2.13.: This table demonstrates the domain wall encoding for a variable $y \in \{0, 1, 2, 3\}$, with | visualising the domain wall. The variable bits are preceded and followed by an imagined bit in grey.

In the case of the domain wall encoding, whether the variable y takes the i -th value v_i is checked by taking the difference of the relevant two neighbouring binary variables to detect whether a domain wall is present at that location:

$$c_{dw}(y, v_i) = \begin{cases} 1 - x_0 & \text{if } i = 0 \\ x_{i-1} - x_i & \text{if } 0 < i < k - 1 \\ x_{k-2} - 0 & \text{if } i = k - 1 \end{cases} \quad (2.37)$$

If a domain wall is detected at i , this check can, in principle, either be 1 if a one is followed by a zero (10) or -1 if a zero is followed by a one (01). We will refer to the latter case as an inverted domain wall. But since any state that contains an inverted domain wall is invalid, the check can only return 0 or 1 in valid states. Generally, a domain wall encoding is in a valid state if, including the imagined bits, it contains only one domain wall. Therefore, the penalty term to penalise invalid domain wall states can be written as the number of domain walls minus one:

$$p_{dw}(y) = \left(-1 + \sum_i c(y, v_i)^2 \right) \quad (2.38)$$

2.3.4. The Variational Quantum Eigensolver (VQE)

The variational quantum eigensolver was proposed by Peruzzo et al. in 2014 [PMS⁺14]. Its purpose is to find an approximation of the eigenstate with the lowest eigenvalue for a Hamiltonian, also called its ground-state, which, among others, has applications in chemistry as well as in material and drug design [PMS⁺14]. To achieve this, it makes use of the variational principle, which minimises the upper bound for a Hamiltonian's minimum eigenvalue.

For a quantum state $|\psi\rangle$ and a Hamiltonian H , this upper bound to the minimum eigenvalue is given by Equation 2.39 [TCC⁺22]. Since for a valid quantum state $|\psi\rangle$, the product

2. Foundations

$\langle\psi|\psi\rangle$ must be 1 due to the property of state normalisation from Equation 2.7, the expectation value $\langle\psi|H|\psi\rangle$ provides the upper bound to the minimum eigenvalue of the Hamiltonian. This can be explained by the fact that, as explained in Subsection 2.2.3, $\langle\psi|H|\psi\rangle$ is the expectation over the eigenvalues of H that can be observed when repeatedly measuring $|\psi\rangle$ with respect to the observable Hamiltonian H . As a result, when the expectation takes the value E , eigenvalues of value E or lower must have been observed.

$$E_{min}(H) \leq \frac{\langle\psi|H|\psi\rangle}{\langle\psi|\psi\rangle} \quad (2.39)$$

The VQE algorithm exploits this fact by using a parameterised quantum circuit to initialise a trial state $|\psi(\theta)\rangle$ based on some parameter values θ . It then uses a classical optimiser in the variational workflow as outlined before to find $\arg \min_{\theta}(\langle\psi(\theta)|H|\psi(\theta)\rangle)$. This amounts to finding parameter values that minimise the upper bound of the minimum eigenvalue, which yields an approximation to the minimum eigenvalue and minimum eigenstate of the Hamiltonian.

The VQE algorithm does not propose a specific parameterised quantum circuit (ansatz) nor a specific qubit initialisation to fit all optimisation problems. Instead, both can be chosen to fit the use case at hand. With regard to the qubit initialisation, this state is typically chosen as all qubits being in state zero ($|0\rangle^{\otimes n}$) [TCC⁺22]. It can also be advantageous to select a superposition ($H^{\otimes n}|0\rangle^{\otimes n}$) or even a state based on partial knowledge about the solution as an initial state [BCLK⁺22].

A lot of research has been done on the ansatz choice for VQE. In general, the ansätze from current research can be roughly classified into problem-inspired and hardware-efficient ansätze [BCLK⁺22]. Problem-inspired ansätze derive their circuit structure from the properties of the problem to be solved. Hardware-efficient ansätze, on the other hand, are created solely with the capabilities of the quantum hardware in mind and are not based on aspects of the optimisation problem [TCC⁺22].

The capabilities of an ansatz can be discussed in terms of *expressibility* and *trainability*. Expressibility refers to how much of the total state space the ansatz can reach, whereas trainability refers to how easy it is to learn good parameters for an ansatz [TCC⁺22]. These two properties have been shown to have an inverse relationship, meaning that in selecting a good ansatz, a trade-off between expressibility and trainability has to be made [HSCC22].

A major inhibitor of the trainability of VQE ansätze is the barren plateau problem. This refers to the phenomenon that, in some cases, the gradients of the objective function may decrease exponentially [TCC⁺22]. This can make classical optimisation in such search spaces computationally hard. It has been shown that both high expressibility [HSCC22] and quantum circuit depth [WFC⁺21] can cause barren plateaus in VQE ansätze. As a result, barren plateaus can inhibit both less expressible but deep problem-inspired ansätze and more expressible but shallower hardware-efficient ansätze [WFC⁺21].

2.3.5. The Quantum Approximate Optimisation Algorithm (QAOA)

The quantum approximate optimisation algorithm was originally proposed by Farhi et al. in 2014 [FGG14]. Just like the VQE algorithm, its purpose is to find an approximation of the ground-state eigenvalue of a Hamiltonian by minimising the upper bound for its minimum eigenvalue. The difference from VQE is that in QAOA, the ansatz is designed in a way to

approximate the process of quantum annealing. As a result, QAOA is a special type of VQE algorithm with strict design rules for the ansatz structure.

Quantum annealing is a quantum optimisation process on specialised quantum hardware. It is based on the adiabatic principle, which states that a quantum system will remain in its ground state if it is changed sufficiently slowly. Quantum annealing makes use of this by initialising a quantum system in the ground state of a simple starting Hamiltonian H_s . Over time, the system is then slowly changed to reflect the problem Hamiltonian H_p . As long as this change is applied slowly enough, the final state of the system should be a ground state of the problem Hamiltonian H_p [YRBS22]. This process can also be described by a time-dependent Hamiltonian, where the time-dependent factors $A(t)$ and $B(t)$ decrease from one to zero and increase from zero to one, respectively:

$$H(t) = A(t) \cdot H_s + B(t) \cdot H_p \quad (2.40)$$

To approximate this process on gate-based quantum computers, QAOA uses an ansatz that alternates between applying the initial (also called mixer) Hamiltonian H_M and the problem Hamiltonian H_P [BCLK⁺22]. Each combined application of the mixer and problem Hamiltonian forms one layer of the QAOA ansatz, governed by two parameter values, β_i and γ_i , respectively. The parameter values can be interpreted as the degree to which the respective Hamiltonian is applied. Repeating those layers p times yields the QAOA ansatz, which offers $2p$ parameter values to be optimised. The initial state to which this ansatz is applied is typically the equal superposition state $|+\rangle^{\otimes n}$ [BCLK⁺22]. In combination, this yields the QAOA's trial state:

$$|\psi(\gamma, \beta)\rangle \equiv e^{-i\beta_p H_M} e^{-i\gamma_p H_P} \dots e^{-i\beta_1 H_M} e^{-i\gamma_1 H_P} |+\rangle^{\otimes n} \quad (2.41)$$

This is a discretisation of the continuous quantum annealing process, called trotterisation. For small p , it is an approximation, while for $p \rightarrow \infty$, this yields the quantum annealing process exactly [BCLK⁺22].

2.4. Classical Optimisation Algorithms

VQAs need a classical optimisation algorithm to optimise the parameter values of the ansatz. This section explains some foundations of classical optimisation algorithms in general and adds some details on the SPSA algorithm and evolutionary algorithms, which are used within this thesis.

2.4.1. Optimisation Problems

Optimisation is the minimisation (or maximisation) of an objective function f over a parameter vector x , where the individual parameter values may be constrained by equality constraints \mathcal{E} and inequality constraints \mathcal{I} . This can be formalised with Equation 2.42 [NW06]. Since minimisation and maximisation are exchangeable by changing the sign from $f(x)$ to $-f(x)$, the following explanations will only focus on minimisation.

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ & \text{subject to} \quad c_i(x) = 0, i \in \mathcal{E}, \\ & \quad \quad \quad c_i(x) \geq 0, i \in \mathcal{I}, \end{aligned} \quad (2.42)$$

2. Foundations

Optimisation problems can be distinguished based on several properties that impact the best choice of optimisation algorithms. For instance, the parameter values can be discrete or continuous, constrained or unconstrained, and the objective function can be linear or non-linear [NW06]. Since in the context of VQAs the parameter values denote continuous rotation angles, the corresponding classical optimisation problem is continuous. As rotations are periodic in their effect, any real number is a valid rotation angle, which means that there is no direct need to constrain the value range of the parameters in VQAs. Finally, the fact that the objective function of VQAs has been shown to exhibit local minima [Nan19] [AK22] implies that the objective function of VQAs is typically non-linear. This yields the classical optimisation task in VQAs as an unconstrained, continuous, non-linear optimisation problem.

The best solutions for such an optimisation problem are global minima of the objective function. A global minimum x^* is a parameter vector for which no other parameter vector with a lower objective function value exists [NW06]:

$$\forall x \in \mathbb{R}^n : f(x^*) \leq f(x), \quad (x^* \in \mathbb{R}^n) \quad (2.43)$$

Often, finding the global minimum of an optimisation problem is a computationally hard task. It can then be desirable to find a local minimum instead, which can be easier to find and may represent a solution that is good enough. For a local minimum x' , it suffices that no parameter vectors in its neighbourhood $\mathcal{N}(x')$ have a lower objective value [NW06]:

$$\forall x \in \mathcal{N}(x') : f(x') \leq f(x), \quad (x' \in \mathbb{R}^n) \quad (2.44)$$

2.4.2. Optimisation Algorithms

Optimisation algorithms solve optimisation problems iteratively. They start with an initial guess of the parameter vector x . They then repeatedly create a new, likely improved, guess from the previous guess. These guesses are also called iterates. This process is continued until the algorithm terminates. The way in which this process is implemented varies between optimisation algorithms [NW06]. Depending on whether optimisation algorithms try to find a global or a local minimum, they can be classified as local or global optimisation algorithms.

Local Optimisation Algorithms

To find a local minimum close to an initial guess x , local optimisation algorithms often make use of gradient information on the objective function. Such algorithms are called gradient-based algorithms. They use the gradient information to make steps in the general direction of the negative gradient $-\nabla f$ from their current guess x . This allows such algorithms to iteratively improve their guesses until they reach a local minimum in which the gradient is zero. The simplest gradient-based method is the simple gradient descent algorithm, which updates its guesses with a learning rate of η by following the direction of the negative gradient directly: $x_{n+1} = x_n - \eta \cdot \nabla f(x)$ [TCC⁺22]. Other gradient-based methods use more intricate update rules and sometimes even higher-order gradients.

Typically, gradient-based optimisation methods calculate the gradient of the objective function analytically. For some objective functions, this can be expensive. It can even be impossible if the exact objective function is unknown. In such cases, an approximation $g(x)$ of the gradient $\nabla f(x)$ can be calculated by sampling values from the objective function. A

common approach to this is to use the finite difference method [BCCS22]. It approximates the gradient by introducing small differences for each parameter value individually and observing the resulting changes in the objective function value. If the parameter values are perturbed in both directions, this is also called the central finite differences method. With this method, the i -th component of the approximated gradient $[g(x)]_i$ is defined as follows, where σ is the size of the perturbation and e_i is the unit vector, which is one in the dimension i and zero elsewhere [BCCS22]:

$$[g(x)]_i = \frac{f(x + \sigma e_i) - f(x - \sigma e_i)}{2\sigma} \quad (2.45)$$

This approximation method uses $2n$ evaluations of the objective function, where n is the dimensionality of the parameter vector [BCCS22]. To prevent the number of function evaluations from scaling with the parameter vector's dimensionality, the simultaneous perturbation approximation can be used. It needs only two function evaluations to perturb all parameter values simultaneously. The gradient is then assigned to the parameters based on the measured difference and the extent to which each parameter was perturbed [Spa98]. With this method, the i -th component of the approximated gradient $[g(x)]_i$ is defined as follows, where σ is the size of the perturbation and Δ is the random perturbation vector, which indicates the direction of the perturbation [Spa98]:

$$[g(x)]_i = \frac{f(x + \sigma \Delta) - f(x - \sigma \Delta)}{2\sigma \Delta_i} \quad (2.46)$$

When using simultaneous perturbation in a gradient descent optimisation algorithm to approximate the gradient, the resulting algorithm is called simultaneous perturbation stochastic approximation (SPSA). Multiple papers on VQAs have found SPSA to be a well-performing classical optimiser for VQAs [Loc22] [MFP⁺22] [BMWV⁺23].

Another way to deal with the unavailability of gradient information is to use local optimisation algorithms, which do not rely on gradient information at all. Such algorithms are also called gradient-free algorithms. In classical optimisation, the choice of a gradient-free optimisation algorithm typically limits the accuracy or expense of the optimisation when compared to gradient-based methods [LMW19]. This is corroborated to some extent for the use case of VQAs, where studies have shown that the gradient-based SPSA algorithm frequently outperforms other gradient-free optimisation methods [MFP⁺22] [BMWV⁺23].

Global Optimisation Algorithms

Finding and identifying the global minimum of an objective function can be computationally difficult for many optimisation problems [NW06]. As a result, meta-heuristics are commonly used in global optimisation. Meta-heuristics are general-purpose optimisation algorithms that do not employ problem-specific knowledge and thus are applicable to a wide range of optimisation problems. They typically do not offer guarantees on finding the global minimum and instead try to find as good a solution as possible while keeping their resource usage efficient [SEBB22]. Among such meta-heuristics, the most common global optimisation algorithms are population-based [RDD23]. That means that for each iteration step of the optimisation, they evaluate a group of parameter guesses, which enables them to explore a wider region of the search space. In evolutionary algorithms (EAs), which can be regarded

2. Foundations

as the state of the art in population-based optimisation [SEBB22], this iteration process is inspired by biological evolution.

Evolutionary Algorithms

Biological evolution is based on two pillars: competition-based selection and mutation. Competition-based selection refers to the fact that in a population of individuals that are constrained by limited resources, natural selection favours those that compete most effectively for these resources. The most effective individuals are then most likely to reproduce and create offspring. Mutation refers to the fact that reproduction can introduce small, random variations in the offspring. Over many generations, the combination of these processes explores many variations while selecting for the most useful ones, pushing the general fitness of the population to increase [ES15].

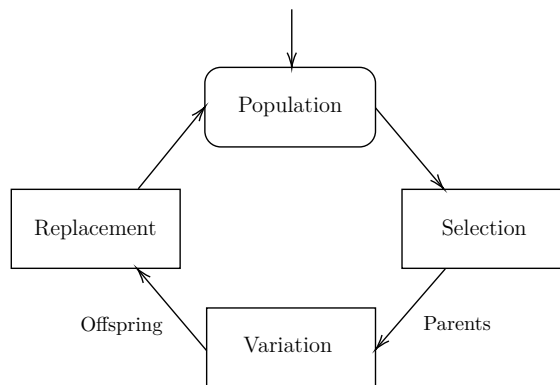


Figure 2.14.: This figure shows the general workflow of evolutionary algorithms, in which selection, variation, and replacement are used to, over many generations, improve the fitness of the individuals within the population.

In EAs, individuals in a population represent a solution to the optimisation problem, which is encoded in the individual's genome. The effectiveness with which an individual competes in the population is rated by a fitness function, which is typically closely related to but not necessarily equal to the objective function to be minimised.

Mirroring biological evolution, EAs use an iterative loop of selection, variation, and replacement operations (see Figure 2.14) to increase the fitness of the individuals over many generations and, as a result, find better solutions to the optimisation problem [BOM15]. Selection stochastically selects individuals from the population to serve as parents for the next generation in a way that is biased towards selecting fitter individuals. From the selected parents, new individuals are created as random variations of their parents, either by mutation of single parents or by crossover of the genes of two or more parents. Finally, the new individuals are used to replace all or some of the individuals in the previous population to form the next generation of the population [BOM15]. The specifics of these operations vary between EAs.

3. Related Work

In this chapter, an overview of the related work relevant to this thesis is given. To that end, we first give a quick overview of current classical approaches to solving the JSSP. This then motivates the investigation of quantum optimisation methods for solving the JSSP. From these, the most relevant related work is on quantum annealing and VQA approaches to solving the JSSP. The issues of the VQA approaches for solving the JSSP then lead to research on general VQA approaches that adapt their ansatz circuit during the optimisation procedure. From there, we identify a research gap, which motivates the main research question of this thesis.

Classical Approaches

The state of the art on classical approaches for solving the JSSP has been reviewed many times in the last two decades [JM⁺98] [GS06] [ÇB15] [ZDZ⁺19] [LHWK21]. Generally, JSSP optimisation methods can be classified into exact and approximate solving methods [ZDZ⁺19] [LHWK21].

Exact methods are guaranteed to find an optimal solution to the JSSP problem. These methods include, among others, branch and bound methods, as well as mixed-integer programming [Lom65] [Man60]. Yet, since the JSSP is NP-hard, these algorithms cannot solve the JSSP to optimality in polynomial time [GJS76] [LHWK21]. This leads to even medium-sized problem instances taking a long time to solve. Take, as an example, mixed-integer programming solving software such as CPLEX, GUROBI, or SCIP [IBM22] [Gur24] [BBC⁺23]. They have been shown to need up to an hour to solve JSSP problems with 15 jobs and 15 machines to optimality on a consumer-grade computer [KB16]. This can be an issue for time-critical scheduling problems.

In such cases, approximate methods can be used. They are not guaranteed to find optimal solutions [LHWK21] but instead sacrifice optimality for efficiency. An illustration of this fact can be seen in the work of Zhang et al., whose approximate algorithm was able to find approximate solutions to JSSP instances with 15 jobs and 15 machines in less than three minutes [ZLRG08]. Common approximate methods for solving the JSSP include constructive and meta-heuristic approaches [ZDZ⁺19]. The constructive approaches, like the priority dispatch rule method, are heuristics that build a schedule step by step, starting from an empty schedule [BM85]. The meta-heuristic approaches include approaches such as evolutionary algorithms and tabu-search [FB91] [BC95].

Quantum Annealing Approaches

In the 1990s, quantum annealing was proposed as a heuristic for solving combinatorial optimisation problems (which also includes the JSSP) on specialised quantum computing hardware [FGS⁺94] [KN98] [YRBS22]. While it has been shown that quantum annealing is not yet capable of challenging the state of the art in classical optimisation algorithms, recent

3. Related Work

studies have shown that in some special cases, quantum annealing can outperform classical heuristics [TAM⁺22] [JC23].

While this thesis is not focused on progress in quantum annealing, it is still valuable to investigate how previous research on quantum annealing for solving the JSSP has dealt with encoding the problem as a Hamiltonian.

In 2015, Venturelli et al. presented a foundational paper on solving the JSSP as a decision problem using quantum annealing [VMR15]. Solving the JSSP as a decision problem means that their quantum annealing approach only searches for valid JSSP solutions and does not discern solutions based on the makespan (the time needed to finish all jobs). To achieve this, they use a time-indexed problem formulation in which each operation o in a job is assigned a collection of binary variables $x_{o,t}$, ($t \in \{0, \dots, T\}$), each of which determines whether the operation is scheduled to commence at t . Only one of the variables per operation may be 1 at a time, as an operation cannot start at two different times. This can also be understood as a one-hot encoding of the integer start time for each operation. Such binary, time-indexed problem formulations are also known from mixed-integer linear programming approaches [KB16]. The number of binary variables per operation is bounded by the choice of T . Venturelli et al.'s QUBO Hamiltonian formulation then consists of penalty terms, which increase the energy of invalid states. This includes penalty terms to penalise invalid encoding states. It also includes quadratic penalty terms of the form $\alpha \cdot x_{o_1,t_1} \cdot x_{o_2,t_2}$ for each combination of binary variables, where scheduling the operation o_1 at t_1 and scheduling the operation o_2 at t_2 would result in a conflict. As long as the penalty weights α are chosen to be bigger than zero, this yields a Hamiltonian in which all its ground-states have an energy of 0 and are valid solutions to the given JSSP. Furthermore, all invalid states have an energy greater than 0. Venturelli et al. further provide a way to prune the operation start time variables $x_{o,t}$ by observing that an operation can never start before all its preceding operations in a job have finished. They further observe that given a time limit T , any operation cannot start so late that its subsequent operations in a job do not have sufficient time to finish within the time limit. Therefore, the possible start times t of an operation are bounded from below by the length of all operations that precede it and bounded from above by T minus the length of this operation and all its subsequent operations in a job. With this Hamiltonian formulation, Venturelli et al. were able to solve JSSP problem instances of up to 6 jobs and 6 machines on a D-WAVE quantum annealer. While the quantum annealer was slower than classical solvers, the authors were optimistic that future quantum annealers might alleviate this.

Further benchmarks based on Venturelli et al.'s approach have been done by Carugno et al. [CFC22]. They solved JSSP decision problem instances of up to 26 jobs and 26 machines using both quantum annealing and classical optimisation algorithms. Their results were mixed, with quantum annealing sometimes overperforming and sometimes underperforming the classical solving algorithms.

An extension to Venturelli et al.'s approach for solving the JSSP has been provided by Kurowski et al. [KWS⁺20], who use a sliding window to decompose a JSSP problem into smaller sub-problems, which can more easily be solved on a quantum annealer. Kurowski et al. tested this approach on a JSSP problem instance with six jobs and six machines, but found that classical algorithms still outperformed their approach.

Other recent work on quantum annealing has approached solving other variants of the JSSP, such as the single-stage JSSP, the dynamic JSSP, the flexible JSSP, or the dynamic JSSP [AHY22] [SWGA23a] [SWGA23b] [SWK⁺24] [TZS24].

VQA Approaches

In comparison to research on quantum annealing, less research seems to have been done on optimising scheduling problems by using variational quantum algorithms.

Plewa et al. have investigated the use of VQAs, namely QAOA and VQE, to optimise workflow scheduling problems using quantum simulators [PSR21]. The workflow scheduling problem instances they investigated contained 3–4 machines, 3–4 tasks, and needed 10–15 qubits. They compared the efficacy of different integer encodings, namely one-hot, domain wall, and binary encodings. They found that the use of denser encodings, which reduce the number of invalid states, can lead to better results and allow for the optimisation of larger problem instances. When comparing the QAOA and VQE algorithms, Plewa et al. found that for the workflow scheduling problem instances they investigated, VQE performed better than QAOA but was more impacted by the choice of the classical optimiser than QAOA. They also observed that increasing the ansatz depth of the VQAs too much led to a deterioration of the obtained results. For QAOA, they found the optimal amount of layer repetitions to be two, whereas for VQE, the optimal amount of ansatz layers was one.

Amaro et al. have also compared the performance of several VQA algorithms for optimising the JSSP on real quantum hardware [ARF⁺22]. Specifically, they investigated an optimisation variant of the JSSP in which the last operation of each job is assigned a due date (a specific time at which the operation should finish). The optimisation goal is then not to minimise the makespan but to minimise the total deviation from the due dates, which includes both finishing an operation too late or too early. Like Venturelli et al. in their quantum annealing approach, Amaro et al. encode this problem as an Ising Hamiltonian by using a one-hot encoding to map the operation start times to binary variables and then adding penalty terms to increase the energy of invalid states. On top of that, they then add an optimisation term to assign higher energies to states with a higher deviation from the due dates. They then compared four VQAs, namely QAOA, VQE, VarQITE, and F-VQE, for solving a problem instance that needed 5 qubits. They found that QAOA converged much slower and had a lower probability of observing the ground state at the final measurement than the other algorithms. Amaro et al. reasoned that this was partially due to the problem-inspired QAOA ansatz not being suited to the quantum computing hardware, as the other

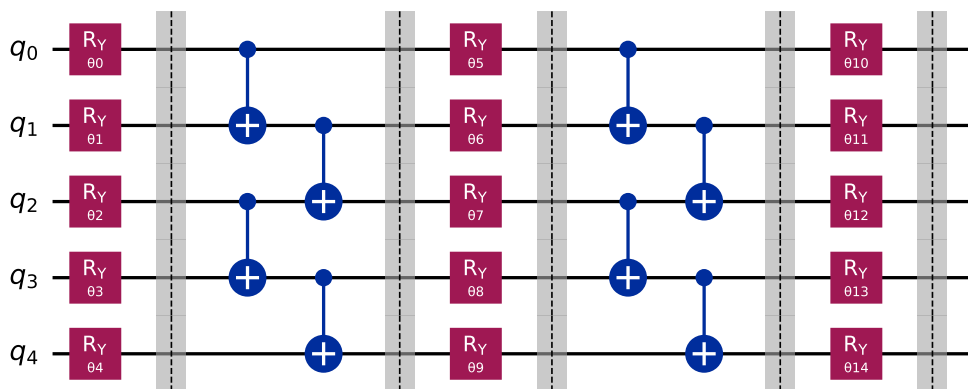


Figure 3.1.: This figure shows the hardware-efficient VQA ansatz used in Amaro et al.’s comparison of VQA algorithms with two layer repetitions.

3. Related Work

algorithms all used the hardware-efficient ansatz shown in Figure 3.1. The other algorithms all converged to states with a reasonable likelihood of measuring the ground state. As F-VQE displayed the fastest convergence and highest likelihood of measuring the ground state, Amaro et al. then focused on benchmarking F-VQE for larger problem instances needing up to 23 qubits. For JSSP problem instances of increasing size, it could then be observed that the convergence speed slowed and the likelihood of measuring the ground state decreased significantly. This seems to imply that the optimisation landscape is challenging to optimise over, even for the short hardware-efficient VQA ansatz used here.

Finally, Kurowski et al. [KPS⁺23] investigated the use of QAOA to optimise the makespan of the JSSP using quantum simulators. As a base, they use the same Hamiltonian formulation as Venturelli et al., to which they add a term that penalises solutions with a higher makespan, so that states with a lower makespan are always assigned a lower energy than states with a higher makespan. This penalty term consists of one penalty per job, of the form $(n_J + 1)^t$, where n_J is the number of jobs in the problem instance and t is the time at which the last operation of the specific job is finished. Since it holds that $n_J \cdot (n_J + 1)^t < (n_J + 1)^{t+1}$, the sum of the individual penalties is always smaller for all jobs ending at t than even for one job ending at $t + 1$. With the problem encoded as a Hamiltonian, Kurowski et al. then employ a strategy that exploits the fact that given a QAOA ansatz of p layers with good parameter values, good initial parameter values for an ansatz of $p + 1$ layers can be interpolated due to patterns in the parameter values [VGS⁺20] [ZWC⁺20]. Kurowski et al. call this the educated guess strategy. First, Kurowski et al. optimise the parameters, starting from random values, multiple times to find good parameters for a QAOA ansatz with three layers. Once good initial parameters are found, the amount of layers is increased, and the initial parameters for the new ansatz are interpolated. Starting from the interpolated parameters, the parameters are then optimised again. This loop is repeated until good enough measurement results are found. Using this routine, Kurowski et al. were able to get high measurement probabilities for the ground state of the Hamiltonian. But the fact that the authors determined 500 random initial points were needed to find good parameters for the three-layered ansatz again seems to indicate that the optimisation landscape is challenging to optimise over for the problem-inspired QAOA ansatz used here.

Adaptive Ansatz Approaches

As outlined in Section 2.3.4 on the VQE algorithm, the ansatz choice strongly influences the trainability of any VQA. This is due to both hardware-efficient shallow ansätze and problem-inspired deep ansätze being affected by vanishing gradients and bad local minima, which makes learning good parameter values difficult [HSCC22] [WFC⁺21] [AK22]. The experimental results from the previous subsection seem to indicate that the VQA approaches for solving the JSSP are likely not exempt from these problems. This makes it interesting to investigate adaptive ansatz approaches that iteratively grow an ansatz during the optimisation process by selecting and appending operations to the ansatz that contribute most towards minimising the objective function [TCC⁺22]. The hope is that such approaches can alleviate issues such as barren plateaus and bad local minima by using shallower ansätze and by adjusting the optimisation landscape during the optimisation process.

Among the first approaches suggested in this manner are the Adaptive Derivative Assembled Pseudo-Trotter ansatz VQE (ADAPT-VQE) algorithm and the Qubit-ADAPT-VQE algorithm by Grimsley et al. and Tang et al. [GEBM19] [TSB⁺21]. These algorithms

were designed for the simulation of chemical systems. The ansatz of these algorithms is initialised in an initial state suitable to the optimisation problem. Then a collection of unitary operators is chosen, which can be used to extend the ansatz. In ADAPT-VQE, these operators are selected to be fermionic operators suitable to the chemical system that is to be simulated, whereas in Qubit-ADAPT-VQE, these are picked to be a selection of Pauli strings. Both algorithms are then an iterative loop of operator selection and parameter optimisation. The operators are chosen by separately extending the ansatz with each operator and determining for which operator the gradient at the current parameter values is the steepest. This operator is then selected and added to the ansatz. Then, all parameter values are optimised. This iterative loop terminates once the gradients for all operators fall below a certain threshold. Claudino et al. found in their evaluation of ADAPT-VQE that while both VQE and ADAPT-VQE provided solutions with good accuracy, ADAPT-VQE was much more resilient with regard to the choice of the classical optimisation algorithm [CWMH20]. Grimsley et al. found in subsequent work that the combination of local optimisation and adaptation of the search space, by growing the ansatz, makes ADAPT-VQE very resilient to local minima and barren plateaus [GBB⁺23].

Inspired by the ADAPT-VQE algorithm, Zhu et al. introduced an adaptive ansatz approach for QAOA, which they called Adaptive Derivative Assembled Problem Tailored QAOA (ADAPT-QAOA) [ZTB⁺20]. In their approach, they iteratively add QAOA layers and optimise the parameter values. The central aspect is that the mixer part of the added QAOA layer is not fixed but chosen from a pool of mixer operators so that the gradient with the added QAOA layer is as large as possible. Zhu et al. used ADAPT-QAOA to solve random Max-Cut problem instances and found that ADAPT-QAOA needed much fewer ansatz layers than standard QAOA to converge to good solutions.

Another approach has been introduced by Bilkis et al.'s work on the Variable Ansatz (VAns) algorithmic framework [BCV⁺23]. It treats the ansatz and parameter optimisation tasks as two nested optimisation loops. The ansatz optimisation is the outer optimisation loop, and it optimises discrete variables, which characterise the ansatz structure. This involves adapting the ansatz, which can be done by ansatz insertion and ansatz simplification. In ansatz insertion, a unitary operator is randomly selected from a pool of preselected operators. It is then, according to some rules, placed in the ansatz, with its parameters initialised, so that the inserted unitary evaluates to the identity operator. In the simplification step, superfluous quantum gates and quantum gates with a low impact on the objective function are removed. Once a step in the outer optimisation loop has been done, the parameters are optimised in the inner optimisation loop to evaluate how good the outer optimisation step was. This then enables the outer optimisation step to be accepted or rejected. Bilkis et al. tested VAns against VQE with hardware-efficient ansätze of comparable length and found VAns to yield superior results. Using noisy simulations, they also found VAns to be much less impacted by noise than the hardware-efficient VQE. The authors also expect that the short circuits produced by VAns could mitigate the effect of noise-induced barren plateaus.

As can be seen from this related work, adaptive ansatz methods are promising because they generally yield better results while using shorter ansatz circuits than non-adaptive approaches. On top of that, preliminary results on their resistance to noise, local minima, and barren plateaus also seem promising. Yet, due to their repeated parameter optimisation as part of their iterative routine, they need a very high number of measurements of the expectation value with respect to the problem Hamiltonian. This is especially an issue, as access to real quantum hardware is still limited and expensive. It also calls into question

3. Related Work

whether the benefits of adaptive ansatz methods outweigh their added computational costs.

Adaptive Ansatz Approaches using Evolutionary Algorithms

Another type of adaptive ansatz approach that has recently gained significant attention is the use of evolutionary algorithms to find both a good ansatz structure and good parameter values [BCV⁺23]. In these evolutionary algorithms, each individual in a population represents an ansatz structure and its parameter values. Variation, selection, and replacement of these individuals over multiple generations enables the search over multiple ansatz structures and parameter values in parallel. This lets these algorithms explore a wider range of ansatz structures, which should make it less likely for them to be hindered by local minima [RHP⁺19]. On top of that, the fittest ansatz structures often appear multiple times in a population. This means that they are evaluated multiple times per generation, which should average out the impact of quantum noise on good solutions [RHP⁺19].

The first approach proposed in that manner was the Evolutionary VQE (EVQE) algorithm proposed by Rattew et al. [RHP⁺19]. In EVQE, the individuals consist of an ansatz circuit consisting of a sequence of random circuit layers as well as the accompanying parameter values. The circuit layers are constructed from U_3 , controlled U_3 gates, and identity gates, so that each qubit is modified by exactly one gate. For an example of an EVQE individual with two layers, see Figure 3.2. The initial population is initialised with one random circuit layer per individual. Mutations can then, with a certain probability, optimise all parameter values of an individual, one circuit layer at a time. Other mutations can add new random circuit layers or remove circuit layers to reduce the ansatz depth. Before evaluating the fitness of an individual, the parameters of its last circuit layer are optimised. The fitness of an individual is then based on the ansatz’s expectation value with respect to the Hamiltonian, penalised by the ansatz depth and its controlled gate count. In their evaluation of EVQE, Rattew et al. demonstrate that EVQE can produce ansätze that are up to 18 times shallower and use up to 12 times fewer controlled gates than problem-inspired ansätze. They also showed that EVQE learns to construct ansätze that are very noise-resistant, leading to EVQE being significantly less affected by noise than VQE.

Similar to EVQE is the Multiobjective Genetic VQE (MoG-VQE) approach proposed by Chivilikhin et al. [CSU⁺20]. In their approach, the ansatz of an individual is constructed

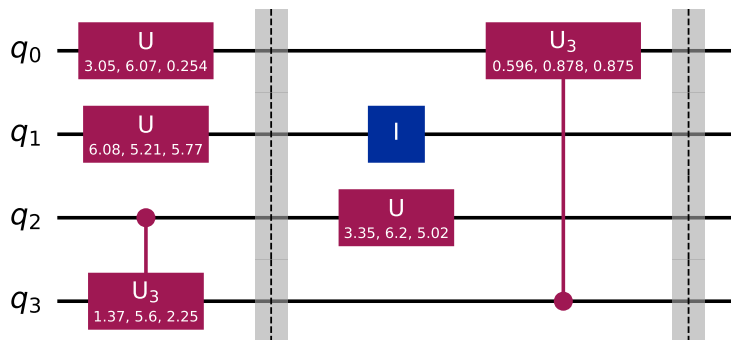


Figure 3.2.: This figure shows an EVQE individual with two circuit layers. The separation between layers is indicated by the grey barriers.

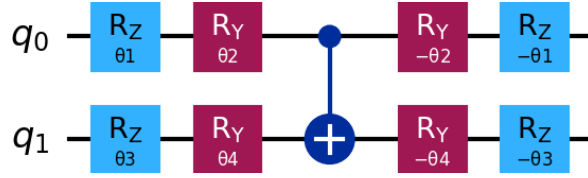


Figure 3.3.: This figure illustrates the gate block structure used in the MoG-VQE algorithm [CSU⁺20]. Combinations of multiple such gate blocks form an ansatz.

using multiple gate blocks (see Figure 3.3), where these gate blocks can be placed on any two qubits of the ansatz circuit. The initial population is generated with individuals, which can use either a structured or a random gate block placement. The mutations in MoG-VQE only adapt the ansatz structure by either adding gate blocks at random positions in the ansatz or by removing random gate blocks from the ansatz. Before the selection of fit individuals, all parameters for all individuals are optimised. As fitnesses, Chivilikhin et al. use both the expectation value and the CNOT gate count separately, as CNOTs are particularly expensive and noisy operations. The individuals are then selected in accordance with the multi-objective NSGA2 algorithm. Chivilikhin et al. compared their approach to VQE using a hardware-efficient ansatz and found that MoG-VQE was able to reach accurate solutions while using significantly fewer CNOT gates.

With the Quantum Neuroevolution of Augmenting Topologies (QNEAT) algorithm, another approach similar to EVQE and MoG-VQE was recently presented by Giovagnoli et al. [GTMS23]. In their algorithm, the gates in the ansatz of an individual can only be placed according to a layered, fixed structure, with the genome of the individual determining which of these gates is actually placed. See Figure 3.4 for an illustration of an example individual. Its first layer showcases the full layer structure, with all possible quantum gates allowed by the fixed structure being placed. The other layers adhere to the same structure but do not contain all the gates that could be placed. Evolutionary mutations can then vary the individual by adding gates to the ansatz, removing gates, or randomly perturbing the parameter values of the individual. In contrast to EVQE and MoG-VQE, the variation of the

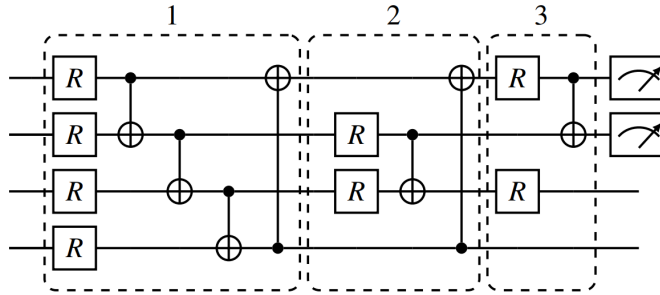


Figure 3.4.: This figure illustrates the ansatz structure of the QNEAT algorithm. Taken from Giovagnoli et al.’s paper on QNEAT [GTMS23] (CC BY 4.0 License¹). In the first layer of the individual, all possible gates are placed, showcasing the structure of a layer. In the other layers, not all gates have been placed.

¹<https://creativecommons.org/licenses/by/4.0/>

3. Related Work

individual can not only be achieved with mutation but also with crossover. This combines the gates and parameter angles from two parent individuals into a new individual based on some crossover rules and randomness. Giovagnoli et al. then benchmarked QNEAT on reinforcement learning and the Max-Cut combinatorial optimisation task. For combinatorial optimisation, they compared QNEAT against QAOA and found that QNEAT was capable of achieving similar accuracies to multi-layer QAOA while using significantly fewer quantum gates.

In summary, evolving ansatz approaches improve on adaptive ansatz approaches by leveraging the advantages of the parallel optimisation of multiple ansatz structures. This could enable evolving ansatz approaches to be even more resistant to noise, local minima, and barren plateaus. Yet, like the adaptive approaches, the evolving ansatz approaches also suffer from the high amount of expectation value measurements they need. In fact, this problem should even be more severe for evolving ansatz approaches due to their parallel approach, which increases the number of evaluated ansatz structures per iteration drastically. Whether the benefits provided by evolving ansatz methods are worth the cost likely comes down to how evolving methods scale for increasing problem sizes and, as a result, increasing amounts of qubits.

Research Gap and Research Question

Yet, with little conclusive research on both the scaling of adaptive ansatz and especially the scaling of evolving ansatz methods, it remains unclear how beneficial such methods can be for problem instances of increasing sizes [TCC⁺22]. This is an important gap in current research. On top of that, it seems like neither adaptive ansatz approaches nor evolving ansatz approaches have been applied to solve the NP-hard JSSP. This motivates this thesis to investigate the scaling of evolving ansatz methods when applied to the JSSP, which yields the main research question of this thesis:

How do evolving ansatz VQE algorithms scale in terms of computational effort and solution quality for job shop scheduling problems?

To limit the extent of this thesis, this scaling is only investigated for the EVQE algorithm presented by Rattew et al. [RHP⁺19]. Notably, two main factors influence this choice. The first being that the evaluation of EVQE is more detailed when compared to MoG-VQE and QNEAT, with only EVQE being evaluated in a noisy scenario. This makes the results shown for EVQE seem the most promising. Secondly, the search space of possible ansatz structures proposed in EVQE is less constricted and uses more general quantum gates than that of MoG-VQE and QNEAT. This seems to indicate that of these algorithms, EVQE is the most suitable for the general investigation of the scaling of evolving ansatz methods.

4. The Evolutionary VQE Algorithm

In this chapter, the EVQE algorithm, presented by Rattew et al. [RHP⁺19], is described on a conceptual level as it is implemented for this thesis. This means that, in some cases, details may be changed from the original EVQE algorithm. This is particularly the case where not enough details were given in the original paper and gaps had to be filled in. To prevent confusion, such changes will be clearly declared. This chapter starts with an overview of the steps involved in the EVQE's repeating evolutionary routine in Section 4.1. Following that, the population and its individuals are explained in Section 4.2. Then the selection procedure is explained in Section 4.3. Finally, the variation of the individuals is explained in Section 4.4, and the termination criterion is explained in Section 4.5.

4.1. Overview

In Figure 4.1, the general steps involved in the EVQE algorithm are visualised. The state of the EVQE optimisation process is represented by its population of individuals, which improves as the evolutionary process continues. Each of these individuals represents a separate parameterised ansatz consisting of at least one circuit layer with an assignment of parameter values. To increase the quality of the population over multiple generations, selection, variation, and replacement are applied to the population. The selection is based on the evaluation of the ansatz individuals. This involves optimising the last layer of all individuals and then measuring their expectation values to gauge how good an individual is. Once fit individuals have been selected, variation is then applied to create slightly different individuals based on the selected individuals to create the next generation of the population. Since the creation of new individuals only involves one parent per new individual, this is also called asexual reproduction. The newly generated individuals of the new generation then entirely replace the individuals from the previous generation. As can be seen in Figure 4.1, this workflow of evaluation, selection, variation, and replacement forms a cyclic procedure, where each turn equals one generation in the EVQE algorithm. This procedure is repeated until a termination criterion determines that the optimisation process is done. This then ends the procedure, and the best individual, therefore the best ansatz and parameter value combination, is returned. Measuring the state provided by this ansatz should then offer, with high likelihood, a good solution to the optimisation problem encoded in the Hamiltonian.

4.2. Population

The population of the EVQE algorithm consists of individuals, each of which represents an ansatz structure with parameter values. In this section, it is first explained how the genome of an individual codifies that information. Then it is explained how random genes (circuit layers) are generated. This is then used to explain how the population is initialised. Finally, speciation is explained, which is used to keep the population diverse.

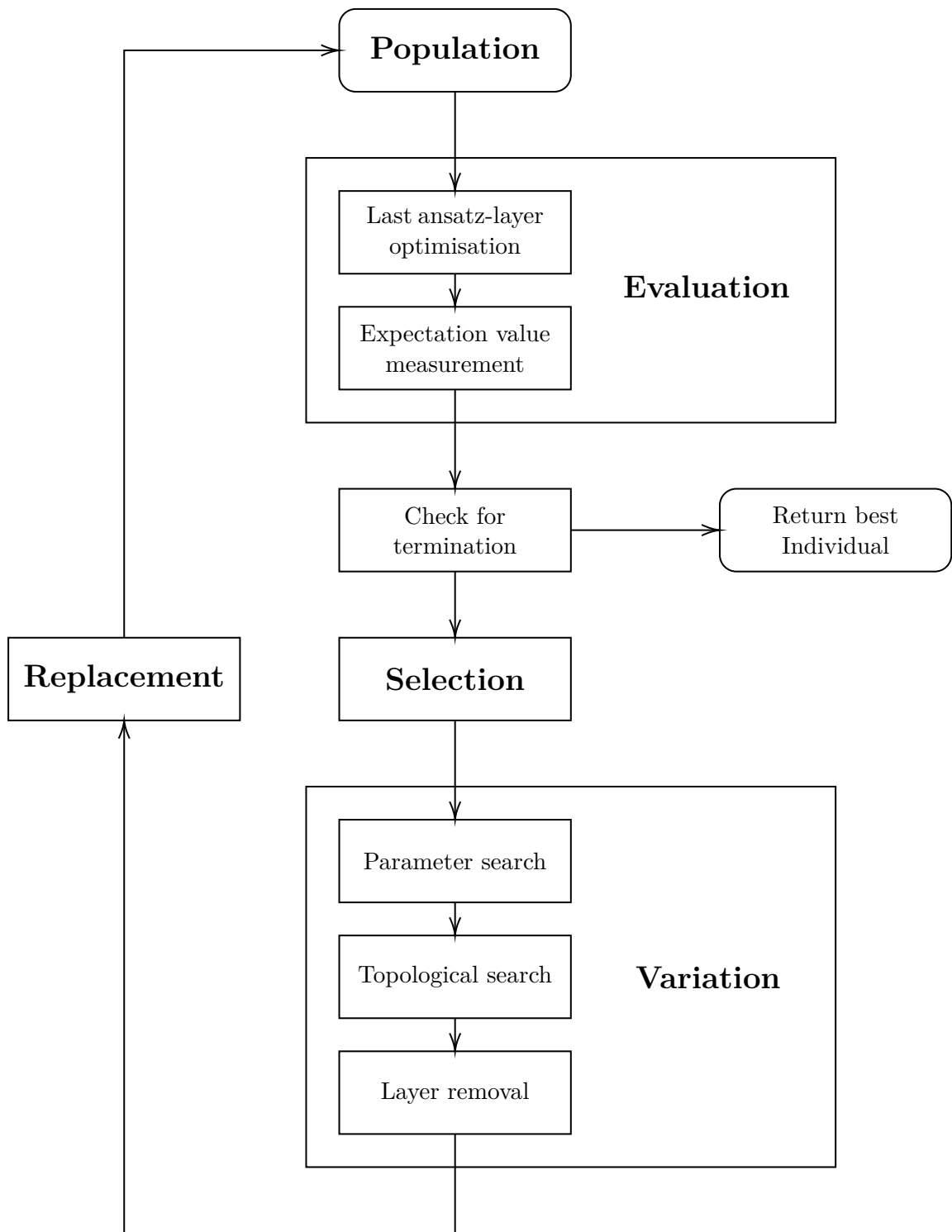


Figure 4.1.: This figure shows a graphical representation of the EVQE algorithm's cyclical evolutionary optimisation procedure.

4.2.1. Individual Genomes

An individual in EVQE represents a parameterised ansatz circuit consisting of at least one circuit layer and the accompanying parameter values. All the information necessary to construct that ansatz circuit and populate the parameter values is stored within the individual's genome.

The most fundamental parts of the genome are genes. Each gene γ_k represents the structure of a unique parameterised circuit layer. As such, each gene is an assignment of quantum gates, namely the U3 gate, the controlled U3 gate, and the identity gate, to the individual qubits of the ansatz circuit, so that each qubit is assigned exactly one gate. For the purposes of gate assignment, the control part of any controlled gate also counts as a gate. This means that on a qubit controlling a gate, no other gate may be placed in the same gene. The number of qubits in the ansatz is derived from the dimension of the Hamiltonian, which the algorithm is meant to solve. The subscript k for the gene γ_k is the unique identifier of the gene. Note that this only means that genes can be compared for equivalence. In particular, EVQE does not make use of a prespecified pool of genes. Instead, genes are randomly generated when needed.

Within the genome of an individual, parameter values specific to that individual are applied to its genes. This application of parameter values to a gene γ_k yields a gene instance γ_k^i for the individual i . Newly initialised gene instances start out with all parameters being assigned the value 0. As the parameter values of the gene instances of an individual are adapted throughout the evolutionary process, gene instances inherited from previous generations will often contain non-zero parameter values.

The genome of an individual can then be fully described as a sequence of gene instances $g_i = (\gamma_1^i, \dots, \gamma_k^i)$. An illustration of the distinction between genes, gene instances, and the genomes they form, can be seen in Figure 4.2, which was created by Rattew et al. for their paper on the EVQE algorithm [RHP⁺19].

If one writes $U(\gamma_k^i)$ for the unitary operator, which the gene instance γ_k^i represents, then the quantum state produced by the ansatz and parameter values encoded by the genome g_i is given by the following formula:

$$|\psi_{g_i}\rangle = U(\gamma_k^i) \cdot \dots \cdot U(\gamma_0^i) \cdot |0\rangle^{\otimes n} \quad (4.1)$$

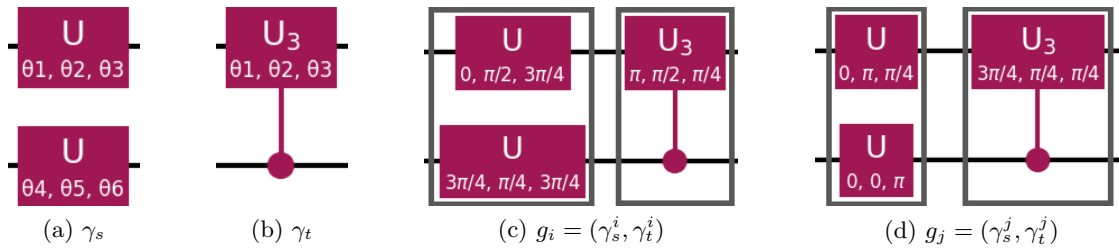


Figure 4.2.: This figure illustrates the differences between genes, genomes, and gene instances in EVQE [RHP⁺19]. (a) and (b) show two different genes γ_s and γ_t , each representing a circuit layer, without fixed parameter values. (c) and (d) show the genomes of two different individuals, i and j . They each consist of two gene instances, highlighted by the grey boxes.

4.2.2. Random Gene Generation

Whenever a new gene instance needs to be initialised, either to extend a genome or initialise a new genome, a random gene needs to be created as a base for that gene instance first. Due to the fact that genes represent the structure of an ansatz layer consisting of quantum gates, gene generation involves choosing quantum gates and assigning them to the individual qubits of the ansatz. Since Rattew et al. do not outline their procedure for gate choice and placement during the gene generation, we first outline our basic procedure for random gate placement. After that, we discuss how the gate placement procedure can be adapted to prevent redundant gates in the ansatz.

Basic Random Gate Placement Procedure

For the basic gate placement procedure, we choose between U3 and CU3 gates. We try to assign gates fairly, so that a qubit should have the same chance of being assigned a CU3 or a U3 gate. To achieve that, first, for each qubit, we randomly designate which type of gate can be assigned to it. With an equal likelihood of 50%, it is either designated for the placement of a U3 gate or a CU3 gate. To then place a CU3 gate on two qubits, two qubits are chosen uniformly at random from the qubits designated for usage with CU3 gates. The CU3 gate is then assigned to these two qubits, so that it is controlled by the first qubit and controls the second qubit. These qubits are then removed from the list of designated qubits to prevent any further gates being assigned to them. This procedure is repeated until no more than one qubit remains designated for the CU3 gates. Since a CU3 gate cannot be assigned to only one qubit, on this qubit and all other qubits designated for the placement of U3 gates, U3 gates are placed. Figure 4.3 demonstrates the entire gate placement procedure for the generation of a gene on 4 qubits.

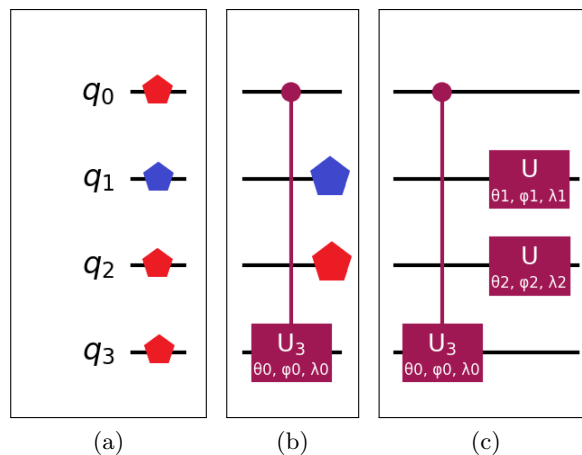


Figure 4.3.: This figure illustrates the stages of the basic random gene generation. (a): First, the qubits are either designated for a CU3 (red pentagon) or a U3 (blue pentagon) gate with a 50% chance each. Then the CU3s are randomly placed (b). Finally, the U3 gates are placed (c).

Conditional Random Gate Placement Procedure

If a gene is generated to extend an already existing genome, the newly assigned quantum gates extend the ansatz specified by that genome. In that case, care needs to be taken during the gene generation to prevent the placement of redundant quantum gates, which do not extend the capabilities of the ansatz. In particular, Rattew et al. list the following as redundant gate placements: A U3 gate directly following a U3 gate on the same qubit is redundant, as the same effect could be achieved by merely adjusting the parameters of the first U3 gate. The same goes for a CU3 gate directly following another CU3 gate, on the same qubits and in the same orientation. Such gate placements would increase the number of parameters in the ansatz, without extending the reachable search space. Therefore, Rattew et al. forbid such gate placements. The restriction of redundant gate placements can lead to situations in which no gate can be assigned to a qubit. In that case, an identity gate can be assigned to it. Due to the fact that after a CU3 gate, a U3 gate is always permitted to be placed, no identity gate can ever be placed after a CU3 gate. The converse does not always hold true. After a U3 gate, during the gate placement procedure, there may not be enough qubits left to place a CU3 gate. This implies that an identity gate will only ever follow a U3 gate. As a consequence, no U3 gate can be placed after an identity gate, as the identity gate between the U3 gates would not prevent their redundancy.

To comply with this restriction on gate placements, slight modifications need to be made to the basic random gate placement procedure if it is used when extending a pre-existing genome. First, if a U3 gate or an identity gate is assigned to a qubit on the previous gene, the same qubit may not hold a U3 gate on the newly generated subsequent gene. This means that this qubit is directly designated as a CU3 qubit, without randomness. The same cannot be done for CU3 gates on the previous gene, as another CU3 gate could be placed on the same qubits but in the inverse orientation without being redundant. Therefore, the qubits to which, in the previous gene, CU3s were assigned are again designated for use with U3s

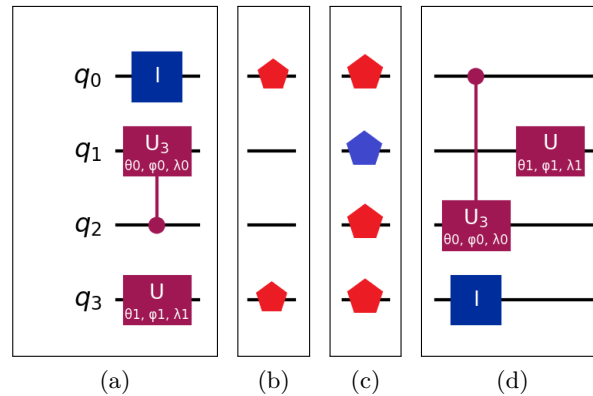


Figure 4.4.: This figure illustrates how a random gene is generated to follow the gene shown in (a) without placing redundant gates. As shown in (b), following an identity or U3 gate in (a), the qubits are directly designated as CU3 qubits (red pentagons). Following a CU3 gate in (a), the qubits are randomly designated as either U3 (blue pentagons) or CU3 qubits (c). After the random gate placement, no gate could be placed on qubit 3, leaving it with an identity gate, as shown in (d).

4. The Evolutionary VQE Algorithm

and CU3s with equal likelihood. During the placement of the CU3 gates, if a random CU3 placement is selected that is against the redundancy rules, it is discarded and re-drawn. If an excess qubit designated for CU3 placement remains that cannot be designated as a U3 qubit due to the redundancy rules, an identity gate is placed on it. Figure 4.4 demonstrates the entire conditional gate placement procedure for the generation of a gene on 4 qubits.

4.2.3. Population Initialisation

For the initialisation of the population, Rattew et al. specify that the population is initialised with *population_size* many individuals. The genome of each of these individuals is initialised to contain one gene instance derived from a random gene, with all parameter values set to zero. This creates individuals, whose ansatz acts as the identity operator. As a result, the state produced by all initial individuals is $|\psi_{g_i}\rangle = |0\rangle^{\otimes n}$.

4.2.4. Speciation and Population Diversity

An important feature of evolutionary algorithms is that their population allows them to cover a wide area of the search space. This advantage can be lost if a particularly fit individual or group of closely related individuals, over time, replaces all other individuals due to strong selection pressure. In that case, the population loses its diversity and, as a result, its ability to broadly explore the search space. This may lead to a premature convergence to a local minimum. A way to combat this problem is to group genetically similar individuals into so-called species and adjust the fitness of the individuals to penalise overly large species so that no species may take over the whole population.

This approach is followed by Rattew et al. [RHP⁺19]. To group the individuals into species, a similarity measure between individuals is needed, which in evolutionary algorithms is commonly referred to as genetic distance. For their genetic distance measure, Rattew et al. use differences in the ansatz structure to quantify the similarity of individuals. This is enabled by the asexual reproduction used in EVQE. Due to the asexual reproduction, individuals only have one parent. In addition, the ansatz structure can only be changed by adding or removing gene instances from the end of the genome, which will be further explained in Section 4.4. As a result, the number of genes at the end of the genome that are not shared between individuals roughly indicates how long ago the individuals diverged from a shared ancestor. Rattew et al. calculate this by subtracting the number of gene instances based on the same genes in the genome from the average number of gene instances in the two individuals' genomes. That calculation is shown in Equation 4.2. Note that $\lceil x \rceil$ refers to rounding x up to the next integer, $|g_i|$ refers to the number of gene instances in the genome g_i , and $g_i[k]$ refers to the k -th gene instance of the genome g_i . Furthermore, $\llbracket g_i[k] = g_j[k] \rrbracket$ is a term that is 1 if the k -th gene instance of g_i is based on the same gene as the k -th gene instance of g_j , and 0 otherwise.

$$\delta_{ij} = \lceil \frac{1}{2} \cdot (|g_i| + |g_j|) \rceil - \left(\sum_{k=1}^{|g_i|} \llbracket g_i[k] = g_j[k] \rrbracket \right) \quad (4.2)$$

Given this distance measure, Rattew et al. use a representative individual, called the species representative, to represent the whole species. Any individual within the species, declared by a species representative, must have a genetic distance to the representative that

is lower than the *genetic_distance_threshold*, which is set as a parameter of the algorithm. To group the individuals into species, the following workflow is used for each individual in the population:

1. Assign the individual to the first species, for whose species representative the genetic distance to the individual is below the *genetic_distance_threshold*.
2. If there is no such species, assign the individual as a new species representative.

This speciation procedure is carried out each generation before the selection. To maintain species continuity between generations, the initial set of species representatives used for this process consists of one random species member from each species of the previous generation.

4.3. Evaluation and Selection

In this section, it is first explained why the parameters of the last gene instance of an individual are optimised before measurement. Then it is explained which factors constitute the fitness function. Finally, it is explained how the fitness function translates to selection probabilities in the section on the selection procedure.

4.3.1. Last Layer Optimisation

In evolutionary algorithms, new individuals are created by the variation of previous individuals. The task of selection, then, is to select useful variations and filter out harmful variations. One variation operator in EVQE is "topological search". It is explained in Subsection 4.4.2. It creates offspring from individuals by appending a new gene instance to the end of their genome. Importantly, this gene instance is initialised with all parameter values at zero, so that the new gene instance acts as the identity operator. This means that the newly added gene instance at the end of the genome does not change the behaviour of the ansatz and, as such, does not impact its performance. This makes it impossible to discern whether the newly added gene instance is useful or harmful. Therefore, to help evaluate the usefulness of these variations for all individuals, the parameter values of their last gene instances are optimised before determining their fitness. If a new gene instance gained by variation is useful, this optimisation procedure should substantially improve the fitness of its individual, making it more likely to be selected. Let us now define an operation $opt(\gamma_k^i) \rightarrow \gamma_k^{i'}$, which uses a classical optimisation procedure, as explained in Section 2.4, to optimise the parameter values of exactly one gene instance to minimise the expectation value $\langle \psi_{g_i} | H | \psi_{g_i} \rangle$. The last layer optimisation operation σ on a genome g_i is then defined as follows:

$$\sigma : (\gamma_1^i, \dots, \gamma_n^i) \rightarrow (\gamma_1^i, \dots, \gamma_{n-1}^i, opt(\gamma_n^i)) \quad (4.3)$$

Note that the exact optimisation procedure used is not fixed in this specification. We reflect this in our implementation, which allows the user to freely specify an optimisation procedure, as long as it adheres to a specific interface.

4.3.2. Fitness Score

Once the last layers of all individuals in the population have been optimised, their fitness scores can be calculated. In EVQE, the fitness score of an individual consists of three

4. The Evolutionary VQE Algorithm

main components. Namely, these are the individual’s expectation value with respect to the Hamiltonian, which is penalised by both the count of gene instances in the genome $|g_i|$ and the count of CU3 gates in the genome $CU3(g_i)$. These penalties bias the evolution towards ansatz circuits, which are shallower and use fewer controlled gates. The extent of the penalties is determined by the factors α and β , which are set as parameter values of the EVQE algorithm. All values combined yield the basic fitness score for an individual i (see Equation 4.4). Note that a lower fitness score is considered better here, as finding good eigenstates of a Hamiltonian is typically a minimisation task.

$$f_i = \langle \psi_{g_i} | H | \psi_{g_i} \rangle + \alpha \cdot |g_i| + \beta \cdot CU3(g_i) \quad (4.4)$$

To encourage genetic diversity in the population, this fitness is then adjusted based on the amount of individuals $|S(i)|$ in the species $S(i)$, which is the species to which the individual i belongs. Note that lower values for f_i are better, and the fitness for individuals in large species should be worse. Therefore, we adopt the following adjusted fitness f_i^a , which penalises individuals in large species by multiplication:

$$f_i^a = f_i \cdot |S(i)| \quad (4.5)$$

This is counter to the adjusted fitness formulation presented in Rattew et al.’s paper [RHP⁺19], which seems to be an error in notation. It would lower the fitness value for large species and thus improve the fitness of individuals in large species:

$$f_i^a = \frac{f_i}{|S(i)|} \quad (4.6)$$

4.3.3. Selection of fit individuals

In EVQE, due to asexual reproduction, each offspring has one parent. Therefore, to fully replace the individuals of the previous generation with new offspring, *population_size* many individuals need to be selected as parents to seed the next generation. Rattew et al. specify that the parents are randomly drawn from the population with replacement and probabilities proportional to the adjusted fitness of the individuals. Yet, they do not specify the exact method to map fitnesses to selection probabilities. Therefore, we defined our own fitness to selection probability mapping.

To map the highest likelihood to the fitnesses with the lowest values, we first need to invert the fitness values so that the fittest individual is actually assigned the largest value:

$$inv(f_i^a) = \frac{1}{f_i^a} \quad (4.7)$$

The selection probability of an individual is then its fraction of the sum of all inverted probabilities:

$$p_{select}(i) = \frac{inv(f_i^a)}{\sum_{j \in Population} inv(f_j^a)} \quad (4.8)$$

We employ Python’s `random.choices` method to sample *population_size* many individuals with replacement according to the selection probabilities.

4.4. Variation

With the parents for the next generation selected, variation is applied to create offspring from the parent individuals. The variation operators used for this are parameter search, topological search, and layer removal. They are explained in more detail in this section. Each variation operator can be understood as creating a modified version of an individual. For each operator, there is a separate probability with which it will be applied to an individual, given as a parameter to the EVQE algorithm. The exact manner in which these operators are applied is not specified by Rattew et al. We implement it so that these operators are applied in sequence for each individual. This means that if, for an individual, multiple variations are chosen to be applied, the subsequent variations are applied to the individual resulting from the previous variation. Consequently, the child individual may be the result of zero to three mutations.

4.4.1. Parameter Search

In parameter search, all parameters of an individual are optimised. This is done gene instance by gene instance in a random order. As before, during the optimisation of one gene instance, the parameters of all other gene instances remain fixed. This procedure again makes use of a classical optimisation procedure. In our implementation, the parameter search makes use of the same classical optimisation procedure as the last layer optimisation, which is given to the EVQE algorithm as a parameter. Formally, the parameter search can be characterised as follows:

$$\pi : (\gamma_1^i, \dots, \gamma_n^i) \rightarrow (\text{opt}(\gamma_1^i), \dots, \text{opt}(\gamma_n^i)) \quad (4.9)$$

4.4.2. Topological Search

In topological search, a gene instance based on a randomly generated gene is added to the individual. The parameter values of that gene instance are initialised to be zero, so that the new gene instance acts as an identity operator. This has the advantage of not perturbing the state of the individual, so that parameter searches in future generations can smoothly continue from the individual's current state. The topological search can be characterised as shown in Equation 4.10. Figure 4.5 shows a visualisation of an example application of the topological search operator to an individual.

$$\tau : (\gamma_1^i, \dots, \gamma_n^i) \rightarrow (\gamma_1^i, \dots, \gamma_{n+1}^i) \quad (4.10)$$

4.4.3. Layer Removal

In layer removal a uniformly random number of gene instances $k \in \{1, \dots, n-1\}$ is removed from the end of the individual's genome, where n is the number of gene instances in the individual's genome. This variation can change the fitness of an individual drastically. The layer removal can be characterised as shown in Equation 4.11. Figure 4.6 shows a visualisation of an example application of the layer removal operator to an individual.

$$\rho : (\gamma_1^i, \dots, \gamma_n^i) \rightarrow (\gamma_1^i, \dots, \gamma_{(n-k)}^i) \quad (4.11)$$

4. The Evolutionary VQE Algorithm

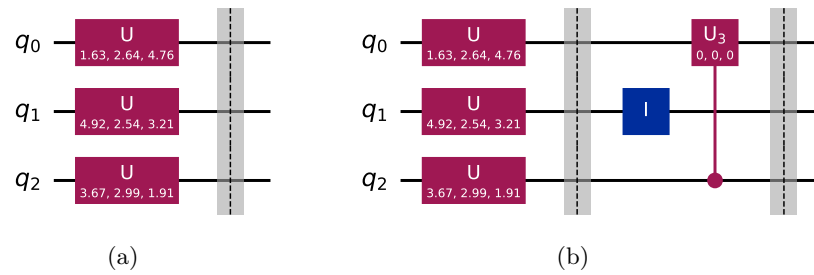


Figure 4.5.: This figure shows an individual before (a) and after (b) topological search was applied.

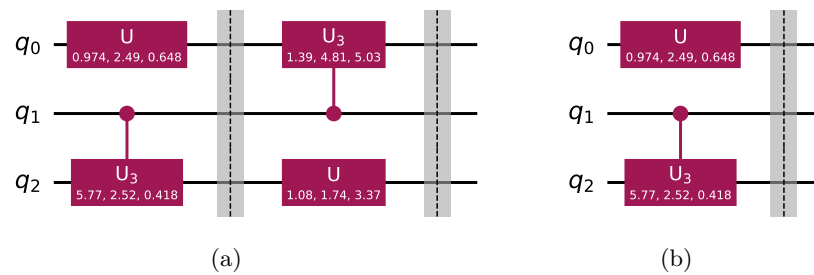


Figure 4.6.: This figure shows an individual before (a) and after (b) layer removal for one layer was applied.

4.5. Termination Criteria

For the termination, in our implementation, we enable three types of termination criteria. The first termination criterion terminates after a fixed number of evolutionary generations. The second terminates after a certain number of expectation value measurements have been exceeded. This can be useful if access to real quantum hardware is expensive or limited. The final termination criterion can be user-specified as long as it adheres to a certain interface. This termination criterion receives the evaluation results for each generation and can, based on these results, choose to terminate the optimisation. This allows the user to specify a wide range of termination criteria. We employ this capability in our benchmarks to let the EVQE algorithm run as long as it needs to converge. The exact termination criterion used for the benchmarking procedure is explained in Chapter 7.

5. Encoding the JSSP as a Hamiltonian

In this chapter, it is explained how the JSSP, as defined in Section 2.1, can be expressed as a QUBO problem. The QUBO problem can then in turn be converted to a Hamiltonian, as explained in Subsection 2.3.2. To express the JSSP as a QUBO problem, the JSSP's discrete variables first need to be encoded into binary variables, as explained in Subsection 2.3.3. From there, a quadratic function $f(x)$ on the vector x of binary variables needs to be defined. It needs to assign a value to each possible state of x that expresses the quality of the solution to the JSSP instance, as provided by this state of x . This function should be designed in such a way that finding $\operatorname{argmin}_x f(x)$ returns a state of x , which translates to an optimal solution of the JSSP. Note that the goal is minimisation, which means that lower values are better. We will refer to these values as energies to be more consistent with Hamiltonian terminology. In particular, it is required that $f(x)$ assigns:

- a high energy to all states of x that violate the encoding of the discrete variables
- a high energy to all states of x that translate to an invalid solution to the JSSP
- a low energy to all states of x that translate to a valid solution to the JSSP
- a lower energy to states of x that translate to valid solutions with lower makespans

To fulfil this requirement, we structure the quadratic function $f(x)$ to consist of separate terms. The term $\operatorname{opt}(x)$ describes the optimisation part of the function and yields lower energies for better makespans. The term $p_{\operatorname{enc}}(x)$ penalises states in which encoded variables do not contain valid values by yielding high energies for invalid encoding states. The term $p_{\operatorname{prc}}(x)$ penalises states in which at least one operation violates the precedence constraint in between the operations of a job by yielding high energies for such states. An example would be the second operation of a job starting before the first operation of a job. The term $p_{\operatorname{ovl}}(x)$ penalises states in which any two operations, which are executed on the same machine, overlap. This is also done by assigning such states high energies. Summing all these terms together yields the quadratic function $f(x)$ as shown in Equation 5.1.

$$f(x) = \operatorname{opt}(x) + p_{\operatorname{enc}}(x) + p_{\operatorname{prc}}(x) + p_{\operatorname{ovl}}(x) \quad (5.1)$$

To fulfil the requirements on the state energies, it is particularly important that Equation 5.2 holds. This ensures that invalid states are always of higher energy than valid states.

$$p_{\operatorname{enc}}(x) + p_{\operatorname{prc}}(x) + p_{\operatorname{ovl}}(x) = \begin{cases} > \max(\operatorname{opt}(x)) & \text{if } x \text{ is invalid} \\ 0 & \text{if } x \text{ is valid} \end{cases} \quad (5.2)$$

To explain how the individual terms of the QUBO function $f(x)$ are constructed, notations are defined in Section 5.1. Then, the value ranges of the discrete variables and their encoding is explained in Section 5.2. Afterwards, the construction of the penalty terms is explained in Section 5.3. Finally, the optimisation term is explained in Section 5.4. An example of the resulting energy landscape is then shown in Section 5.5.

5.1. Definitions

As explained in Section 2.1, each job in the JSSP consists of an ordered sequence of operations, which are assigned a duration and a machine on which to execute. We define the set of all operations in a JSSP instance as \mathbb{O} , the set of all jobs in a JSSP instance as \mathbb{J} , and the set of all machines in a JSSP instance as \mathbb{M} . We now define that each operation $o \in \mathbb{O}$ is assigned a unique identifier $i \in \mathbb{N}$. This allows specific operations to be denoted as o_i .

To access relevant properties of each operation o , we define mappings from the operations to their properties, so that $machine(o) : \mathbb{O} \rightarrow \mathbb{M}$ maps the operation to the machine on which it is executed. We also define $job(o) : \mathbb{O} \rightarrow \mathbb{J}$ so that the operation is mapped to the job to which it belongs. We further define $duration(o) : \mathbb{O} \rightarrow \mathbb{N}$ to map operations to the duration for which they need to be processed and $position(o) : \mathbb{O} \rightarrow \mathbb{N}$ to map the operation to its position within the ordered sequence of operations of the job.

As a useful shorthand, we also define \mathbb{O}_j and \mathbb{O}_m to be the set of operations that are executed on a specific job or machine respectively:

$$\mathbb{O}_j = \{o_i \in \mathbb{O} \mid job(o_i) = j\} \quad (5.3)$$

$$\mathbb{O}_m = \{o_i \in \mathbb{O} \mid machine(o_i) = m\} \quad (5.4)$$

5.2. Variables and Variable Encoding

The goal of the JSSP is to schedule all operations in such a way that the time for all jobs to finish (makespan) is as low as possible. This scheduling can be done by finding a good start time t for each operation. We assume that both the start time t and the processing duration of an operation can be specified in discrete time units, where the earliest possible start time is 0. If the start time of any operation can be delayed indefinitely, there are infinitely many possible start times to choose from. To combat this, like Venturelli et al., we introduce an upper bound T for the makespan [VMR15]. This upper bound needs to be at least as large as the minimum makespan of the problem instance to be solved. Otherwise, no valid solutions can be found. Furthermore, it should not be chosen too large to prevent the number of possible start times from becoming too large to be represented by the limited number of qubits of a quantum computer. For real applications, the minimum makespan is typically unknown, in which case, multiple optimisation trials with varying upper bounds would have to be run to find a solution. For our benchmarks, we choose T based on knowledge of the minimum makespan.

As Venturelli et al. showed, the upper bound enables the available start times for each operation to be bounded by considering the length of the operations preceding and following the operation in its job. The length of all operations preceding an operation in a job can be defined as:

$$len_{prc}(o_i) = \sum_{\substack{o_j \in \mathbb{O}_{job(o_i)}, \\ position(o_j) < position(o_i)}} duration(o_j) \quad (5.5)$$

The length of all operations subsequent to an operation o_i in the same job can also be defined as:

$$len_{ssq}(o_i) = \sum_{\substack{o_j \in \mathbb{O}_{job(o_i)}, \\ position(o_j) > position(o_i)}} duration(o_j) \quad (5.6)$$

Note that the operations preceding an operation o_i in a job need to be finished, before o_i can be scheduled to start. This means that $len_{prc}(o_i)$ provides the lower bound for the start time of o_i . Similarly, given the makespan limit T , both o_i and all operations subsequent to o_i in this job need to be able to finish within T . This means that $T - (duration(o_i) + len_{ssq}(o_i))$ provides the upper bound for the start time of o_i . As a consequence, the set of all available start times for an operation o_i can be defined as:

$$stimes(o) = \{t \in \mathbb{N} \mid len_{prc}(o_i) \leq t \leq T - (duration(o_i) + len_{ssq}(o_i))\} \quad (5.7)$$

In most cases, the set of available start times will contain more than two values. In such cases, the discrete variable modelling the choice between those start times has more than two states. This means that, as discussed in Subsection 2.3.3, the discrete start time variable for each operation needs to be encoded onto multiple binary variables. In their approach, Venturelli et al. have used the one-hot encoding [VMR15]. Yet, test results by Plewa et al. have shown denser encodings to perform better during the optimisation process [PSR21]. Therefore, we choose to encode the discrete optimisation variables as domain wall variables. Note that we assign the possible start times to the domain wall variable's states in order. Therefore, the first domain wall state $1|00\dots00$ refers to the earliest possible start time, the second domain wall state $11|0\dots00$ refers to the second-earliest start time, and so on.

To distinguish the domain wall variables from the binary QUBO variables, we will refer to the domain wall variable for representing the possible start times of an operation o_i as y_{o_i} . The vector of all domain wall variables for all operations o_i is then referred to as y . The function to check whether a domain wall variable y_{o_i} contains the value v : $c_{dw}(y_{o_i}, v)$ remains as defined in Equation 2.37. The QUBO objective function can then also be rewritten in terms of the variable vector y , which we will use for simplicity's sake going forward:

$$f(y) = opt(y) + p_{enc}(y) + p_{prc}(y) + p_{ovl}(y) \quad (5.8)$$

5.3. Penalties

In this section, the makeup of the individual penalty terms are explained. Note that to regulate their interplay, we provide weighting parameters w_{enc} , w_{prc} , and w_{ovl} to increase or decrease the weighting of certain constraints.

5.3.1. Encoding Penalties

To penalise invalid encoding states for all domain wall variables, we sum their penalty terms $p_{dw}(y_{o_i})$, as defined in Equation 2.38, for each individual domain wall variable. Note that the individual penalty terms need to be adjusted by a scaling factor $s(y_{o_i})$ to prevent issues, which will be explained in more detail in Subsection 5.3.4. The resulting penalty term is:

$$p_{enc}(y) = w_{enc} \cdot \left(\sum_{o_i \in \mathbb{O}} s(y_{o_i}) \cdot p_{dw}(y_{o_i}) \right) \quad (5.9)$$

5.3.2. Precedence Penalties

Violations of the precedence constraints occur when an operation of a job is scheduled to start before one of its preceding operations is finished. To penalise such states, all combinations of start time values that violate this condition need to be found. To formalise that, first the set of tuples of directly subsequent operations is defined:

$$ssq_ops(\mathbb{O}_j) = \{(o_i, o_k) \in \mathbb{O}_j \times \mathbb{O}_j \mid position(o_i) + 1 = position(o_k)\} \quad (5.10)$$

Given two subsequent operations, the start time combinations for which they violate the precedence constraint are then given by each combination in which the second operation starts before the first operation has finished:

$$prc_viol(o_i, o_k) = \{(t_i, t_k) \in stimes(o_i) \times stimes(o_k) \mid t_k < t_i + duration(o_i)\} \quad (5.11)$$

With these definitions, the penalty term, which applies penalties to all precedence constraint violations, is then defined as:

$$p_{prc}(y) = w_{prc} \cdot \left(\sum_{j \in \mathbb{J}} \sum_{\substack{(o_i, o_k) \in \\ ssq_ops(\mathbb{O}_j)}} \sum_{\substack{(t_i, t_k) \in \\ prc_viol(o_i, o_k)}} c_{dw}(y_{o_i}, t_i) \cdot c_{dw}(y_{o_k}, t_k) \right) \quad (5.12)$$

5.3.3. Overlap Penalties

Violations of the overlap constraint occur when two operations on a machine overlap due to their scheduling, resulting in a machine having to process two operations at once, which is impermissible. To check for all combinations of operations and start times that are impermissible, first the set of all possible operation combinations for a given machine is defined in Equation 5.13. Note that the comparison of the identifiers $i < k$ serves to ensure that each combination of operations appears only once in $pair(\mathbb{O}_m)$, as it excludes one of the two possible orderings for each pair of operations.

$$pair(\mathbb{O}_m) = \{(o_i, o_k) \in \mathbb{O}_m \times \mathbb{O}_m \mid o_i \neq o_k \wedge i < k\} \quad (5.13)$$

Given a combination of two operations, the impermissible start time combinations are then given by combinations that lead to operations overlapping at some point. These can be identified by testing whether both operations start before the other has finished:

$$ovl_viol(o_i, o_k) = \{(t_i, t_k) \in stimes(o_i) \times stimes(o_k) \mid \begin{aligned} &t_i < t_k + duration(o_k) \wedge \\ &t_k < t_i + duration(o_i) \end{aligned}\} \quad (5.14)$$

The penalty term for the overlap constraint violations can then be defined as:

$$p_{ovl}(y) = w_{ovl} \cdot \left(\sum_{m \in \mathbb{M}} \sum_{\substack{(o_i, o_k) \in \\ pair(\mathbb{O}_m)}} \sum_{\substack{(t_i, t_k) \in \\ ovl_viol(o_i, o_k)}} c_{dw}(y_{o_i}, t_i) \cdot c_{dw}(y_{o_k}, t_k) \right) \quad (5.15)$$

5.3.4. Invalid Penalty Interactions

An important fact is, that the penalty formulations for both the precedence constraint and the overlap constraint depend on $c_{dw}(y, v)$ only ever being zero or one. But as explained in the section on encodings, if an inverted domain wall is placed on v , $c_{dw}(y, v)$ evaluates to minus one. This means that the penalties in which $c_{dw}(y, v)$ is involved will flip their sign due to $c_{dw}(y, v)$ becoming negative. If $c_{dw}(y, v)$ is now part of n precedence and k overlap constraints with the respective weights w_{prc} and w_{ovl} , this reduces the energy by $2 \cdot (n \cdot w_{prc} + k \cdot w_{ovl})$. At the same time, any domain wall variable containing an inverse domain wall ($\dots 0|1 \dots$) must contain at least three domain walls, as its first and last imaginary bits are fixed. This is showcased in the following, where $|$ again denotes these domain walls:

$$1 \dots | \dots 0 | 1 \dots | \dots 0 \quad (5.16)$$

Since the valid domain wall variable states contain only one domain wall, placing an inverted domain wall increases the number of domain walls from one to three. This increases the energy by $2 \cdot w_{enc}$. Therefore, the destruction of the domain wall variable's validity is worth it if $c_{dw}(y, v)$ is part of n precedence and k overlap constraints and the following holds:

$$2 \cdot w_{enc} < 2 \cdot (n \cdot w_{prc} + k \cdot w_{ovl}) \quad (5.17)$$

This leads to the reasoning for the scaling term $s(y_{o_i})$ in the encoding penalties, as shown in Equation 5.9. Assume that the maximum number of constraints on any value of the domain wall variable is known. Also assume that the encoding penalty weight is at least as large as the other penalty weights ($w_{enc} \geq w_{prc} \wedge w_{enc} \geq w_{ovl}$). In that case, multiplying $p_{dw}(y_{o_i})$ by $s(y_{o_i}) = n + k + 1$ ensures that destroying the domain wall variable always worsens (increases) the energy. This is due to the fact, that destroying the domain wall variable then increases the energy by $2(nk + 1) \cdot w_{enc}$, which is larger than the maximum decrease in energy $2 \cdot (n \cdot w_{prc} + k \cdot w_{ovl})$.

5.4. Optimisation Goal

This section explains the formulation of the optimisation goal. As outlined beforehand, its highest value for valid states should be lower than any of the penalty weights $w_{enc}, w_{prc}, w_{ovl}$. This ensures that even states with the worst possible makespan are preferable to invalid solutions. Furthermore, it must also be ensured that minimising the optimisation goal minimises the makespan of the solution. The formulation of the optimisation goal in this case consists of two parts, the makespan minimisation term $mmin(y)$ and the early start term $estart(y)$, whose relative influence is modified by a continuous factor γ in the range $[0, 1]$. The makespan minimisation term rewards states with a lower makespan, whereas the early start term rewards states for starting all operations as early as possible. Both $mmin(y)$ and $estart(y)$ are normalised to the value range $[0, 1]$. The maximum size of the optimisation term is then regulated by the weight w_{opt} :

$$opt(y) = w_{opt} \cdot ((1 - \gamma) \cdot mmin(y) + \gamma \cdot estart(y)) \quad (5.18)$$

5.4.1. Makespan Minimisation

For the makespan minimisation term, we follow the approach proposed by Kurowski et al. [KPS⁺23]. As explained in Chapter 3, their approach involves applying a weight $(n_J + 1)^t$ to the last operation of each job, where n_J is the number of jobs and t is the time at which the particular operation ends. To reiterate, since it holds that $n_J \cdot (n_J + 1)^t < (n_J + 1)^{t+1}$, the worst state of a lower makespan is always assigned a lower energy than a state with a higher makespan. Let the set of the last operations of all jobs now be defined as:

$$last(\mathbb{O}) = \{o_i \in \mathbb{O} \mid \forall_{o_k \in \mathbb{O}_{job(o_i)}} : position(o_k) \leq position(o_i)\} \quad (5.19)$$

Since the makespan minimisation term reaches its maximum value if all jobs end exactly at the maximum makespan T , its maximum value is given as $n_J \cdot (n_J + 1)^T$. This allows the makespan minimisation term to be normalised by division. This results in the following term for $mmin(y)$:

$$mmin(y) = \frac{\left(\sum_{o_i \in last(\mathbb{O})} \sum_{t \in stimes(o_i)} c_{dw}(y_{o_i}, t) \cdot (n_J + 1)^{t+duration(o_i)} \right)}{n_J \cdot (n_J + 1)^T} \quad (5.20)$$

5.4.2. Early Start for all Operations

To minimise the makespan, it is not enough to schedule the last operation of a job earlier, as its preceding operations must also be scheduled earlier to prevent violations of the precedence constraint. This can make it difficult for an optimiser to reduce the makespan if it can only see the makespan as an optimisation goal, as scheduling the preceding operations earlier by themselves is not rewarded. This motivates the introduction of the early start term, which rewards the scheduling of all operations as early as possible.

Let $enum(stimes(o_i))$ now be the set of all tuples (t, j) of the start times $t \in stimes(o_i)$, each with a corresponding index $j \in \{0, \dots, |stimes(o_i)| - 1\}$, so that a smaller index j equals a smaller start time. Assigning this index as a weight to each start time for each operation then provides a term, which encourages the scheduling of all operations as early as possible. This term is also normalised by division. This results in the following term for $estart(y)$:

$$estart(y) = \frac{\sum_{o_i \in \mathbb{O}} \sum_{(t,j) \in enum(stimes(o_i))} c_{dw}(y_{o_i}, t) \cdot j}{\sum_{o_i \in \mathbb{O}} |stimes(o_i)| - 1} \quad (5.21)$$

5.5. Resulting Energy Landscape

To showcase the energy landscape resulting from the QUBO construction, we visualise the energy for all states of a small problem instance using 2 jobs and 2 machines. This instance can be seen in Figure 5.1. The optimal makespan for this problem instance is 3. For the QUBO construction, we select a makespan limit T of 4. We further choose $w_{enc} = 300$, $w_{pre} = 150$, $w_{ovl} = 150$. We also set $w_{opt} = 100$ and $\gamma = 0.25$. The resulting QUBO needs eight binary variables. As a result, it expresses the energy of $2^8 = 256$ different states. In the following visualisations, these individual states will be referred to by their integer index, retrieved by reading the state bitstrings as an integer.

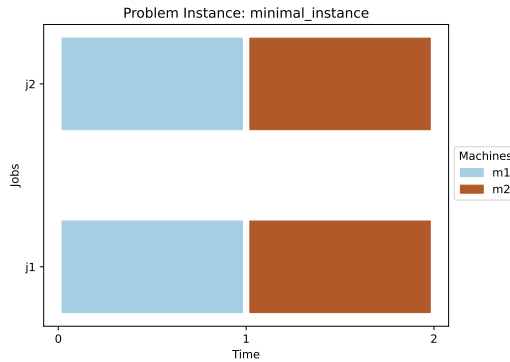


Figure 5.1.: This figure shows a small JSSP problem instance using 2 jobs and 2 machines. Its minimum makespan is 3.

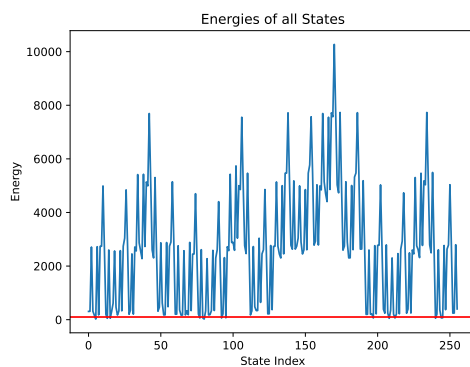


Figure 5.2.: This figure visualises the full energy landscape for the small JSSP instance. The red line denotes $w_{opt} = 100$. Any state with less energy than that is valid. All other states are invalid. It can be seen that the vast majority of states are invalid.

Plotting the energy of these states in Figure 5.2 yields a very jagged energy landscape, with the wide majority of states having high energies. Since most of the states are above the maximum value w_{opt} , which the optimisation term may take, it can be surmised that the vast majority of the states are invalid.

This does not come as a surprise, as the QUBO formulation here employs four domain wall variables with two qubits each. Each of those domain wall variables has only three valid values. All possible combinations of these variable values yield only $3^4 = 81$ valid domain wall encoding states, which is less than a third of the 256 available states. Of course, the valid domain wall states can in turn be invalid according to the scheduling rules of the JSSP, which further drastically reduces the amount of valid states. This is showcased in Figure 5.3, in which all energies larger than w_{opt} (invalid states) are filtered out. This leaves only 14 states, which are fully valid in accordance with both the domain wall encoded variables and the JSSP scheduling rules. Note that there are only two global minima, representing the only solutions with a makespan of 3. Figure 5.4 shows the two schedules that achieve these makespans. All other solutions yield a makespan of 4, which is why they exhibit much higher energy values.

5. Encoding the JSSP as a Hamiltonian

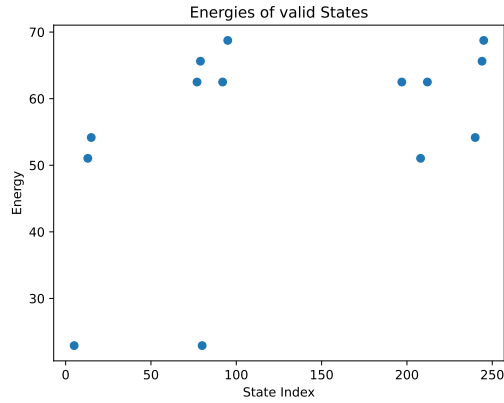


Figure 5.3.: This figure visualises the energy landscape for the small JSSP instance if all energies larger than $w_{opt} = 100$ are filtered out. The remaining energies are energies for fully valid states. The global minima at $x = 5$ and $x=80$ both have an identical energy of 22.9.

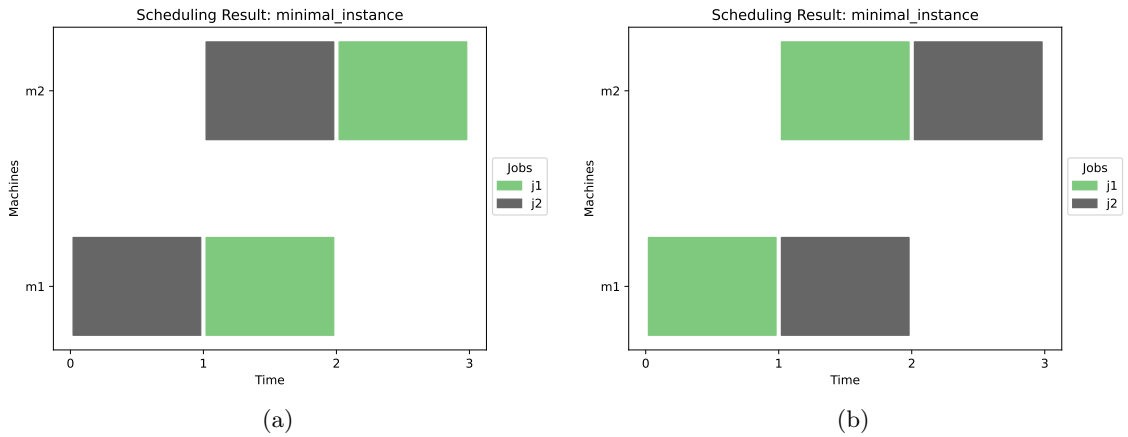


Figure 5.4.: This figure shows the solutions to the small JSSP instance with makespan 3. These correspond to the global minima at (a) $x = 5$ and (b) $x = 80$ of the QUBO's energy landscape. Both represent ideal solutions with makespan 3, differing only in their ordering of the jobs.

6. EVQE - Issues and Improvements

In this chapter, a few issues that were encountered during preliminary testing of the EVQE algorithm are explained, and fixes to these issues are presented. These fixes take the form of slight alterations to the EVQE’s optimisation procedure. In our implementation, we offer these fixes as optional parameters for the EVQE algorithm.

6.1. Individual Initialisation

During preliminary tests of the EVQE algorithm on some randomly generated JSSP instances (see Section 7.3 for more details), the EVQE algorithm mostly performed well. Yet, there were some JSSP instances in which the EVQE algorithm seemed incapable of exiting the state in which its individuals are initialised $|\psi_i\rangle = |0\rangle^{\otimes n}$. One problem instance for which this occurred is the JSSP instance with 2 jobs and 3 machines shown in Figure 6.1.

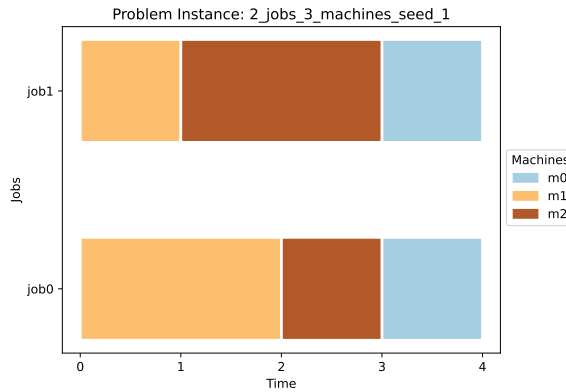


Figure 6.1.: This figure shows a problematic JSSP problem instance with 2 jobs and 3 machines.

Running the EVQE algorithm to solve that problem instance, it was visible, that the EVQE algorithm struggled to find better states than $|0\rangle^{\otimes n}$. As it can be seen in Figure 6.2a, this is evidenced by the fact that, during all generations, the expectation value of the best individual only worsened. Further evidence is that, in the end, the best-found individual is still mostly in the $|0\rangle^{\otimes n}$ state. This can be seen in Figure 6.2b, which showcases the distribution of measured basis-states when measuring the best individual’s ansatz.

As a result, for this problem instance, we theorise that the state $|0\rangle^{\otimes n}$ is a local minimum that is hard to leave for the EVQE algorithm. This seems logical, as in the zero state, each individual domain wall variable contains only zero bits. That makes it a valid state for all domain wall variables since it contains only one domain wall per variable: $1|0\dots 00$. As a result, the zero state minimises the encoding penalty term of the Hamiltonian. Furthermore, as explained in Section 5.2, for each operation, we assign the earliest possible start time to

6. EVQE - Issues and Improvements

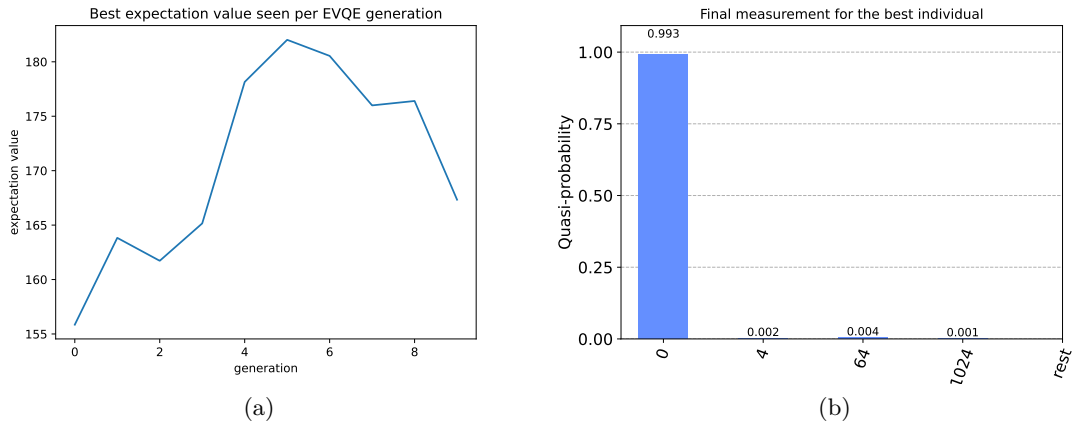


Figure 6.2.: This figure demonstrates the issues of the EVQE algorithm during an optimisation run for the problem instance shown in Figure 6.1. (a) shows the expectation value of the best individual for each generation. (b) shows the measurement result for the best individual found during the evolution. As it can be seen, EVQE was incapable of finding better states than the $|0\rangle^{\otimes n}$ state.

the domain wall state consisting of only zeros. This means that the zero state also minimises the optimisation term and the early start term of the Hamiltonian, as in the zero state all operations are scheduled to start as early as possible. From this local minimum, finding a state with a lower energy would involve rescheduling operations to start later, which worsens the optimisation term, without violating the encoding validity. Depending on the number of operations that need to be rescheduled at the same time to leave the local minimum, this might be a harder or easier task, which explains why this issue is not encountered for all problem instances.

To quantify how hard it is to leave the local minimum for this problem instance, we look at how many bits need to be flipped to 1 at the same time to leave the local minimum and reach a state with a lower energy. In Table 6.3, for the problem instance from Figure 6.1, we list all states with a lower energy than the zero state, for which no more than five bits from the zero state need to be flipped. We obtained these values by classically computing the energy for each possible state and then filtering them accordingly. As can be seen, the lowest number of bits that need to be flipped at the same time to reach a better state from the zero state is 4. We believe that this step might be too complex for the classical optimiser to reliably make with the very simple ansatz circuits of newly initialised EVQE individuals.

As a countermeasure, we propose two different initialisation schemes for new individuals in the EVQE algorithm. Both of these try to initialise the individuals outside the zero state to avoid this hard local minimum in the JSSP Hamiltonian.

The first proposal is to initialise the individuals with $n \geq 1$ gene instances, whose parameter values are chosen uniformly at random from the range $[0, 2\pi]$. This ensures that the individuals start out in a state other than the zero state. Furthermore, since the parameters of each individual are randomly chosen, each individual represents a different initial state. This enables the exploration of a range of initial states, with better initial states being more likely to continue on into later generations.

$ \psi\rangle$	$\langle\psi H \psi\rangle$	bit flips needed
$ 000000000000\rangle$	158.3	-
\vdots	160.4 – 8914.6	1 - 3
$ 010000010101\rangle$	33.3	4
$ 010101010000\rangle$	33.3	4
$ 010000110101\rangle$	60.4	5
$ 010100010101\rangle$	35.4	5
$ 010101010100\rangle$	35.4	5
$ 010101110000\rangle$	60.4	5
$ 110000010101\rangle$	60.4	5
$ 110101010000\rangle$	60.4	5

Figure 6.3.: This table shows eigenstates with their eigenvalues (energy) for the Hamiltonian of the JSSP instance shown in Figure 6.1. Specifically, these are eigenstates with energies lower than the energy of the zero state $|0\rangle^{\otimes n}$, which need the fewest bit flips to be reached from the zero state. For reference, it also contains the energy of the zero state and the range of energies for all states, which only need up to three bitflips.

The second proposal is to always prepend the ansatz of each individual with a fixed operator O_{init} , specified by the user. In that case, the state given by any individual is modified to:

$$|\psi_{g_i}\rangle = U(\gamma_k^i) \cdot \dots \cdot U(\gamma_0^i) \cdot O_{init} \cdot |0\rangle^{\otimes N} \quad (6.1)$$

This approach seems less flexible than the first proposal, but it might be worth it if there is a useful way to determine good initial states. This could, for instance, be the case if there is prior knowledge of some aspects of possible solutions. A natural seeming choice for O_{init} would be one Hadamard gate per qubit, so that the initial state for each individual is the superposition over all possible states $|+\rangle^{\otimes n}$.

Further preliminary tests showed that the first approach often converges faster but can sometimes get stuck back in the zero state local minimum. This seems to happen if one of the individuals starts out too close to the zero state local minimum and the other individuals are less fit, leading the individual in the zero state to dominate the population. This can be alleviated by initialising the individuals using more than one gene instance with random parameters per individual, which reduces the chance that any individual starts too close to the zero state. For the second approach, while it converges slower, it seems less likely to fall back into the local minimum. We theorise that the difference in convergence speed comes down to the first method being able to explore a wide range of initial states. A visualisation of the different convergence speeds on the JSSP problem instance from Figure 6.1 is shown in Figure 6.4. It shows that using the initialisation with two random gene instances and random parameters, EVQE finds an expectation value below $w_{opt} = 100$ after two generations. Prepending the individuals with Hadamard gates instead slows the convergence, so that EVQE needs 6 generations to find an expectation value below 100. For the benchmarking of the EVQE algorithm, we chose the first initialisation scheme with two random gene instances per individual due to its fast convergence.

6. EVQE - Issues and Improvements

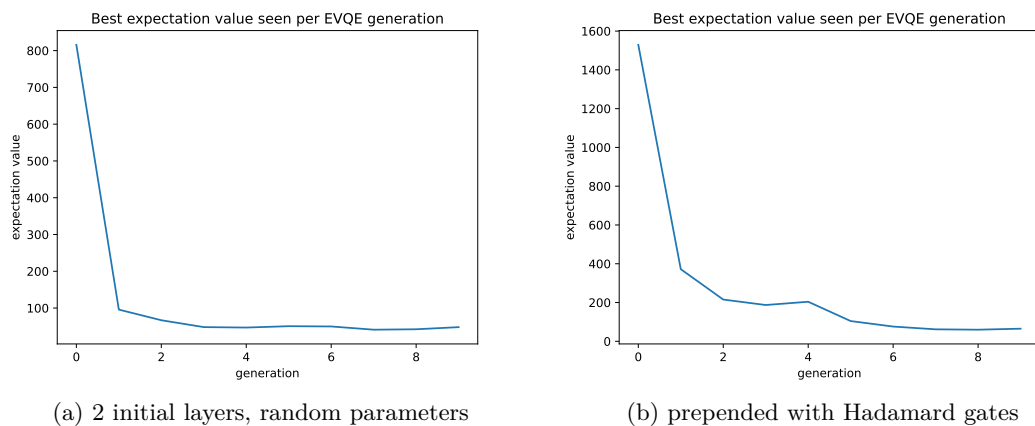


Figure 6.4.: This figure shows two EVQE optimisation runs for the JSSP instance shown in Figure 6.1. (a) uses an initialisation approach with two random gene instances and random parameters for each initial individual. (b) uses an initialisation approach by prepending Hadamard gates to each individual’s ansatz. (a) converges faster, presumably due to its ability to explore a range of good initial states. Note that the y-axis scales are different. (a) finds a state with an expectation value below 100 in generation 2, whereas (b) only finds such a state in generation 6.

6.2. Selection Pressure

Another issue that we encounter is that further along during the EVQE optimisation process, when most individuals are already relatively fit, the fittest individual in the population can sometimes be lost in the selection process if it is not selected as a parent for the next generation. This can revert the progress of one to a few generations and delay convergence.

This is possibly an issue of selecting individuals with likelihoods proportional to their fitness, as used by Rattew et al. Such selection methods are also called proportional selection. In particular, it is a known issue of proportional selection that there is a large selection pressure if there are large differences between fitness values, but only a small selection pressure if the fitness values do not differ very much [BOM15]. Selection pressure is the degree to which fitter individuals are more likely to be selected over less fit individuals in the selection process [BD95]. This aspect of proportional selection can be an issue for later generations of the EVQE algorithm, in which much of the population is already reasonably good, but further small improvements could still be found. To exemplify the issue, in Table 6.5, we provide the fitness values for a fictional population of five individuals, where the fitness values do not differ greatly. Like in EVQE, lower fitness values are better. The table also contains the selection probabilities calculated as outlined in Section 4.3.3. In particular, the fittest individual is not much more likely to be selected than the second-fittest individual, showcasing the issue of a low selection pressure caused by proportional selection.

If these small differences in selection likelihood are an issue, ordinal selection can be used instead of proportional selection. In ordinal selection schemes, instead of the absolute fitness value of an individual, its ranking in the population is the basis for its selection likelihood. As

Fitness	Selection Probability
30	24.6%
32	23.0%
40	18.4%
42	17.6%
45	16.4%

Figure 6.5.: This table shows selection probabilities for a population of five fictional individuals when using proportional selection as explained in Section 4.3.3. In particular, it shows the issue of low selection pressure if the fitness values do not differ greatly. See, for instance, the individual with fitness 30, which is only slightly likelier to be selected as a parent than the individual with fitness 32.

a result, such ordinal selection schemes are not impacted by the fitness values only differing by small amounts [BOM15]. A commonly used ordinal selection scheme is tournament selection. In tournament selection, for each parent to be selected, a tournament is held. In each tournament, k individuals are drawn from the population uniformly at random, with or without replacement. The fittest of the individuals in the tournament is then selected as a parent. If the individuals are selected without replacement, each individual can appear at most once in each tournament. In that case, the $k - 1$ least fit individuals can never be selected, as the k th individual in the tournament will always be better. If individuals are selected for the tournaments with replacement, even the least fit individual can win the tournament if it is selected k times in the same tournament and thus does not have to face a better competitor. A visualisation of the tournament selection process with a tournament size of $k = 2$ is shown in Figure 6.6.

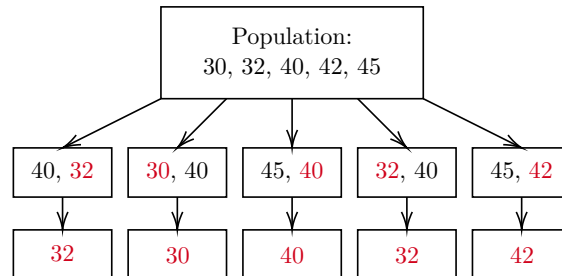


Figure 6.6.: This figure illustrates the tournament selection process, with a tournament size of $k = 2$, where five parents are selected from a population of five individuals. The numbers represent the fitness values of the individuals. Five tournaments are held to select five parents. The fittest individual for each tournament that is selected as a parent, is highlighted in red.

The tournament size k is a hyperparameter, which specifies the selection pressure. For a tournament size of 1, parents are selected uniformly at random, whereas increasing tournament sizes result in an increasing likelihood of selecting the fittest individual. A commonly chosen tournament size is $k = 2$. In Table 6.7, we show the selection probabilities for the same fictional population as in Table 6.5 when using tournament selection with replacement and a tournament size of $k = 2$.

6. EVQE - Issues and Improvements

Fitness	Selection Probability
30	36%
32	28%
40	20%
42	12%
45	4%

Figure 6.7.: This table shows selection probabilities for a population of five fictional individuals when using tournament selection with replacement and a tournament size of $k = 2$. When compared to the selection likelihoods for the proportional selection shown in Table 6.5, one can see that for small differences in fitness values, tournament selection still applies a reasonable selection pressure.

To prevent selection issues when fitnesses start to equalise, we add tournament selection, in which individuals are drawn with replacement, as an alternative to the proportional selection as proposed by Rattew et al. With that, both the choice of whether tournament selection should be used and the tournament size, are added to the list of parameters of our implementation of the EVQE algorithm.

7. Methodology

In this chapter, the methodology for benchmarking the EVQE algorithm is explained. To that end, the goals of this thesis are reiterated in Section 7.1. After that, the performance metrics used to evaluate the algorithm’s performance are explained in Section 7.2. Then, in Section 7.3, it is explained how we generate random problem instances to benchmark the optimisation algorithms on. After that, we discuss the choices made when configuring the algorithms for the benchmarks in Section 7.4. Following that, in Section 7.5, we demonstrate how we use automatic algorithm configuration to set all hyperparameters that were not fixed by our manual choices. Finally, with the benchmarking objectives, performance metrics, problem instances, and algorithm configuration explained, we outline our benchmarking procedure in Section 7.6.

7.1. Objectives

To investigate how the performance of Evolving Ansatz VQE approaches scales with increasing problem sizes, we investigate the scaling of the EVQE algorithm while comparing it with the more commonly used VQE and QAOA. To fulfil this goal, we need to run each of these algorithms multiple times on problem instances of increasing size to gauge the average performance of each algorithm on each problem size. This data can then be used to make educated inferences on the scaling behaviour of the investigated algorithms. To achieve this, we need:

- well-defined performance metrics;
- a set of problem instances of varying sizes;
- well-chosen hyperparameters for each algorithm;
- a rigorous benchmarking procedure.

Each of these items will be explained in more detail in the following sections of this chapter.

7.2. Performance Metrics

For the evaluation of optimisation heuristics and meta-heuristics, the performance metrics of importance can be grouped into three classes. The first of these classes concerns the quality of the solutions the heuristic finds, the second concerns the computational effort needed to obtain these solutions, and the third concerns the robustness of the heuristic to variations in the problem instances [BGK⁺95] [Tal09]. There are many ways to quantify these performance metrics [Tal09], each with its own upsides and drawbacks, making the choice of specific performance metrics difficult. In the following, we will explain our choice of performance metrics for the computational effort, the solution quality, and the heuristic’s robustness.

Computational Effort

When empirically evaluating the computational effort needed to find the solution for an optimisation problem, the computation time needed by the heuristic is a commonly used performance metric. This has the drawback of being dependent on the computing hardware used to run the heuristic [Tal09]. Another drawback in the context of evaluating VQAs using a classical simulator is that the runtime depends strongly on the runtime of the simulation of the quantum system. The issue is that the simulation of a quantum system generally has a runtime exponential in the system’s size [Hom22]. Therefore, measuring the runtime of VQAs over increasing problem sizes when using quantum simulators would show an exponential increase in runtime due to the scaling of the quantum simulations. This would misrepresent the quantum algorithms, as this effect would not appear when using real quantum hardware.

Measures for the computational effort independent of the heuristic’s runtime also exist. Remember that, as explained in Section 2.4, optimisation algorithms iteratively create parameter vectors x as educated guesses that try to minimise the objective function $f(x)$. For each guess, the optimisation algorithm then needs to evaluate the objective function for the parameter vector x to retrieve the quality of that guess. As a result, one runtime-independent metric of computational effort is the number of times the optimisation heuristic has to evaluate the objective function. According to Talbi, this is an acceptable measure of computational effort if the effort for evaluating the objective function f is high and does not depend on the parameter vector x that is evaluated [Tal09].

In VQAs, the objective function to be evaluated is the expectation value $\langle \psi(\theta) | H | \psi(\theta) \rangle$ for the parameter vector θ and the Hamiltonian H . This is done by measuring the ansatz circuit repeatedly to approximate the expectation value. In particular, the measurement effort to approximate the expectation value of the quantum state with respect to the problem Hamiltonian scales polynomially with the size of the problem Hamiltonian [TCC⁺22]. This means that evaluating the expectation value is expensive, so an effective VQA should query the expectation value as few times as possible for a good runtime [BCLK⁺22]. As the effort to approximate the expectation value scales with the Hamiltonian’s size, but an optimisation run is done on a single, fixed Hamiltonian, the cost of evaluating the expectation value during the optimisation procedure should not change depending on the parameter vector θ . Since the expectation value evaluations are expensive but do not depend on the specific parameter vector, the number of expectation value evaluations needed by a VQA is an acceptable metric for its computational effort, according to the criteria outlined by Talbi [Tal09]. As a result, we will use the number of needed expectation value evaluations as our metric for the computational effort.

With the amount of expectation value evaluations fixed as the metric for computational effort, the question remains until what point in the VQA optimisation procedure the expectation evaluations should be counted. To gauge the full computational cost of a VQA run, one has to count all expectation value evaluations until the termination of the VQA algorithm. We will refer to that count of expectation value evaluations until termination as $NEXP_{term}$. Yet, depending on the termination criterion, it can take some time to detect that a VQA optimisation run no longer improves. In that case, the best-found solution was already observed some time before the termination. Since we want to benchmark how fast the VQAs find good solutions and not how well-chosen our termination criteria are, it seems reasonable to only count the expectation value evaluations until the best solution found by the VQA is reached. We will refer to that count of expectation evaluations as $NEXP_{best}$.

Another point of interest is how quickly the VQAs progress within the optimisation process. This can be quantified by observing, after how many expectation value evaluations, the VQAs find expectation values below certain boundary values b .

Remember that measuring a quantum state with respect to an observable, like the Hamiltonian, yields some eigenvalue and an accompanying eigenstate, with the likelihoods depending on the amplitudes of the quantum state. Over multiple such measurements, the expectation value is then the weighted average of the observed eigenvalues weighed by their measurement likelihood. Remember also that for an Ising Hamiltonian encoding a QUBO, the eigenstates encode possible solutions, and the eigenvalues represent the energy of these solutions. Therefore, the expectation value with respect to this Hamiltonian can be interpreted as the weighted average over the energies for the solutions in the quantum state's superposition. Since the weights in that weighted average are the measurement likelihoods, they must be positive. As a consequence, if an expectation value E is measured for a quantum state, there must be solutions with energy $e \leq E$ in the quantum state [Juk11]. In other words, if the VQA, at any time, reports an expectation value, it must have found solutions with at most that energy. This enables the choice of the following boundary values.

The first interesting boundary value is the lowest penalty weight employed in the Hamiltonian encoding, as explained in Chapter 5. Since the Hamiltonian is encoded in such a way that each invalid solution has at least the energy of the corresponding penalty weight, each solution with an energy lower than the lowest penalty weight must be valid. As a consequence, once the expectation value for the quantum state falls below that boundary, that quantum state must contain a valid solution with a non-zero measurement probability. We refer to that boundary value as E_{bval} , and the number of expectation value evaluations needed to fall below that boundary as $NEXP_{val}$. The second interesting boundary value is the lowest energy of any solution with a suboptimal makespan. This is due to the fact that the Hamiltonian encoding was designed so that solutions with a higher makespan should always have a higher energy than solutions with a lower makespan. As a consequence, if one calculates the lowest energy for a solution with a suboptimal makespan, all solutions with a lower energy must be solutions with an optimal makespan. Therefore, once the expectation value for the quantum state falls below that boundary value, that quantum state must contain an optimal solution with a non-zero measurement probability. We refer to that boundary value as E_{bopt} and the number of expectation value evaluations needed to fall below that boundary as $NEXP_{opt}$. Note that the boundary value E_{bval} is easily known, as it follows from the penalty weights set by the user in the Hamiltonian encoding. The boundary value E_{bopt} , on the other hand, is not normally known, as finding it involves calculating the energies for all solutions to a problem instance, which effectively solves the instance to optimality. We only calculate it to evaluate the optimisation progress of the VQA algorithms. Figure 7.1 now visualises the concepts discussed so far using an example optimisation run of EVQE on a JSSP problem instance needing 21 qubits.

Solution Quality

As explained in Section 2.3, after a VQA finishes its optimisation routine, its quantum system is measured one final time, yielding a probability distribution of states, each of which is a possible solution to the problem to be solved. We ensure that this final measurement is done using the best parameter values observed during the whole VQA routine. To gauge the quality of the final state yielded by the VQA, we can then look at the measurement likelihoods

7. Methodology

of good solutions. In particular, we will distinguish the total measurement likelihood P_{val} for all states that are valid solutions to the JSSP instance and the total measurement likelihood P_{opt} for all optimal solutions to the JSSP instance. The latter are solutions whose makespan matches the minimum makespan of the JSSP instance. Note that optimal solutions are a subset of valid solutions. Therefore, it always holds that $P_{opt} \leq P_{val}$.

Heuristic Robustness

Since there is no consensus on how to define the robustness of a heuristic [Tal09], we will not attempt to define a specific metric here. Instead, for our benchmarks over multiple problem instances, we report if there are any large inconsistencies in the performance of a VQA between problem instances.

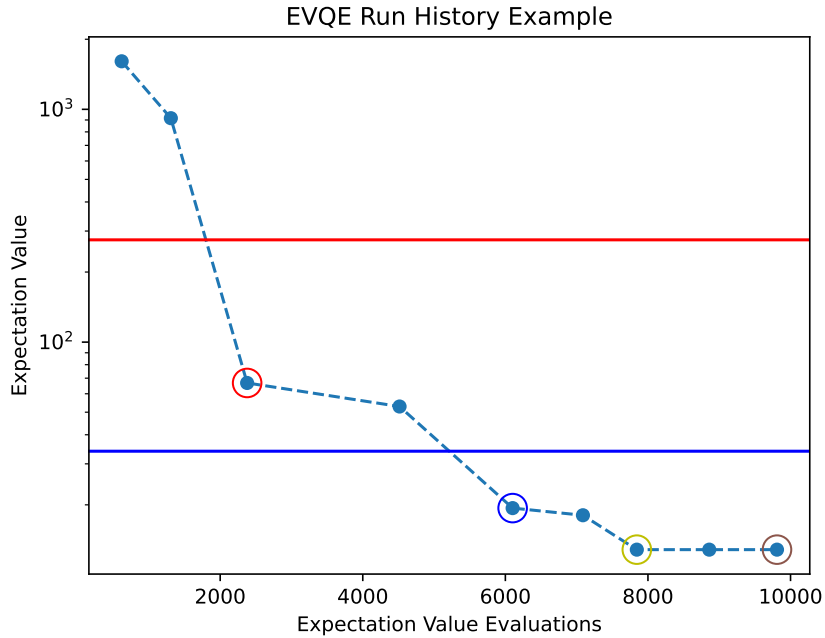


Figure 7.1.: This figure shows the optimisation history of an example run of the EVQE algorithm. Each point in the plot shows the best expectation value found during a generation of the evolutionary algorithm and the total number of expectation value evaluations used at the end of that generation. The red horizontal line shows the energy boundary for valid states E_{bval} , while the blue horizontal line shows the energy boundary for optimal states E_{bopt} . As can be seen in the graph, EVQE first provably found valid solutions in the third generation (highlighted in red), first provably found optimal solutions in the fifth generation (highlighted in blue), found its best expectation value in the 7th generation (highlighted in yellow), and terminated after 9 generations (highlighted in brown). This amounts to the following amounts of expectation value evaluations: $NEXP_{val} = 2387$, $NEXP_{opt} = 6102$, $NEXP_{best} = 7844$, $NEXP_{term} = 9810$

7.3. Random Problem Instance Generation

In this section, it is explained how a benchmarking dataset of random JSSP problem instances is generated. To that end, we first explain how to generate a JSSP instance of a specific size in terms of the number of jobs and machines from a random seed. Then it is explained which workflow and filtering criteria are applied to generate and select relevant problem instances.

7.3.1. Generating a Single JSSP Instance

In previous work by Taillard et al., it has been outlined how random JSSP instances can be generated, where in each job each operation is exactly assigned one machine [Tai93]. Taillard et al.’s approach consists of two main parts. The first is that each operation of a job is randomly assigned a processing duration from a predetermined range. The second is the random assignment of a unique machine to each operation of a job. Note that it is assumed that each operation of a job needs to be assigned exactly one unique machine from the set of available machines and that there are exactly as many operations per job as available machines. As a result, the random machine assignment can be understood as a random ordering of the available machines for each job.

Our approach to the random generation of JSSP instances follows the general idea of the approach described above, with two main changes. First, instead of assigning the processing duration uniformly at random from the wide range $[1, 99]$ as proposed by Taillard et al., we assign the processing duration p with the probability distribution P shown in Equation 7.1. This is necessitated by the fact that we need compact problem instances so that encoding them as a Hamiltonian does not need overtly many qubits. We still allow processing durations of 2 with a limited likelihood to prevent the problem instances from being entirely uninteresting.

$$P(p) = \begin{cases} 0.75 & \text{if } p = 1 \\ 0.25 & \text{if } p = 2 \end{cases} \quad (7.1)$$

For assigning machines to the operations of a job, we use Python’s `random.shuffle` method to generate a random machine assignment.

The whole generation process, as described above, in our implementation is seeded by one user-specified integer value. This makes the process deterministic for a given seed and a given number of jobs and machines. This allows the repeated retrieval of the same problem instance if the same seed, same number of jobs, and same number of machines are specified.

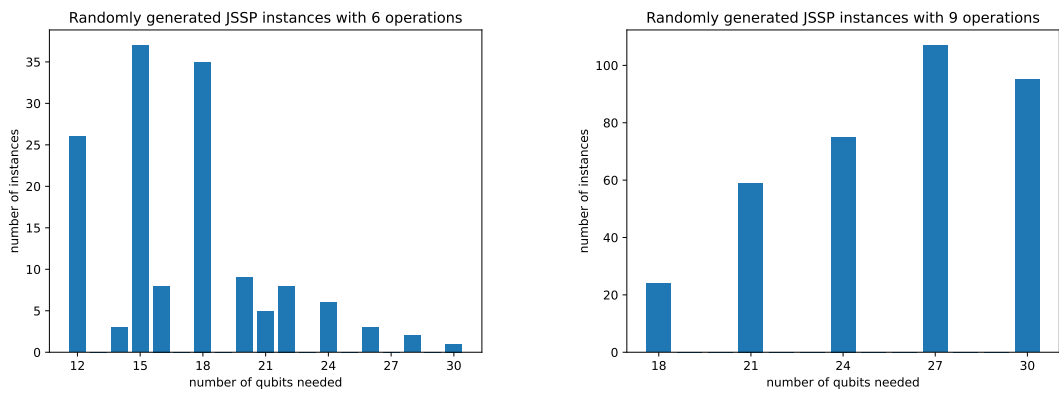
7.3.2. Generating Datasets of JSSP Instances

In the following, the size (x, y) will refer to JSSP instances with x jobs and y machines. To generate datasets of JSSP instances, we randomly generate 1000 JSSP instances with the seeds $s \in \{0, \dots, 999\}$ for each problem size. In particular, we generate instances for the sizes (2,3), (3,2), (3,3), (3,4), and (4,3). For each problem instance, we use the classical SCIP solver [BBC⁺23] to find its optimal makespan. This then enables us to filter the JSSP instances by whether they are interesting and by whether they can be encoded into a Hamiltonian with a manageable amount of qubits.

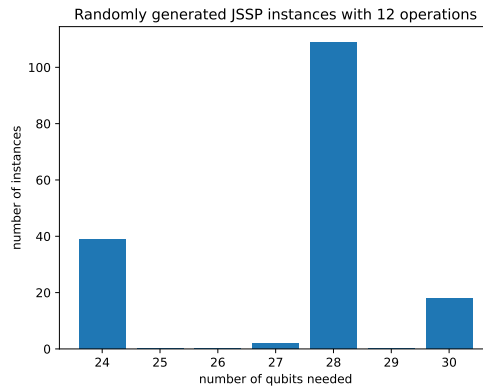
7. Methodology

The first filter criterion we check for is that the optimal makespan of a problem instance is longer than the length of any of its jobs. This ensures that in the problem instance, at least one operation has to be scheduled to start later than its earliest start time to find a valid solution. If this is not the case, all operations in the problem instance can start at their earliest possible start time, making the problem instance trivial to solve and, as such, uninteresting. For the second filter criterion, we create the problem Hamiltonian for the JSSP instance with a makespan limit that is the optimal makespan plus one. We consider a JSSP instance permissible if, using that makespan limit, its Hamiltonian requires at most 30 qubits. Finally, we try to filter out invariants of already-seen JSSP instances, which encode the same problem but with different job or machine names attached.

Following this procedure and grouping the instances first by their number of operations and then by the number of qubits they use, we gain the following number of JSSP instances



(a) Number of JSSP instances with six operations (b) Number of JSSP instances with nine operations



(c) Number of JSSP instances with twelve operations

Figure 7.2.: These figures show the number of randomly generated problem instances, which satisfy our filtering criteria, distributed over the number of qubits that they require: (a) shows the number of problem instances with 6 operations; (b) shows the number of problem instances with 9 operations; and (c) shows the number of problem instances with 12 operations.

in our datasets: Figure 7.2a shows the distribution of JSSP instances with six operations over the number of qubits they use. Figures 7.2b and 7.2c show the same distribution for the nine and twelve operation instances. As can be seen in these figures, problem instances of different sizes cover different numbers of needed qubits. To enable our scaling investigation to cover problems of increasing complexity, we will use JSSP instances with six operations for the qubit numbers 12 and 15, JSSP instances with nine operations for the qubit numbers 18 and 21, and JSSP instances with twelve operations for all numbers of qubits larger than that.

7.4. Manual Algorithm Configuration Decisions

VQAs consist of many parts, for each of which decisions and hyperparameters need to be set to dictate their exact behaviour. While we let many of these hyperparameters be set by automatic algorithm configuration, for some we made deliberate design choices beforehand. This section outlines the most important of these choices and explains their reasoning.

Choice of Classical Optimiser

One central choice, which affects the behaviour of variational quantum algorithms, is the choice of the classical optimisation routine employed to optimise the parameter values of the ansatz. Some comparative studies on classical optimisers in the context of VQAs have been done [BMWV⁺23] [Loc22] [MFP⁺22] [PJSPP21] [SMM23]. They generally found that the choice of the classical optimiser both strongly impacts the performance of the VQAs and their resistance to quantum noise. Two strong candidates emerging from these comparisons are the COBYLA and the SPSA optimisers. COBYLA outperforms SPSA in noise-free contexts, whereas SPSA tends to be much more noise-resistant than COBYLA, leading it to perform better in noisy contexts [PJSPP21] [SMM23]. Even though Rattew et al. employ COBYLA in their paper as the optimiser of choice for the EVQE algorithm, we choose to use the SPSA optimiser. This is due to the fact that we opt to use the same classical optimisation algorithm for EVQE, VQE, and QAOA, so that performance differences do not come down to differences in the classical optimisation method. Employing COBYLA could then mean that in noisy benchmarks, VQE and QAOA would be handicapped due to COBYLA’s susceptibility to noise. The implementation of SPSA we use stems from the Python package `qiskit_algorithms` of the `Qiskit` framework [Qis23].

Termination Behaviour

To decide when to terminate the optimisation process, we employ two limits. The first is a bound on the number of total expectation value evaluations the VQA may make. We set these to high values to act as a contingency to prevent the optimisation algorithms from running indefinitely. The second limit is a termination criterion, which involves the change in the expectation value between subsequent optimisation iterations. In the case of QAOA and VQE, this is the change in expectation value between subsequent SPSA iterations. For EVQE, this is the change in the best observed expectation value between subsequent generations. Specifically, we want to terminate the optimisation process once the expectation values over multiple iterations are stable, as this seems to imply that the VQA is stuck in a local or global optimum and will not go on to find better expectation values. To detect that,

7. Methodology

we look at the magnitude of the change in the expectation value between iterations relative to the magnitude of the expectation values themselves. Let the best expectation value at iteration i of the optimisation be denoted as E_i , then the relative change in expectation value at iteration i , ΔE_i can be written as:

$$\Delta E_i = \frac{|E_i - E_{i-1}|}{|E_{i-1}|} \quad (7.2)$$

We configure QAOA and VQE to terminate once ΔE_i falls below 0.01 for ten consecutive iterations. Meanwhile, we configure EVQE to terminate once ΔE_i falls below 0.01 for two consecutive iterations. We chose this less strict termination criterion for EVQE because its iterations (generations) are much more expensive in terms of expectation value evaluations, and we want to prevent it from running too long after it has already converged.

Target Function

As explained in Section 2.3, normally in VQAs, the value to be minimised by optimising the parameter values θ is the expectation value $\langle \psi(\theta) | H | \psi(\theta) \rangle$. It is the expectation over measured eigenvalues when repeatedly measuring the state $|\psi(\theta)\rangle$ with respect to the Hamiltonian. Yet another measure, the so-called Critical Value at Risk (CVaR), can be used as well [BNR⁺20]. It is motivated by the fact that if, in the final state produced by the VQA, a good solution can be measured with a reasonable likelihood, the measurement likelihood of other worse solutions does not matter greatly. As a result, the idea of the CVaR measure is to derive the expectation value not over all measured eigenvalues but only over the best quantile α of the measured eigenvalues. To be exact, the CVaR is defined as shown in Equation 7.3, where X is a random variable, F_X is its cumulative density function, and $\alpha \in (0, 1]$ denotes the quantile.

$$CVaR_\alpha(X) = \mathbb{E} [X | X \leq F_X^{-1}(\alpha)] \quad (7.3)$$

Using the CVaR has been shown to be beneficial for QAOA and VQE [BNR⁺20] [ARF⁺22]. This seemed to hold true in our preliminary tests. We also found using the CVaR measure to be beneficial to EVQE as it helped reduce the occurrence of premature convergences to local minima. We theorise that this is due to the fact that the CVaR value enables the classical optimiser to explore worse parts of the search space to some degree without being penalised, which helps it escape from local minima.

Due to the benefits offered by the CVaR measure, we opt to use the CVaR measure in all benchmarked VQA algorithms. Specifically, we choose to set α to 0.5, as Amaro et al. [ARF⁺22] have done in their comparative benchmark of QAOA and VQE against F-VQE. This means that only the best 50% of the measured eigenvalues are part of the expectation value. As a result, we expect the VQAs to find final states with the measurement likelihood of good solutions bounded from above at 50%, as the VQAs are not rewarded to increase the measurement likelihood above 50%.

QAOA, VQE: Implementation and Ansatz

For QAOA and VQE, we use implementations from `qiskit_algorithms` [Qis23]. In the QAOA implementation, given a user-specified number of desired layer repetitions, the ansatz is automatically generated based on the Hamiltonian at hand. For the small JSSP instance

from Figure 5.1, the accompanying QAOA ansatz with one layer can be seen in Figure A.1 in Appendix A. With the VQE implementation, the user has to specify an ansatz layer structure, and the number of layer repetitions. For the layer structure, we use the same structure as Amaro et al. [ARF⁺22] (see Figure 3.1 in Chapter 3), as they have shown that this layer structure works reasonably well for solving the JSSP with VQE. For both algorithms, we defer the choice of the number of ansatz layers to the automatic algorithm configuration.

7.5. Automatic Algorithm Configuration

Both the classical optimiser employed in VQAs and the VQAs themselves offer many hyperparameters, which need to be well-chosen to enable the VQAs to work well. A further source of hyperparameters are the weights used to encode the problem Hamiltonian. These can alter the structure of the search space traversed by the VQAs and thus ease or harden the optimisation task. In particular, it might even be the case that different search space structures might be more or less suited for different VQAs. As a result, for each VQA we investigate (i.e., QAOA, VQE, and EVQE), we optimise the hyperparameters of the classical optimiser, the quantum heuristic, and the Hamiltonian together to improve its performance. To ensure that training the hyperparameters does not overfit on any specific problem instance, which would hamper the usefulness of the VQA, we train the hyperparameters over a set of training problem instances. This task in the research literature is also called algorithm configuration. In particular, given an algorithm A with hyperparameters, a collection of problem instances I , and a cost function c , the task of algorithm configuration is finding values for the hyperparameters that minimise c over the problem instances in I [HHLB11]. To explain our hyperparameter optimisation methodology, we first explain our choice of the SMAC3 [MKM⁺22] algorithm configuration software. We then explain the hyperparameters that we need to optimise. Finally, we explain how we configure SMAC3 to find good hyperparameters.

7.5.1. SMAC3

The task of algorithm configuration involves running the VQAs using varying hyperparameter configurations repeatedly over multiple problem instances to gauge their performance. Since each optimisation run of a VQA is expensive in runtime due to the computational effort involved in quantum simulations, we use the algorithm configuration library SMAC3 [MKM⁺22], which offers multiple advantages for configuring algorithms with high runtimes. In particular, SMAC3 is very economical in the number of times it needs to run the VQA to find good configurations. One reason for this is that SMAC3 employs Bayesian optimisation. This means that during the optimisation process, it models the search space based on already-observed configurations. It uses this model to gauge which regions of the search space are particularly promising to sample from for improved configurations, which reduces the number of needed samples to find a good configuration. A further benefit of SMAC3 is that it uses racing. This means that it discards bad configurations early and only runs configurations repeatedly to evaluate them more accurately if they are promising candidates. Finally, since SMAC3 was specifically developed for algorithm configuration, it supports optimisation over multiple problem instances and multi-target cost functions. The latter fact allows us to optimise for configurations that both maximise the quality of the

final state distribution and, at the same time, minimise the number of needed expectation value evaluations.

7.5.2. Hyperparameters

Since evaluating configurations is expensive in runtime, we use educated guesses to restrict the value ranges for each hyperparameter from which SMAC3 may choose as far as possible. This narrows the space of possible configurations and should help SMAC3 find good configurations faster. In the following, we outline the hyperparameters offered by the Hamiltonian encoding process, the SPSA optimiser, as well as the EVQE, VQE, and QAOA algorithms. For each hyperparameter, we mention its data type, explain its effect, and give a value range for SMAC3 to choose from. The data type is specified next to the hyperparameter name in square brackets. Since some hyperparameters only apply to some quantum algorithms, we colour-code the hyperparameters. Grey hyperparameters are hyperparameters that we do not optimise but instead choose a fixed value for. The black hyperparameters are optimised for each VQA. Finally, blue hyperparameters are only optimised for EVQE, while green hyperparameters are only optimised for QAOA and VQE.

Hamiltonian Weightings

The hyperparameters for the Hamiltonian encoding process are the weights used to set the relative importance of the individual terms within the Hamiltonian. These have been explained in detail in Chapter 5.

optimisation term weight [Float]

This weight w_{opt} describes the maximum energy a valid state may have. We set this value to 100.

encoding penalty weight [Float]

This weight w_{enc} ensures that all invalid encoding states have at least energy w_{enc} . We let SMAC3 choose it from the range [110, 1000]. The lower bound ensures that the valid states (up to energy 100) are always better than the invalid states. The upper bound of 1000 was chosen due to the fact that a penalty more than 10 times the size of w_{opt} seems excessive.

precedence constraint penalty weight [Float]

This weight w_{prc} ensures that all states that violate the precedence constraint have at least energy w_{prc} . With the same reasoning as for the encoding weight, we let SMAC3 choose it from the range [110, 1000]. We also constrain its value so that it can never be chosen to be higher than w_{enc} . This is done to prevent invalid interactions of the penalty terms.

overlap constraint penalty weight [Float]

This weight w_{ovl} ensures that all states which violate the overlap constraint have at least energy w_{ovl} . We also let SMAC3 choose it from the range [110, 1000]. We also constrain its value to never be higher than w_{enc} .

early start term importance [Float]

The optimisation term consists of the makespan and early start optimisation terms,

whose relative importance is weighted by a factor γ . Specifically, γ determines what portion of the optimisation term is made up of the early start term. We let SMAC3 choose this factor from the range $[0.1, 0.5]$. We define the upper bound to limit the early start term from becoming more important than the makespan term. The lower limit was chosen due to our belief that dropping the early start term entirely would be detrimental to the optimisation process.

SPSA

The implementation of SPSA from the Python package `qiskit_algorithms` [Qis23] offers many hyperparameters. We exclude hyperparameters that concern the approximation of second-order gradients, as second-order gradient approximation in SPSA does not seem to be commonly used in research on VQAs.

maxiter [Int]

This is the maximum number of times that SPSA approximates the gradient and updates the parameter values. SPSA may use less than `maxiter` iterations if a termination criterion is hit beforehand. For the usage of SPSA in EVQE, since it will be used many times as a subroutine, we let SMAC3 choose it from a relatively low range of $[10, 50]$. This is meant to prevent EVQE from becoming too costly, due to the overtly high number of SPSA iterations. For QAOA and VQE, since SPSA is run only once in their routine, we do not want to terminate prematurely based on `maxiter`. Therefore, we set `maxiter` high enough so that our termination criterion based on the number of expectation value evaluations will always trigger first.

perturbation [Float]

SPSA approximates the gradient by randomly perturbing the parameter values and measuring the resulting changes in the expectation value. This hyperparameter determines by how much the parameter values are perturbed. In Equation 2.46, it is designated as σ . We let SMAC3 choose the `perturbation` from the range $[0.01, 0.5]$, as previous research on SPSA in the context of VQAs has shown that it needs a `perturbation` and `learning_rate` larger than typical in other applications such as classical machine learning [Loc22].

learning_rate [Float]

Once SPSA has approximated a gradient, the `learning_rate` determines to what extent the parameters are updated in that direction. We also let SMAC3 choose it from the range $[0.01, 0.5]$.

resamplings [Int]

Normally, SPSA perturbs the parameter values in two opposite directions and makes two expectation value measurements to approximate the gradient. If a more accurate gradient approximation is wanted, this gradient approximation can be repeated multiple times, with their average yielding a more accurate gradient. This hyperparameter determines how often the gradient is approximated. We let SMAC3 choose it from the low range of $[1, 4]$ to prevent the individual SPSA iterations from becoming too costly in terms of the number of needed expectation value evaluations.

trust_region [Bool]

This is a boolean hyperparameter that, if enabled, prevents overtly large update steps. This is done by scaling the update step vector to size 1, if its size would otherwise exceed 1. We let SMAC3 choose it from the values $\{TRUE, FALSE\}$.

blocking [Bool]

This boolean hyperparameter determines whether the update step should be rejected if it worsens the expectation value too much. We let SMAC3 choose it from the values $\{TRUE, FALSE\}$.

allowed_increase [Float]

If blocking is enabled, this hyperparameter determines what increase in expectation value is considered as worsening the result too much. We let SMAC3 choose it in the range $[0, 500]$.

termination behaviour [Int]

The main termination criteria for EVQE, VQE, and QAOA were outlined in Section 7.4 and remain fixed during the algorithm configuration procedure. For VQE and QAOA, this termination criterion is the termination criterion for SPSA, due to SPSA being the main part of their optimisation routine. But for EVQE, this termination criterion does not concern SPSA directly, as SPSA is only a repeated subroutine of EVQE. As a result, there is still a need for a termination criterion for the SPSA optimiser within EVQE. We define this termination criterion just like for QAOA and VQE. But since in EVQE, it is not entirely clear how quickly the SPSA subroutine should terminate, we let SMAC3 choose how often ΔE_i must fall below 0.01 consecutively for SPSA to terminate. We let SMAC3 choose this value from the range $[2, 10]$.

last_avg [Int]

`qiskit_algorithm`'s implementation of SPSA normally returns the parameter values and expectation value seen just before termination, even if better ones were encountered before. The `last_avg` hyperparameter adjusts this behaviour so that the average of the last k parameter values and the expectation measured for these average parameters are returned. Since for QAOA and VQE we read out the best parameter values and expectation value seen during the entire SPSA optimisation run, `last_avg` does not impact them. For our implementation of EVQE, such a readout is not easily doable without making it less compatible with other classical optimisation algorithms. As a result, our implementation of EVQE uses the results as returned by `qiskit_algorithm`'s SPSA, meaning that the `last_avg` hyperparameter impacts EVQE. Therefore, we let SMAC3 choose `last_avg` for EVQE in the range $[1, 4]$.

EVQE

EVQE, as explained in Chapter 4, offers many hyperparameters to tweak its behaviour. Keep in mind that for the hyperparameter optimisation, we already include the improvements outlined in Chapter 6.

population size [Int]

The `population size` determines how many individuals are contained in EVQE's

population. Decreasing it reduces the number of expectation evaluations per generation, as fewer individuals are needed to be varied and evaluated. On the other hand, increasing it helps the evolutionary algorithm explore a larger area of the search space. We fix it at 10 for the hyperparameter optimisation process, as we do not want SMAC3 to reduce the `population size` too much when it tries to reduce the number of expectation value evaluations. Instead, the effect of the `population size` is investigated manually during the benchmarking.

genetic distance [Int]

The `genetic distance` roughly determines by how many genes individuals may differ and still be part of the same species. This value should not be set too large, so that dissimilar individuals cannot be considered to be of the same species. We let SMAC3 choose it from the range $[1, 3]$.

α penalty [Float]

The α penalty determines to what extent the fitness of an individual is adjusted based on the number of gene instances (circuit layers) in its genome. Increasing this penalty should lead to shallower ansatz circuits but make the search for good ansatz circuits more difficult. We allow SMAC3 to choose it from the range $[0, 0.4]$.

β penalty [Float]

The β penalty determines to what extent the fitness of an individual is adjusted based on the number of CU3 gates in its genome. Increasing this penalty should lead to fewer controlled gates in the ansatz but make the search for good ansatz circuits more difficult. We allow SMAC3 to choose it from the range $[0, 0.4]$.

likelihood of topological search [Float]

This hyperparameter determines the likelihood with which a gene instance (circuit layer) is added to an individual during the variation step of EVQE. To allow EVQE to explore sufficient amounts of ansatz circuits, we found, in preliminary testing, that this value should not be too small. We allow SMAC3 to choose this likelihood from the range $[0.4, 0.8]$.

likelihood of parameter search [Float]

This hyperparameter determines the likelihood with which all parameters for all circuit layers of an individual are optimised using repeated calls of the classical optimisation subroutine during EVQE's variation step. This variation operator is very costly in terms of expectation value evaluations due to its repeated calls of the classical optimisation subroutine. As a result, we limit its likelihood and let SMAC3 choose it from the range $[0, 0.5]$.

likelihood of layer removal [Float]

This hyperparameter determines the likelihood with which gene instances are removed from an individual's genome during EVQE's variation step. To prevent individuals from being pruned too often, which will hurt the search for good ansatz circuits, we let SMAC3 choose it from the limited range of $[0, 0.15]$.

tournament size [Int]

As discussed in Section 6.2, the tournament size modifies the selection likelihoods of

7. Methodology

individuals as parents for the next generation. Increasing the tournament size increases the likelihood of selecting fit individuals. To keep the population diverse, it should not be set too large. We let SMAC3 choose it from the two values $\{2, 3\}$.

QAOA / VQE

Apart from the hyperparameters offered by the Hamiltonian encoding process and the classical optimiser, for QAOA and VQE, there is only one additional hyperparameter to optimise.

ansatz layer repetitions [Int]

As explained in Section 2.3, the ansätze of QAOA and VQE consist of repeating ansatz layers. Increasing the number of ansatz layers increases the expressibility of the ansatz but may make them more difficult to optimise. For both QAOA and VQE, we let SMAC3 choose the number of the ansatz layer repetitions from the range $[2, 4]$.

7.5.3. Configuring SMAC3

To configure SMAC3 to optimise the hyperparameters for EVQE, VQE, and QAOA, we need to define what cost functions it should minimise, over which problem instances it should minimise them, and how it should run the VQA algorithms to evaluate configurations.

Cost Functions

SMAC3 needs cost functions to map optimisation runs of a VQA to scalars, which indicate the quality of the VQA run to SMAC3. These cost functions must be defined so that lower scalar values represent better VQA runs, as SMAC3 only supports minimisation.

For our purposes, there are two main goals for SMAC3 to optimise. The first is the likelihood of measuring good solutions to the JSSP in the final state measurement, and the second is the number of expectation value evaluations needed throughout the optimisation run. The number of expectation value evaluations is already a scalar value, where lower values are better. Therefore, it can be reported directly to SMAC3. The measurement likelihoods of good solutions are less straightforward to encode in a scalar cost function since higher likelihoods for good solutions are better and there are different solution qualities to be distinguished. In particular, one needs to discern the measurement likelihoods for suboptimal and optimal solutions, as we prefer solutions with an optimal makespan over solutions with suboptimal makespans. To deal with both facts, we define the 'inverse state quality' as starting from 100 and subtracting 1 for each percent of measurement likelihood for optimal solutions and 0.5 for each percent of measurement likelihood for suboptimal solutions. This yields a value that is 100 if no valid solution can be observed and 0 if the measurement likelihood for optimal solutions is 100 percent. Since with suboptimal solutions, no value lower than 50 can be reached, the inverse state quality also encourages SMAC3 to find configurations that tend towards finding optimal solutions. As an example, if the likelihood of measuring optimal solutions is 25% and the likelihood of measuring suboptimal solutions is 50%, the corresponding inverted state quality is $100 - (1 \cdot 25 + 0.5 \cdot 50) = 50$. In an equation, the inverse state quality can be defined as follows, where P_{opt} is the likelihood of measuring optimal solutions and P_{subopt} is the likelihood of measuring suboptimal solutions:

$$ISQ = 100 * (1 - (P_{opt} + 0.5 \cdot P_{subopt})) \quad (7.4)$$

Since EVQE grows its ansatz circuits during its optimisation procedure, there are two more cost functions that should be considered for EVQE. These are the number of layers in the ansatz circuit of the best individual and the number of CU3 gates in the ansatz circuit of the best individual. These should be considered cost functions, as otherwise SMAC3 might find configurations that unnecessarily encourage adding too many layers or controlled gates to the ansatz circuit. This would hamper EVQE’s applicability on real quantum hardware. As both are already scalar values, where lower values are better, both can be directly communicated to SMAC3 as is.

Problem Instances

As problem instances for SMAC3 to optimise over, we use 5 random JSSP problem instances, containing 6 operations, which need 12 qubits, from our pre-generated dataset of problem instances. Visualisations of these training instances can be found in Figure A.2 in Appendix A.

Run Configuration

We configure SMAC3 so that for each algorithm, it executes at most 2500 optimisation runs. For each of these optimisation runs, we configure the termination criteria of the VQA algorithms to terminate after at most 30,000 expectation value evaluations. This is done to prevent bad configurations from running indefinitely. Furthermore, for each hyperparameter configuration that SMAC3 evaluates, we configure it to run at least two optimisation runs and at most 20 optimisation runs. The purpose of this is that it should discard no configuration prematurely but also not spend overt effort on evaluating any configuration.

7.6. Benchmarking Procedure

Since EVQE needs many expectation value evaluations and free-of-charge access to quantum computing hardware is currently very limited, we run our benchmarks using the `qiskit_aer` quantum simulator of the `qiskit` quantum software framework [Qis23]. Unless otherwise specified, we use the noiseless statevector simulation method. It exactly calculates the amplitudes and, thus, the measurement probabilities for the quantum state $|\psi\rangle$ produced by a given ansatz. This then allows the exact computation of the expectation value with respect to the problem Hamiltonian. Due to the fact that these simulations are computationally intensive, we run them highly parallelised on the High Performance Data Analytics platform ‘terabyte’, which is jointly operated by the DLR (German Aerospace Center) and LRZ (Leibniz Supercomputing Center).

Before the benchmarks, we run the hyperparameter optimisation as described in the previous section for each of the VQAs. We then employ the best-found hyperparameter configuration for each algorithm in all the subsequent benchmarks.

From there, we distinguish benchmarks with two different goals. The first and main goal is to investigate and compare the scaling of computational effort and result quality of the VQAs over increasingly large problem sizes. The second goal is to compare how noise-resistant the VQAs are.

Finally, we obtain results for an EVQE optimisation run on real quantum hardware.

Scaling Benchmarks

To gauge the scaling of the VQAs, we run each VQA 25 times for each of the problem sizes of 12, 15, 18, and 21 qubits. We limit the number of expectation value evaluations the VQAs may take per optimisation run to 15,000 to prevent excessive runtimes on the larger problem instances. For each problem size, these optimisation runs are equally distributed over five problem instances of that size. The problem instances used for benchmarking can be seen in Appendix A in Figure A.3 for 12 qubits, Figure A.4 for 15 qubits, Figure A.5 for 18 qubits, and Figure A.6 for 21 qubits. The problem instances were drawn randomly from our pre-generated dataset of JSSP instances. Care was taken to avoid overlap with the training instances used for the hyperparameter optimisation. Finally, for each optimisation run, we store the history of expectation value evaluations and the final state measurement, which enables us to derive the metrics explained in Section 7.2.

We follow this procedure once for QAOA and VQE each. For EVQE, we follow this procedure three times with different population sizes. Once with a population size of 5, once with a population size of 10, and once with a population size of 20. This enables us to gain insight on how the population size affects the performance and scaling of the EVQE algorithm. Following this, we will refer to these different configurations of EVQE as EVQE5, EVQE10, and EVQE20. Where no number is specified, EVQE refers to EVQE10.

Noisy Benchmarks

So far, we have used statevector simulations, which are perfectly noiseless. For the noisy benchmarks, we introduce shot noise to the statevector simulation. Shot noise is based on the fact that the measurement of expectation values and state distributions for a quantum state is done via repeated probabilistic measurements of the quantum state (the so-called shots) [WCA⁺24]. For statevector simulations, shot noise can be approximated by sampling k shots using the measurement probabilities calculated by the simulator. The fewer shots are used, the less accurate the simulated measurement results may be. We run each VQA 25 times on the 18 qubit problem instances for each of 1024, 256, and 64 shots per measurement. These optimisation runs are again equally distributed over the same five problem instances as used in the scaling benchmarks. Furthermore, the number of expectation value evaluations is again capped at 15,000.

7.6.1. Real Quantum Hardware

As EVQE needs many more expectation value evaluations than is feasible with IBM’s free tier access to quantum hardware, we ourselves are not able to run EVQE on real quantum hardware. Thanks to E.ON Digital Technology, who have more extensive access to IBM quantum computing hardware and who were interested in running and evaluating our implementation of EVQE in QUEASARS, we were able to retrieve results on real quantum hardware.

Since this optimisation run is meant as a proof of concept on real quantum hardware, it is done for a small problem instance shown in Figure 7.3, which needs only 8 qubits. Furthermore, EVQE is configured with a population size of 5. It is also configured to terminate after three generations. The quantum circuits are run on IBM Quantum’s `ibm_nazca` 127 qubit quantum computer, with 512 shots per measurement and no error mitigation.

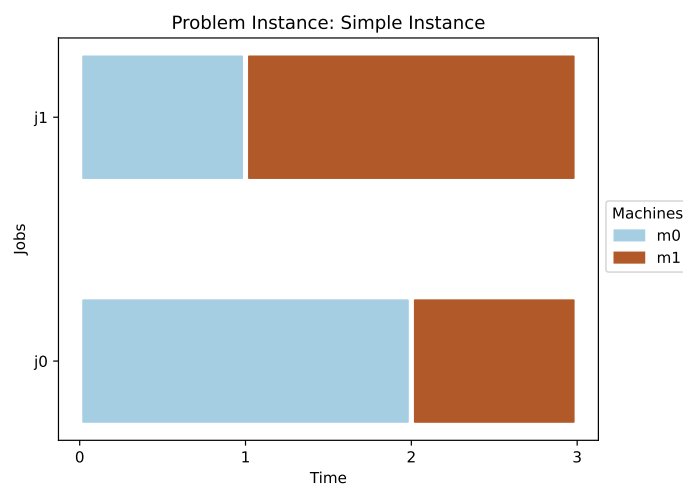


Figure 7.3.: This figure shows the problem instance used for the optimisation run on real quantum hardware. Its optimal makespan is 4, and it needs 8 qubits to optimise with a makespan limit of 5.

8. Results

In this chapter, we present the results for the hyperparameter optimisation and the benchmarks as outlined in the previous chapter. The results for the hyperparameter optimisation are presented in Section 8.1, the results for the noiseless simulations are presented in Section 8.2, and the results for the noisy simulation are presented in Section 8.3. Finally, in Section 8.5, the results are discussed.

8.1. Hyperparameter Optimisation

In this section, the results of the hyperparameter optimisation are presented. For the hyperparameter optimisation, we used multiple cost functions. As a result, for a configuration to definitely be better than another, it needs to be better with regard to all cost functions. Therefore, there can be multiple best configurations, none of which is definitely better than the other, with each of these configurations representing a trade-off between the cost functions. In multi-objective optimisation, such solutions are called *Pareto optimal* and said to be part of the Pareto set, which is the set of solutions for which no better solution exists [BOM15].

As a result, SMAC3 does not find a single best configuration but instead returns a Pareto set of the best configurations it finds. It is then our task to select a configuration from the Pareto set that, in our opinion, represents the most acceptable trade-off between the cost functions. To ease that choice, we plot the main costs, namely the inverted state quality and the number of expectation value evaluations until termination, for each configuration candidate in the Pareto set. Since each configuration was evaluated by SMAC3 multiple times, we can plot both the mean and the quantiles of the observed costs. This helps to evaluate how reliably a configuration solves the training problem instances.

8.1.1. QAOA Pareto Optimal Configurations

For QAOA, SMAC3 found five best configurations during its optimisation procedure. These five configurations and their cost values can be seen in Figure 8.1. Neither of these configurations obtained particularly good inverted state qualities (ISQ), with their respective inverted state quality mean, ranging from about 98 to about 92. Remember that the best possible ISQ is 0, while the worst possible ISQ is 100. This indicates that neither of the QAOA configurations, on average, found valid or optimal solutions with high measurement probabilities. When comparing the configurations, small improvements in the average ISQ between configurations come at the cost of large increases in the average number of needed expectation value evaluations. This seems to hint at the fact that QAOA is having serious difficulties finding quantum states with larger measurement likelihoods of valid or optimal solutions. Therefore, we choose the configuration numbered 1 in Figure 8.1, as slight improvements in measurement likelihoods do not seem to be worth an increase in the number of expectation value evaluations by a factor of two to five.

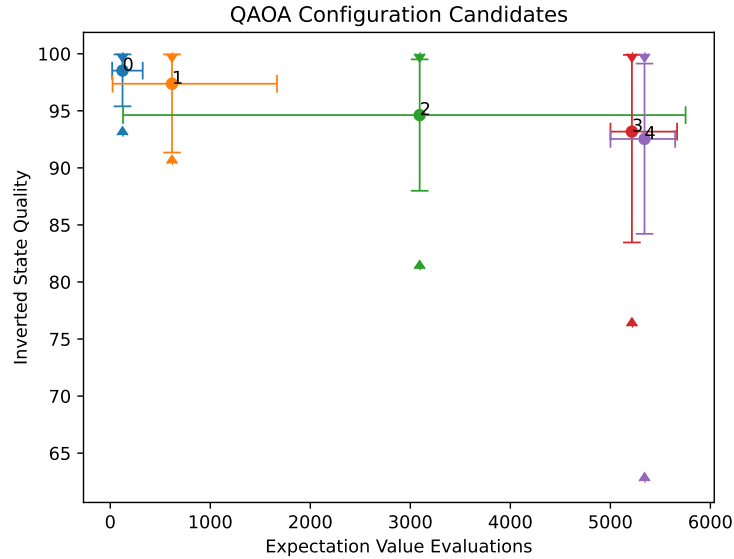


Figure 8.1.: This figure shows the average cost values for the best QAOA hyperparameter configurations found by SMAC3. Lower values are better for both the inverted state quality and the expectation value evaluations. The error bars show the 0.1 and 0.9 quantiles of the cost values, respectively. The triangles denote the absolute minimum and absolute maximum of the ISQ observed for each configuration. The individual configurations are numbered to ease referencing them.

The hyperparameter values for all configuration candidates can be found in Table B.1 in Appendix B. While discussing each chosen parameter value in detail is out of scope, we make some remarks on the parameter values chosen in configuration 1. In particular, SMAC3 chose 3 layer repetitions for configuration 1. For the other configurations in the Pareto set, the layer repetitions varied between 2 and 3 layers. No configuration increased the number of layers to 4. This seems to fit with previous findings by Plewa et al. [PSR21] that increasing the number of ansatz layers is not always beneficial. Furthermore, for configuration 1, a high SPSA perturbation of 0.28 and a high SPSA learning rate of 0.46 were chosen, which seems to fit with previous findings that SPSA benefits from large perturbations and learning rates [Loc22]. Another interesting observation is that the QAOA Pareto set configurations, including configuration 1, tend to enable SPSA’s blocking feature, which prevents bad parameter updates steps. SMAC3 enabled this feature only for QAOA, indicating that the search space spanned by QAOA’s ansatz is particularly difficult to navigate.

8.1.2. VQE Pareto Optimal Configurations

For VQE, SMAC3 found eight best configurations during its optimisation procedure. The configurations and their respective cost values can be seen in Figure 8.2. While the range of needed expectation value evaluations for the VQE configurations is similar to that of QAOA, the VQE configurations found better final states than QAOA. This can be seen in the mean ISQ for the configurations, which ranges from about 45 to about 10. The fact that for each configuration, the mean ISQ falls below 50 means that, on average, each configuration finds

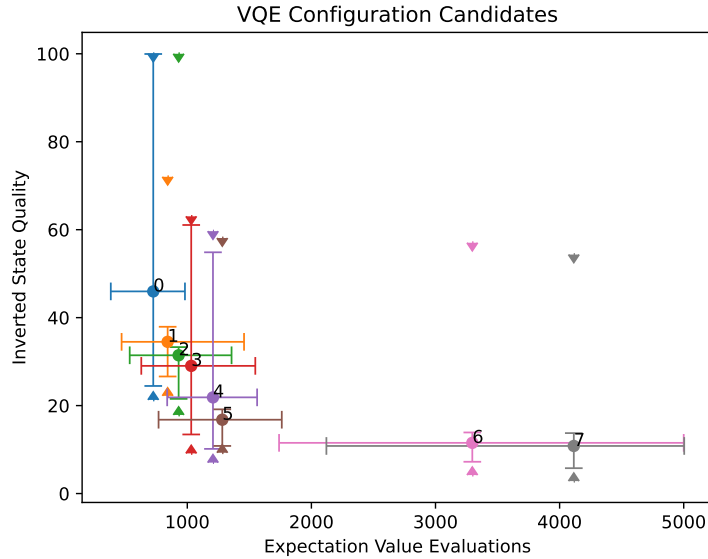


Figure 8.2.: This figure shows the average cost values for the best VQE hyperparameter configurations found by SMAC3 over all their optimisation runs. Lower values are better for both the inverted state quality and the expectation value evaluations. The error bars show the 0.1 and 0.9 quantiles of the cost values, respectively. The triangles denote the absolute minimum and absolute maximum of the ISQ observed for each configuration. The individual configurations are numbered to ease referencing them.

states with at least some measurement likelihood for optimal solutions. This is due to the fact that an ISQ below 50 can only be reached if the state contains optimal solutions. It can also be seen that up to configuration 5, increased numbers of needed expectation value evaluations lead to strongly improved mean ISQ values. But after configuration 5, further increases in expectation value evaluations only yield slight improvements in ISQ. As a result, we choose configuration 5, as it seems to be the best trade-off between the number of needed expectation value evaluations and the quality of the resulting state.

The hyperparameter values for all VQE configuration candidates can be found in Table B.2 in Appendix B. Like for our chosen QAOA configuration, VQE configuration 5 again employs relatively high SPSA perturbation and learning rate values of 0.21 and 0.26, respectively. Furthermore, configuration 5 employs only 2 ansatz layer repetitions, which is the minimum number of ansatz layers SMAC3 was allowed to choose. Interestingly, all VQE configurations apart from 6 and 7 employ only 2 ansatz layer repetitions. Only the configurations 6 and 7 employ 3 ansatz layer repetitions. This seems to indicate that increasing the number of ansatz circuit layers too much is not worth it for the given training problem instances.

8.1.3. EVQE Pareto Optimal Configurations

For EVQE, SMAC3 found nine best configurations during its optimisation procedure. Seven out of these nine configurations, with their respective cost values, can be seen in Figure 8.3. The other two configurations are omitted from the figure, as they were much worse than the

8. Results

rest in expectation value evaluations and state quality. To be exact, they needed more than 20,000 expectation value evaluations and only got an ISQ of about 80. For comparison, all other EVQE configurations remained below 12000 expectation value evaluations and below inverted state qualities of 50. SMAC3 still returned the "bad configurations" as part of the Pareto set, as they achieved very low ansatz depth and CU3 counts, which are the additional cost functions we defined for EVQE. A low-depth ansatz with few CU3 gates is not useful if it is not suited to finding good solutions, so we discarded these two configurations.

From Figure 8.3 it is clear, that the EVQE configuration candidates found by SMAC3 are sufficiently good at finding good final states, as for all configurations, the mean ISQ is far below 50. Like for VQE, for EVQE, an increase in the mean expectation value evaluations used per configuration yields improvements in the mean ISQ. Yet, these improvements are less pronounced, with configuration 3 barely yielding any improvement over configuration 1, even though it uses close to twice the number of expectation value evaluations. Configurations 4, 5, and 6 improve the mean ISQ over configuration 1, but they also need twice to thrice the number of expectation value evaluations. As a result, we choose configuration 1 as it needs few expectation value evaluations while providing good ISQ scores with less variance than the configurations 0, 2, and 3.

The hyperparameter values for all configuration candidates can be found in Table B.3 in Appendix B. Just like for QAOA and VQE, for configuration 1 and all other configurations, SMAC3 used very high perturbation and learning rate values for SPSA.

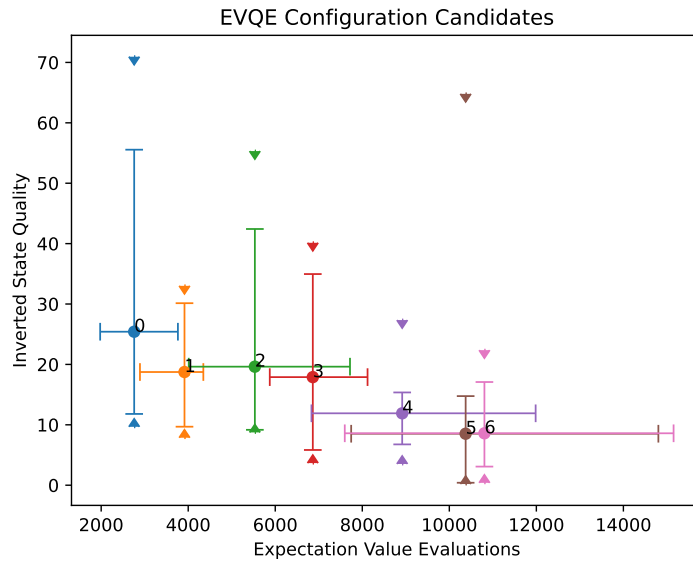


Figure 8.3.: This figure shows the average cost values for the best EVQE hyperparameter configurations found by SMAC3 over all their optimisation runs. Lower values are better for both the inverted state quality and the expectation value quality. The error bars show the 0.1 and 0.9 quantiles of the cost values, respectively. The triangles denote the absolute minimum and absolute maximum of the ISQ observed for each configuration. The individual configurations are numbered to ease referencing them.

8.2. Noiseless Simulation

For the evaluation of the scaling of EVQE, VQE, and QAOA over increasing problem sizes, we ran each algorithm 25 times for each of the problem sizes 12, 15, 18, and 21 qubits. In the following, we will first evaluate the termination behaviour of the algorithms. Then we will evaluate how successful the algorithms were and how good the quantum states found by the VQAs are. Following that, we discuss how quickly the VQAs converge to the various boundary values we defined. Finally, we evaluate the impact of the population size on EVQE.

We will often use box plots to aggregate the results of the 25 optimisation runs per problem size. These box plots are always defined as follows: The horizontal line within the box denotes the median of the result values. The lower and upper edges of the box denote the 0.25 and 0.75 quantiles of the result values. The lower and upper whiskers of the boxes denote the minimum and maximum of the result values.

8.2.1. Termination Behaviour

For each VQA optimisation run, we kept track of the number of expectation values the VQAs needed to evaluate until they were terminated by our termination criterion. As explained in Section 7.2, we refer to that count as $NEXP_{term}$. For each problem size and VQA, we aggregate the 25 obtained $NEXP_{term}$ values using box plots. The results of this are shown in Figure 8.4. Intuitively, as long as the termination criterion is well-behaved, we would expect the $NEXP_{term}$ values to increase over each problem size for all algorithms as the computational effort to solve more complicated problem instances rises. Such a behaviour can be seen for EVQE, which is shown in Figure 8.4 with the blue box plots.

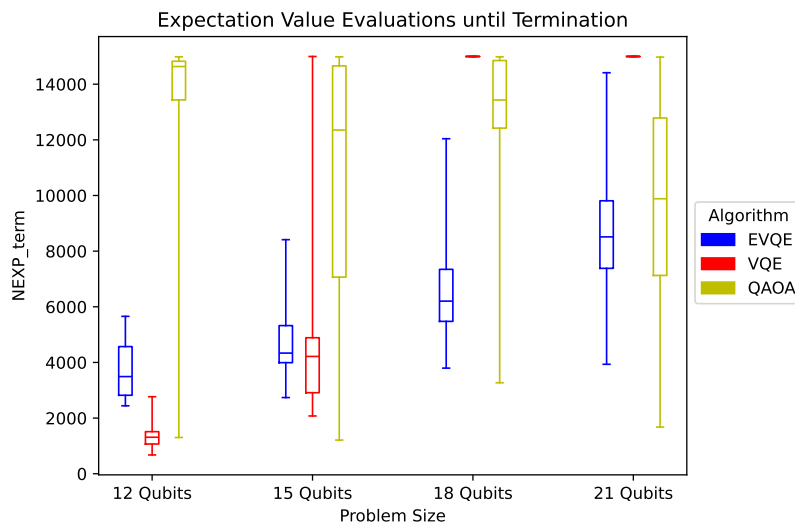


Figure 8.4.: This figure shows the number of expectation value evaluations needed for the algorithms to terminate over multiple problem sizes. Since for each problem size, each algorithm was run 25 times, box plots are used to aggregate the individual results. The lower and upper edges of the boxes show the 0.25 and 0.75 quantiles, respectively. The horizontal line within the boxes shows the median, and the whiskers extending from the box show the minimum and maximum values.

8. Results

For VQE, shown with the red box plots, the termination criterion seems to mostly work well for the problem sizes of 12 and 15 qubits. From 18 qubits upwards, VQE never seems to terminate, leading it to always run into the 15000 expectation value evaluation limit. This is the case even though VQE does find good solutions, as we will discuss in the next subsection. Having a closer look at an individual optimisation run of VQE in Figure 8.5a, it becomes obvious that the fact that VQE does not terminate stems from the expectation values varying strongly between optimisation iterations. Since our termination criterion is based on the stability of expectation values between iterations, these swings in expectation

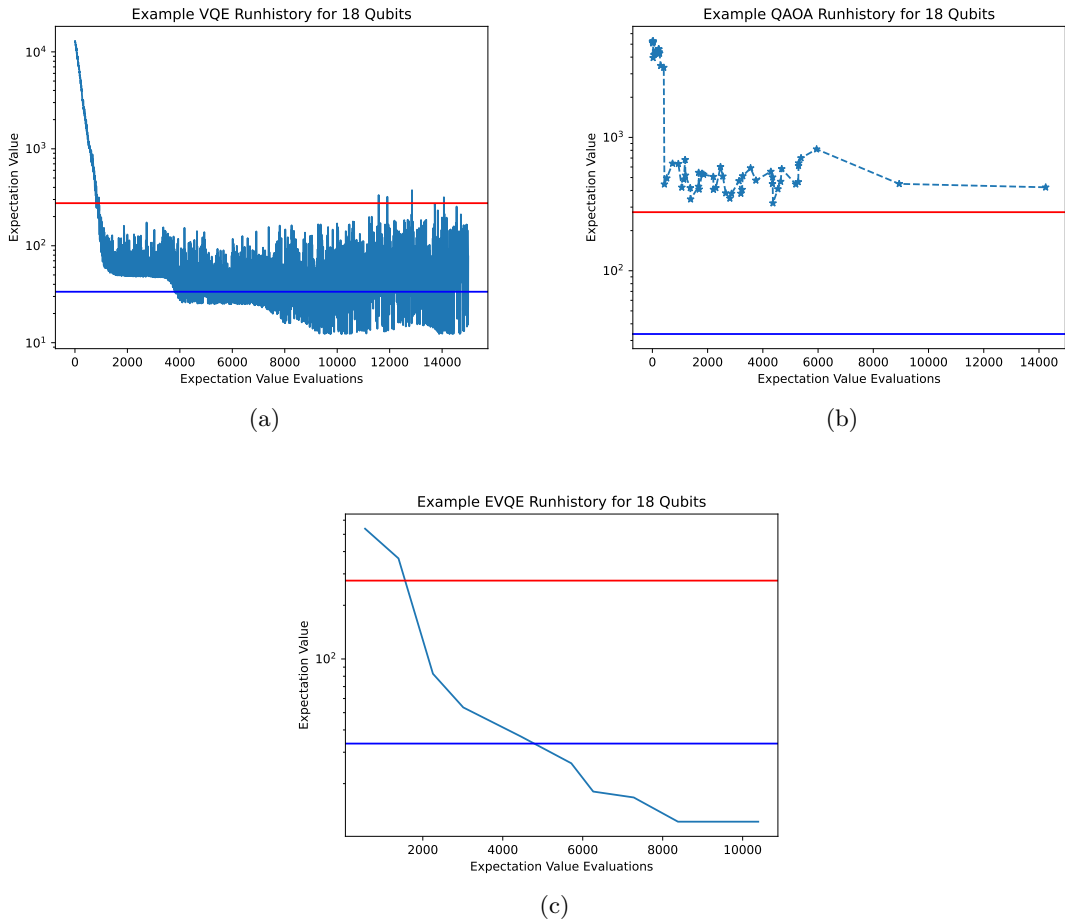


Figure 8.5.: These figures show one optimisation run for VQE (a), QAOA (b), and EVQE (c) for the third 18-qubit JSSP benchmarking instance shown in Figure A.5. The red line denotes the boundary value E_{bval} , and the blue line denotes the boundary value E_{bopt} . VQE finds optimal solutions, but its expectation values vary strongly between optimisation iterations. QAOA does not seem to find good solutions. It also experiences swings in expectation value. Furthermore, the long distances between reported expectation values indicate that SPSA is having difficulties finding good parameter value updates, as it rejects many update steps.

value prevent VQE from terminating.

For QAOA, shown with the yellow box plots, on the other hand, it seems that the optimisation process works very inconsistently, as there are large variances in the $NEXP_{term}$ values. There is also no identifiable trend of the number of expectation value evaluations rising over increasing problem sizes. This matches the bad solution quality obtained by QAOA, which we discuss in the next subsection. For now, let us look at an individual optimisation run of QAOA in Figure 8.5b. As it can be seen, like for VQE, there is a wide variance between expectation values between iterations. But in contrast to VQE, QAOA never finds expectation values below the boundary value E_{bval} , shown in red, indicating that QAOA does not even find valid solutions with a high measurement likelihood. Furthermore, there are long distances between reported expectation values. This is due to the fact that for QAOA’s configuration, as explained earlier, SPSA’s blocking parameter was enabled. This parameter prevents update steps, which would worsen the expectation value too much. As a result, one can infer from the long pauses between reported expectation values that SPSA often rejects many update steps consecutively. This implies that SPSA, as it was configured for QAOA, has major issues stepping effectively through the search space spanned by QAOA’s ansatz.

Looking at a single EVQE optimisation run in Figure 8.5c, it is clear that EVQE’s well-behaved termination is due to the fact that its reported expectation values are much more consistent between optimisation iterations. This comes as no surprise, as in EVQE, an optimisation iteration encompasses a whole generation of its population. As a result, the reported best expectation value per generation has been aggregated over many individuals, which makes it much more robust to the stochastic nature of SPSA’s gradient approximation.

In summary, the SPSA optimizer does not seem to be very stable for VQE and QAOA. This prevents QAOA and VQE from terminating due to our termination criterion being based on the stability of the observed expectation values. As a result, the $NEXP_{term}$ values reported in Figure 8.4 are not particularly useful for gauging the computational effort needed by VQE and QAOA. Still, it is interesting that EVQE enables a decoupling of the termination criterion from the instabilities of the classical optimizer used in its subroutines.

8.2.2. Solution Quality and Success Rate

As explained in Section 7.2, we rate the quality of the quantum state found by the VQAs by evaluating both the likelihood of measuring a solution with an optimal makespan P_{opt} and the likelihood of measuring a valid solution P_{val} from that state. Aggregating the results from our benchmarks yields Figure 8.6a for the measurement likelihoods of valid solutions and Figure 8.6b for the measurement likelihoods of optimal solutions.

For QAOA, it can be seen that in the median case, for 12 qubit problem instances, QAOA finds valid solutions with a measurement likelihood below 10%. For increasing problem sizes, the median measurement likelihood of both valid and optimal solutions quickly tends towards zero. This indicates that QAOA, in its basic form, with the configuration found by SMAC3, is not likely to find either valid or optimal solutions for increasing problem sizes.

For both VQE and EVQE, it can be seen in Figure 8.6b that for nearly all problem sizes, the 0.25-quantile of the measurement probabilities for optimal solutions is above 60%. This indicates that in the majority of the optimisation runs, both VQE and EVQE successfully find optimal solutions with large measurement probabilities. In the median case, the measurement likelihoods of VQE seem to be about 10% higher than the measurement likelihoods provided by EVQE. Still, both measurement likelihoods of VQE and EVQE are surprisingly

8. Results

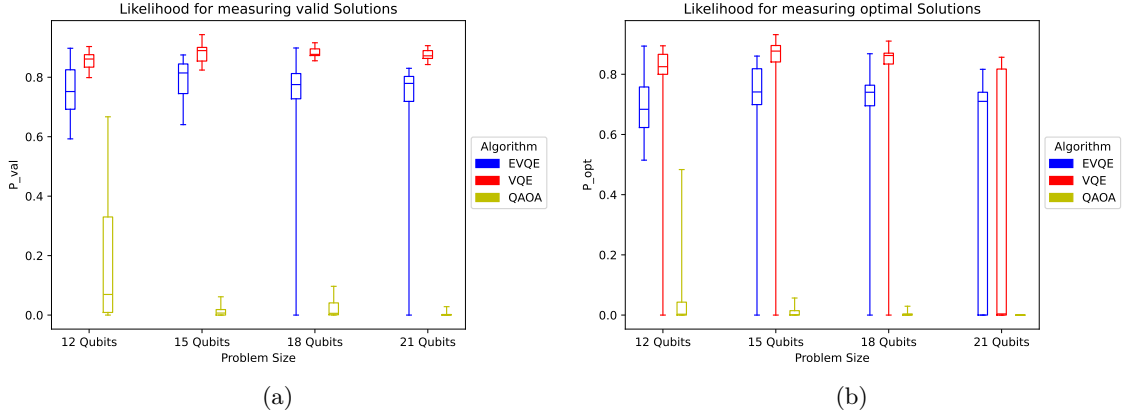


Figure 8.6.: These figures show the likelihood of measuring a valid solution P_{val} (a) or an optimal solution P_{opt} (b) when measuring the best quantum state found by the VQAs over multiple problem sizes. Since for each problem size, each algorithm is run 25 times, box plots are used to aggregate the results.

high, as the CVaR objective only rewards measurement likelihoods up to 50%.

Yet, as the lower box plot whiskers indicate, there are still outlier optimisation runs in which VQE or EVQE find quantum states that do not contain optimal solutions. While VQE always at least finds valid solutions, as can be seen in Figure 8.6a, EVQE even sometimes seems to get stuck in states with zero measurement likelihood for valid solutions. To investigate how often such bad optimisation runs actually happen, we have a look at the “success rate” for the VQAs. We define a VQA run to be successful with respect to valid solutions if, in the quantum state found by the VQA, the measurement likelihood for valid solutions is at least 1%. Similarly, we define a VQA run to be successful with respect to optimal solutions if, in the quantum state found by the VQA, the measurement likelihood

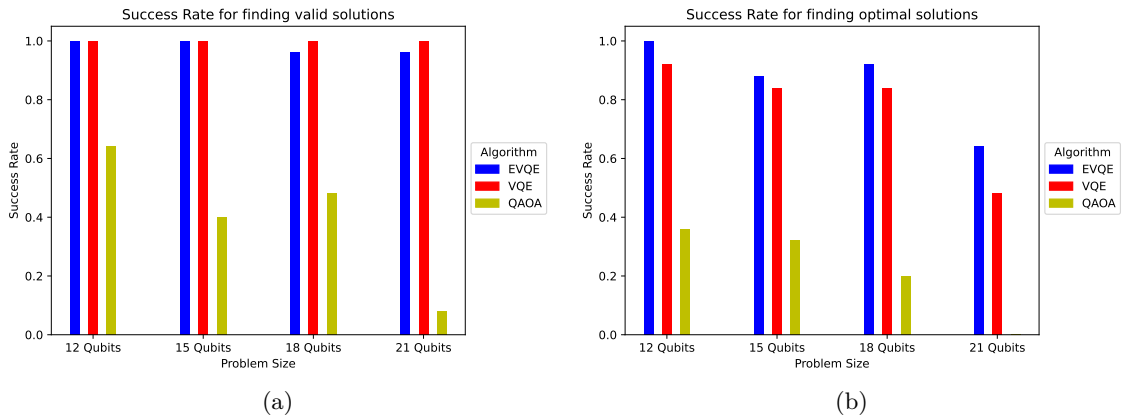


Figure 8.7.: These figures show the success rate for the VQAs over multiple problem sizes of finding quantum states in which valid (a) or optimal (b) solutions can be measured with a likelihood higher than 0.01.

for an optimal solution is at least 1%. In Figure 8.7a, we now plot what portion of the 25 optimisation runs per algorithm and problem size are successful in finding valid solutions. In Figure 8.7b, we compare what portion of them is successful in finding optimal solutions.

The fact that QAOA’s success rate is non-zero for most problem sizes shows that for at least some optimisation runs, it finds measurement likelihoods that exceed 1%. Even so, its success rate plummets with increasing problem sizes, again indicating that the basic form of QAOA does not scale well with increasing problem sizes.

For VQE, it can be seen in Figure 8.7a that for all optimisation runs, VQE successfully finds valid solutions. While this is also mostly true for EVQE, it can be seen that EVQE has a very low likelihood of not being successful in finding valid states. Interestingly, in the success rate for finding optimal solutions, EVQE outperforms VQE. This can be seen in Figure 8.7b. Here, EVQE is consistently more likely to be successful in finding quantum states that contain optimal solutions.

8.2.3. Convergence Speed

Given that the computational effort until termination is not a fair metric for comparing the VQAs, as our termination criterion does not seem to work well for QAOA and VQE, we now compare how much computational effort the VQAs need to reach certain milestones during the optimisation procedure. The exact milestones we use were discussed in Section 7.2. As a short summary, we consider the number of expectation value evaluations needed to reach a quantum state, which must contain valid solutions $NEXP_{bval}$. We also consider the number of expectation value evaluations needed to reach a quantum state, which must contain optimal solutions $NEXP_{bopt}$. Finally, we consider the number of expectation value evaluations needed to reach a quantum state with the lowest expectation value seen by the VQA, referred to as $NEXP_{best}$. We aggregate the results for the 25 optimisation runs for each problem size and algorithm using box plots. Figure 8.8a shows the number of expectation values needed to reach a quantum state that must contain valid solutions. Figure 8.8b shows the number of expectation value evaluations needed to reach a state that must contain optimal solutions. Finally, Figure 8.8c shows the number of expectation value evaluations needed to reach the best quantum state seen by the VQA.

For QAOA, the number of expectation value evaluations needed to reach its best observed quantum state varies highly within the problem sizes, but no identifiable trend can be seen over increasing problem sizes. Yet, the best quantum states QAOA reaches, generally, are not good, as discussed in the previous subsection. This can be seen in Figures 8.8a and 8.8b, in which data on QAOA is mostly missing. This is due to the fact that for larger problem sizes, QAOA never reaches states with expectation values below the boundaries E_{bval} and E_{bopt} , which we use to identify whether a quantum state must contain valid or optimal solutions. Therefore, we cannot plot the number of expectation value evaluations needed by QAOA as it never reaches these milestones.

For VQE and EVQE, it can be seen in Figure 8.8a that VQE for all problem sizes is faster to reach quantum states, which must contain valid solutions. While for both VQE and EVQE, the computational effort to reach such quantum states seems to rise linearly with the problem size, it is interesting to observe that the effort needed by EVQE seems to rise faster than that of VQE. This trend is reversed in Figures 8.8b and 8.8c. While for smaller problem instances, VQE still needs less computational effort to reach quantum states that contain optimal solutions, for increasing problem sizes, the computational effort needed by

8. Results

VQE rises drastically. As a result, VQE needs more computational effort than EVQE for the 21-qubit problem size. At the same time, the variance in the computational effort needed by VQE also rises with the problem size. This shows that for larger problem sizes, EVQE is not only faster to obtain optimal solutions but is also more consistent in the computational effort it needs.

Given the results from this and the previous subsection, it seems clear that VQE is more useful when it suffices to find any valid solution, whereas EVQE is beneficial for larger problem instances when it matters that solutions with as high a quality as possible should be found. QAOA, on the other hand, given our results so far, does not seem like a promising competitor to VQE or QAOA due to the low solution quality it provides with seemingly no advantage in computational effort.

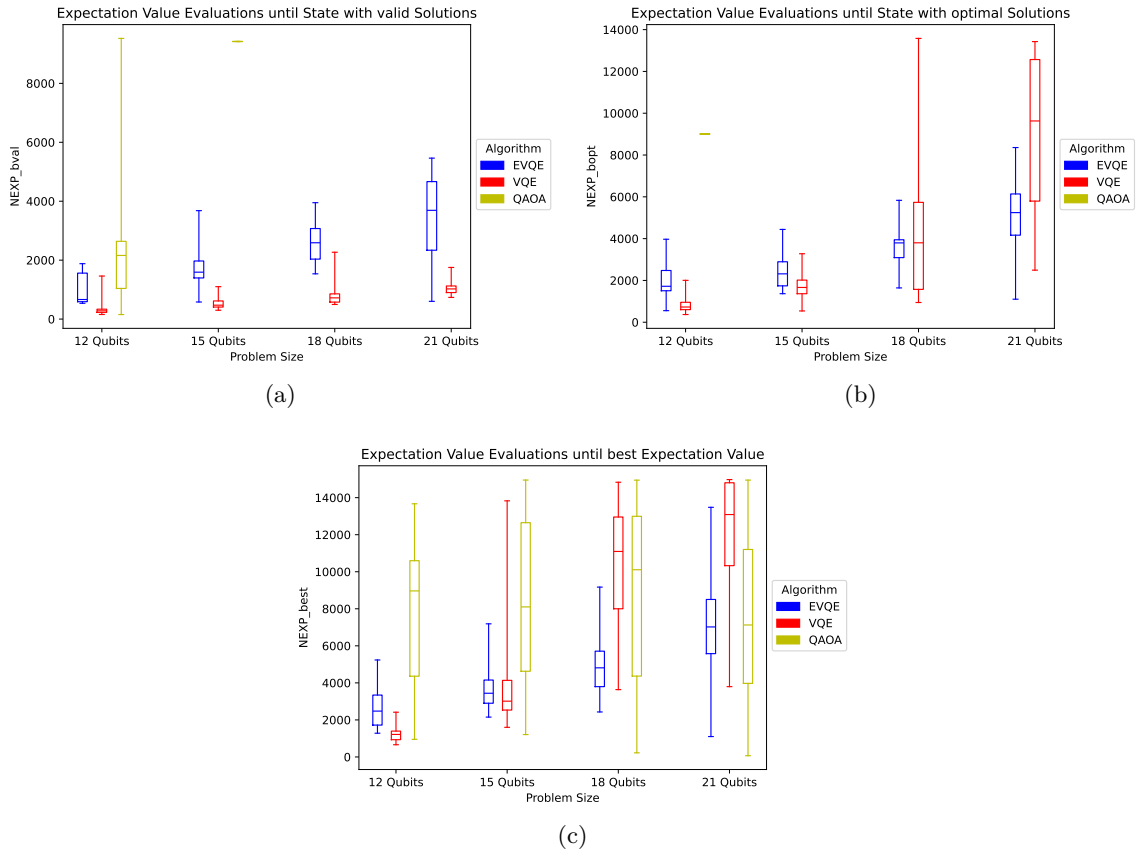


Figure 8.8.: These figures show how many expectation value evaluations the VQAs need to hit certain milestones during the optimisation process for multiple problem sizes. Since for each problem size, each algorithm was run 25 times, these values are aggregated using box plots. In (a), it is shown how many expectation value evaluations the VQAs need to reach expectation values below the boundary E_{bval} . In (b), it is shown how many expectation value evaluations the VQAs need to reach expectation values below the boundary E_{bopt} . In (c), it is shown how many expectation value evaluations the VQAs need to reach the quantum state with the lowest expectation value seen during the optimisation procedure.

8.2.4. Ansatz Complexity

Since EVQE adapts its ansatz circuits during the optimisation process, it is interesting to see how complex the ansatz circuits found during the optimisation procedure are. To that end, we investigate the ansatz circuit depth and controlled gate count for the ansatz circuit used by EVQE to reach the best quantum state. We also compare them to the ansatz depth and controlled gate count for QAOA and VQE. The comparison of the ansatz depth over the benchmarking problem sizes can be seen in Figure 8.9a. The comparison of the controlled gate count over the problem sizes can be seen in Figure 8.9b.

In those figures, it can be seen that the problem-inspired ansatz circuits of QAOA are much deeper and use many more controlled gates than both the hardware-efficient VQE ansatz and the ansatz circuits generated by EVQE. It can also be seen that, as we fixed the number of layer repetitions for VQE at 2, the ansatz circuit depth for VQE was constant over the problem sizes. Yet, for increasing numbers of qubits, more controlled gates were needed in the VQE ansatz.

For all problem sizes and the majority of the optimisation runs, the ansatz depth found by EVQE was lower than the depth of the fixed ansatz of VQE. It is also apparent that the ansatz circuit depth found by EVQE increased with increasing problem sizes. This indicates that EVQE only creates ansatz circuits as deep as needed to solve the problem instances of increasing complexity. For the controlled gate count, it can be seen that until the 18 qubit problem sizes, the majority of the EVQE optimisation runs contained fewer controlled gates in the final ansatz circuit than the fixed ansatz of VQE. Yet, it can also be seen that with increasing problem sizes, the number of controlled gates rises faster for EVQE than for VQE. This leads to EVQE needing more controlled gates than VQE for 21 qubits. One possible reason for the strong increase in the number of controlled quantum gates over increasing problem sizes is that SMAC3 chose a very small β penalty for EVQE. This penalty decreases the fitness of individuals with too many controlled gates. Therefore, setting it too low might have incentivised EVQE to add controlled gates more often than necessary.

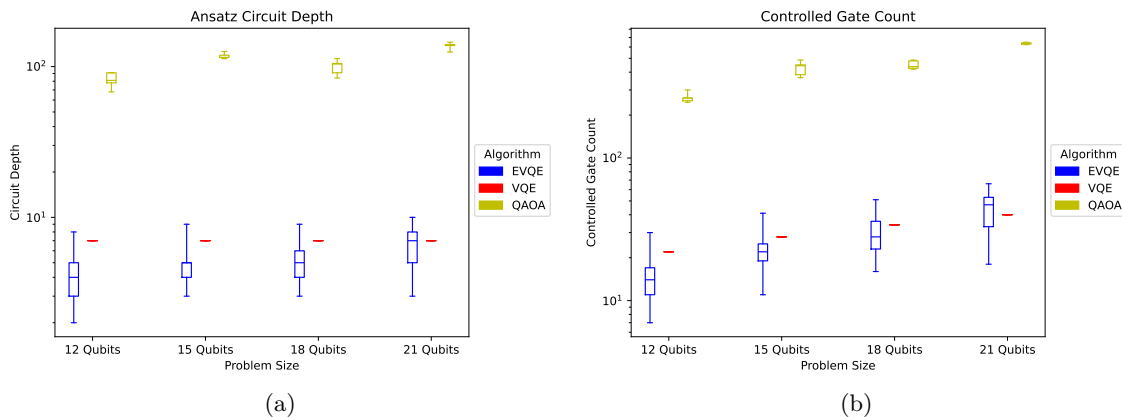


Figure 8.9.: These figures show the depth of the ansatz circuit over all problem sizes and algorithms, aggregated using box plots. The ansatz circuit depth is shown in (a), and the number of controlled gates is shown in (b).

8.2.5. Comparison of EVQE Population Sizes

As we noted in Section 7.5, we did not let SMAC3 choose a population size during the hyperparameter optimisation. Instead, during the hyperparameter optimisation and the benchmarks shown so far, we used EVQE with a fixed population size of 10. To gauge how much the population size influences the performance of EVQE, we also ran the same benchmarks for EVQE with the population sizes 5 and 20. Selected results from these benchmarks are compared to the results provided by EVQE with a population size of 10 in the Figure 8.10.

In Figure 8.10a, we compare the computational effort needed by EVQE with a population size of 5 (EVQE 5), EVQE with a population size of 10 (EVQE 10), and EVQE with a population size of 20 (EVQE 20) to terminate. As it can be seen, increasing the population

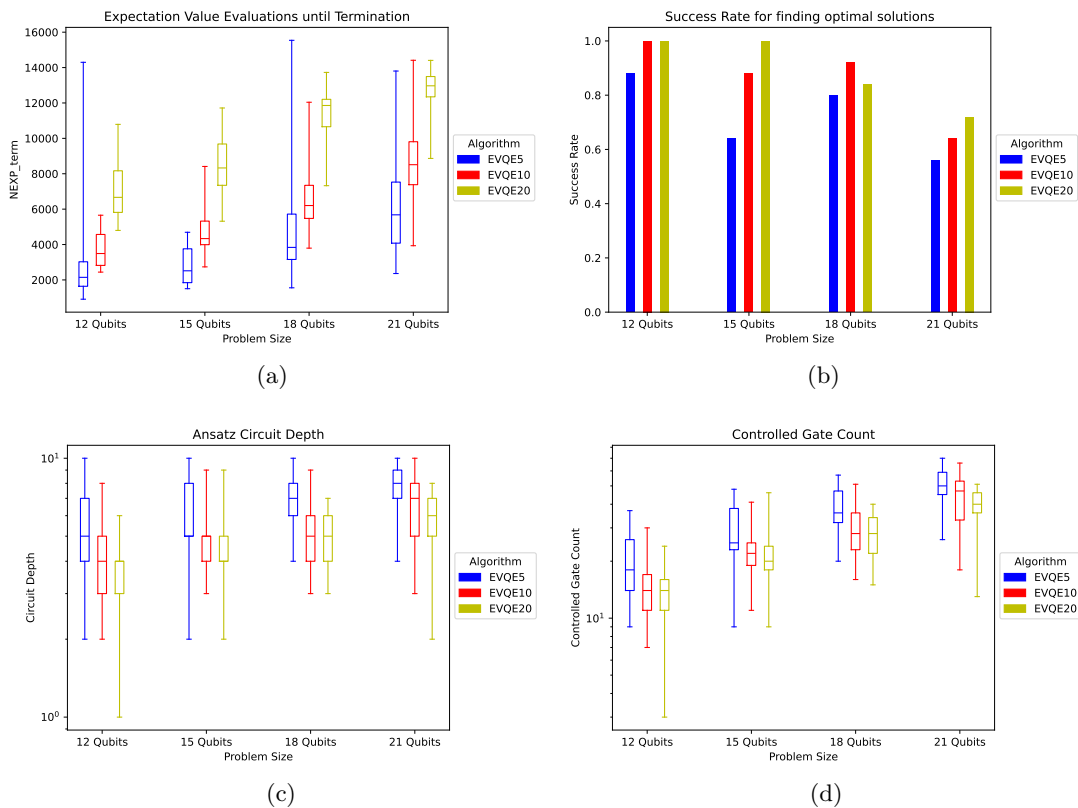


Figure 8.10.: These figures show how changing the population size of EVQE influences the performance of EVQE. To that end, they represent the results of running EVQE with the population sizes 5, 10, and 20 for each problem size. In (a), the computational effort needed by these EVQE variants over the problem sizes is shown. In (b), it is shown how many of the EVQE optimisation runs are successful at finding optimal solutions. Success is defined as measuring an optimal solution from the quantum state found by EVQE with a likelihood higher than 0.01. In (c), the depth of the best ansatz and in (d), the number of controlled gates in the best ansatz found by EVQE during each optimisation run is shown.

size increases the number of expectation value evaluations needed by the same factor as the population size increase. This comes as no surprise, as an increase in the number of individuals within the population also increases the number of times the last layer optimisation and the other variation operators are applied during each generation. Interestingly, the same pattern occurred for the computational effort needed to reach the milestones we defined during the optimisation procedure, implying that larger population sizes do not aid with the speed of convergence.

We also noted that increasing or decreasing the population size does not affect the measurement likelihoods of good solutions in the quantum states if the optimisation run was successful. But it can be seen that increasing the population size increases the portion of optimisation runs that are successful in finding quantum states that contain optimal solutions. This behaviour is shown in Figure 8.10b. Thus, it seems likely that larger population sizes help EVQE explore a wider range of ansatz circuits, which increasingly help it to avoid or escape local minima or barren plateaus.

The effect of a wider exploration of ansatz circuits can also be seen in the Figures 8.10c and 8.10d. There it is shown that for larger population sizes, EVQE generally finds ansatz circuits of lower depth and lower controlled gate counts.

8.3. Noisy Simulation

To investigate how noise affects the VQAs, we ran each VQA 25 times on the 18 qubit problem instances, using different numbers of shots to induce varying amounts of shot noise. Since VQAs are explicitly designed for the usage on noisy quantum hardware, we expect that each VQA should be able to cope with the shot noise to some extent. Still, it is interesting to see whether EVQE is inherently more resistant to noise than the other algorithms.

In Figure 8.11a, we then compare how fast the VQAs reach quantum states, which contain optimal solutions, over the different shot noise scenarios. QAOA, in no optimisation run,

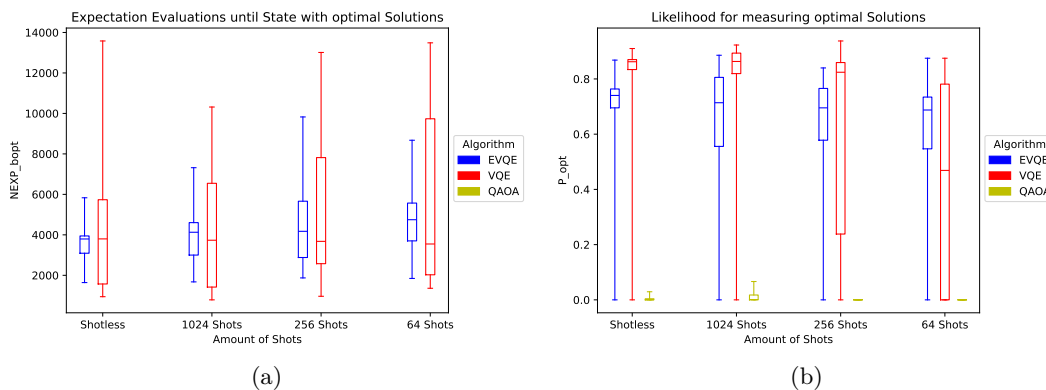


Figure 8.11.: These figures show how EVQE, VQE, and QAOA compare under varying shot noise scenarios. Shotless refers to the noise-free state vector simulation. In (a), it is shown how many expectation value evaluations the VQAs need to reach a quantum state that contains optimal solutions. In (b) it is shown how likely it is to measure an optimal solution from the quantum states found by the VQAs.

8. Results

reached such quantum states, which is why it is missing from that plot. For both EVQE and VQE, the effect of the increasing shot noise can be seen. For EVQE, the median number of expectation value evaluations needed increases slightly over the increasingly noisy scenarios. For VQE, this does not happen, and the median number of needed expectation value evaluations remains about the same over all shot noise scenarios. Yet interestingly, the variance in the number of needed expectation evaluations for VQE raises drastically over increasingly noisy scenarios. Such a large increase in variance cannot be observed for EVQE, indicating that EVQE is much more consistent in its convergence speed even in highly noisy scenarios.

This can also be observed for the quality of the quantum states found by the VQAs in noisy scenarios. We show the likelihood of measuring optimal solutions from the quantum states found by the VQAs in Figure 8.11b. For QAOA, that likelihood is always close to zero. For low shot noise scenarios, both VQE and EVQE provide high measurement likelihoods of optimal solutions with outliers towards 0, if the optimisation run was unsuccessful. Initially, as observed for the noiseless statevector simulations, the measurement likelihoods of VQE are higher than those for EVQE. But for increasing amounts of shot noise, it can be seen that VQE's measurement likelihoods begin to vary strongly and finally fall well below EVQE's measurement likelihoods for the 64 shot scenario.

Both plots together indicate that, under high shot noise levels, VQE is much more inconsistent than EVQE, even though we used the noise-tolerant SPSA optimisation algorithm for both VQAs.

8.4. Real Quantum Hardware

As explained in Section 7.6, EVQE was run on the IBM *ibm_nazca* 127-qubit quantum computer thanks to E.ON Digital Technology. It was configured to run for three generations with a population size of 5. The results of that optimisation run can be seen in Figure 8.12.

The best expectation values found per generation are shown in Figure 8.12a. It can be seen, that EVQE, after the first generation, has already found the optimal expectation value of 22.75. As a result, subsequent generations cannot improve on that expectation value, explaining why the graph in Figure 8.12a is a flat line.

Figure 8.12b then shows the distribution of classical states when measuring the ansatz of the best individual. As it can be seen, state 5 (00000101 in binary) has the highest likelihood of being observed. It represents the optimal solution schedule shown in Figure 8.12c. Since this is the only possible optimal solution, P_{opt} is 0.73 for this optimisation run. P_{val} is only slightly higher at 0.74.

These results show that EVQE can find optimal solutions to the JSSP even on real, noisy quantum hardware.

8.5. Discussion

As discussed in Chapter 3, evolving ansatz VQE algorithms have been shown to be promising due to their noise resistance and their ability to escape local minima and avoid barren plateaus. These advantages are gained by evaluating and adapting a whole population of ansatz circuits during the normal VQA routine. This naturally needs more computational effort in terms of expectation value evaluations than for VQA routines that use only one fixed

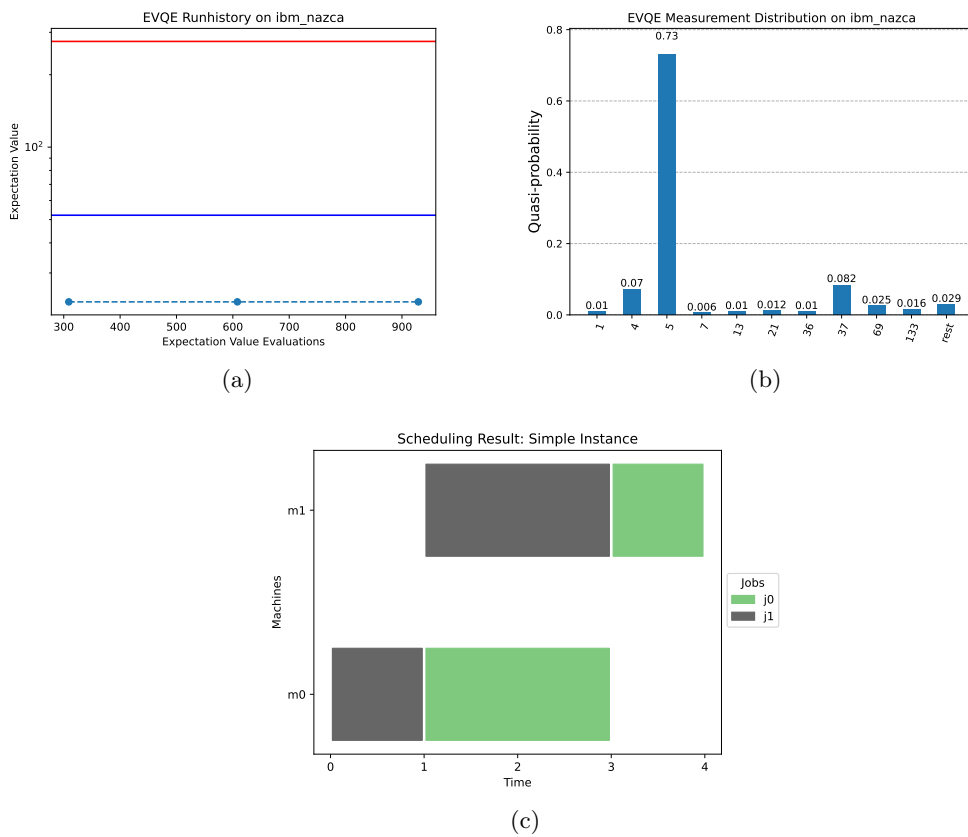


Figure 8.12.: These figures show the results for the EVQE optimisation run on IBM’s *ibm_nazca* 127-qubit quantum computer. In (a), the best expectation value found during each generation is shown. The red horizontal line denotes the boundary value E_{bval} , and the blue horizontal line denotes the boundary value E_{bopt} . EVQE finds the best possible expectation value during the first generation. As a result, it cannot improve the expectation value further during subsequent generations. In (b), the measurement distribution of classical states, retrieved by measuring the ansatz of the best individual multiple times, is shown. For readability, the bitstrings retrieved by the measurement are referred to by their decimal representation. State 5 with the measurement probability of 73% translates to the schedule with the optimal makespan 4, shown in (c).

ansatz, such as VQE. The goal of our benchmarks was to determine how this additional effort, in the case of EVQE, scales over increasing problem sizes, as this is central to determining whether the additional computational effort is worth the advantages afforded by evolving ansatz VQE methods.

We demonstrated the scaling of EVQE’s computational effort in Figures 8.4, 8.8a, 8.8b, and 8.8c. These results clearly show that the computational effort of EVQE rises with increasing problem sizes. In particular, it can also be seen that the computational effort for finding valid solutions rises slower with the problem size, than the computational effort to

8. Results

find optimal solutions. Yet, due to the limited number of problem sizes and optimisation runs we evaluated, we cannot definitely determine whether the increase in computational effort is linear, polynomial, or even exponential.

While we cannot make exact statements on the computational complexity of EVQE, we also benchmarked the scaling of the computational effort needed by the established QAOA and VQE algorithms as a frame of reference. This enables us to compare EVQE’s scaling against more established VQAs. While QAOA did not provide good results, VQE reached good solutions and provided a good frame of reference for the computational effort needed to reach valid or optimal solutions. We could observe that VQE found valid solutions faster than EVQE and scaled better than EVQE for finding valid solutions. This is not necessarily surprising, due to the high computational effort involved in evolving and optimising multiple ansatz circuits. We see no indication that this trend may change for larger problem sizes. As a result, if finding any valid solution is enough, VQE likely seems like a better choice than EVQE. On the other hand, if a better solution or even an optimal solution is needed, the advantages of EVQE start to show. In our benchmarks, it could be seen that the computational effort for finding optimal solutions needed by VQE, while starting out lower than that of EVQE for small problem sizes, scaled worse than the computational effort needed by EVQE. This results in EVQE needing less computational effort on average than VQE to find optimal solutions for the problem size of 21 qubits. The fact that VQE scales worse than EVQE for this use case seems to be a clear trend. As a result, we expect EVQE’s advantage over VQE in terms of the computational effort to widen for problem instances of even larger sizes when optimal solutions are required. On top of that, it could also be seen that the success rate for finding optimal solutions for EVQE was consistently higher than that of VQE, meaning EVQE would need fewer optimisation runs than VQE to find optimal solutions. As a result, EVQE seems to be a more promising candidate than VQE for finding solutions of high quality.

VQE finding valid solutions quickly likely indicates that VQE converges to a local minimum but then has a hard time leaving that minimum and navigating the search space towards better solutions. EVQE’s population-based, adaptive ansatz approach, on the other hand, while more computationally expensive per iteration, seems to help with avoiding local minima and barren plateaus, as indicated in prior research. This might explain why EVQE outperforms VQE at finding optimal solutions for larger problem sizes, where the search space is increasingly complicated.

Note that for approximating the VQAs’ computational effort, we counted all expectation value evaluations needed by the VQAs. While, as we discussed in Section 7.2, this is a passable approximation of computational effort, it does not translate easily to approximating the runtime of EVQE. This is due to EVQE’s population-based nature, which enables many of these expectation value evaluations to be done in parallel. This stands in contrast to both VQE and QAOA, where parallelisation opportunities fully depend on the classical optimiser. In the case of the SPSA optimiser we used, all expectation value evaluations for QAOA and VQE were done in serial. This helped EVQE run much faster in simulation than VQE or QAOA. We did not present this finding with the other results, as access to quantum hardware is currently very limited and expensive. Therefore, assuming easy parallel access to multiple quantum computers seems premature. Yet, should quantum computers become ubiquitous, the easy parallelisation offered by EVQE might yield good speed-ups in runtime over VQE or QAOA.

Note that the degree of parallelisation is easily controlled by EVQE’s population size.

Increasing the population size allows for more parallel expectation value evaluations while at the same time enabling a wider exploration of the search space. We have shown that adjusting the population size of the EVQE algorithm allows a trade-off between its computational cost and the benefits offered by its population-based approach. Lower population sizes needed fewer expectation value evaluations but also offered lower success rates for finding optimal solutions, while larger population sizes increased the success rate and the number of expectation value evaluations. At the same time, these effects do not seem to scale linearly with the population size, so care needs to be taken when choosing the population size to avoid overt diminishing returns.

We also observed further advantages posed by EVQE's population-based approach. On the one hand, we have shown that EVQE, due to its population-based approach, is less affected by noisy optimisation iterations in its classical optimisation subroutine. This allows EVQE to be terminated based on a stability-based termination criterion. Furthermore, we have shown that EVQE behaves very consistently even under strong shot noise, which is likely also due to its population-based approach, as it is less affected by individual erroneous expectation value measurements. Finally, we have shown that these advantages enable EVQE to work well for optimising the JSSP on real, noisy, quantum hardware.

These findings suggest that there are some use cases in which the benefits of evolving ansatz VQE approaches outweigh their heavy computational cost, which in these cases allows evolving ansatz VQE approaches to outperform other VQAs such as VQE or QAOA.

9. Conclusion and Future Work

The goal of this thesis was to investigate how evolving ansatz VQE algorithms scale in terms of computational effort and solution quality for job shop scheduling problems. To that end, we chose to investigate and implement the EVQE algorithm in the open-source Python library QUEASARS¹². To enable the solving of JSSP instances with EVQE and VQA algorithms in general, we also implemented a mapping of JSSP instances to Ising Hamiltonians, which is a problem formulation widely used with VQAs. We then used a random problem instance generation scheme, similar to that proposed by Taillard et al. [Tai93], to generate a dataset of JSSP problem instances of varying sizes for our benchmarks. Using these problem instances, we benchmarked EVQE as well as VQE and QAOA on multiple JSSP instances of increasing sizes. For these benchmarks, we observed both the solution quality and the computational effort needed in terms of how often the expectation value has to be evaluated with respect to the ansatz circuit.

While QAOA did not provide good results, both EVQE and VQE were capable of providing good results for our benchmarking problem instances. When investigating the computational effort the VQAs needed to solve the JSSP benchmarking instances, we found that it had to be distinguished whether finding any valid solution to the JSSP is enough or whether solutions with an optimal makespan are desired. In the first case, our results showcased that VQE consistently found valid solutions to the JSSP instances, using few expectation value evaluations. This also scaled well to larger problem instances. EVQE, on the other hand, due to its population-based approach, needed many more expectation value evaluations to find valid solutions and scaled worse than VQE. The opposite could be observed if the goal is to find solutions with an optimal makespan. While for smaller problem instances, VQE still needs fewer expectation value evaluations than EVQE to find optimal solutions, VQE scales worse in this case. This leads to EVQE needing fewer expectation value evaluations than VQE to find optimal solutions for larger problem instances. We reason that EVQE’s ability to escape and avoid local minima and barren plateaus benefits it in finding optimal solutions across the search space as the search space becomes increasingly complex with increasing problem sizes. We expect this trend to continue for even larger problem instances, making EVQE the preferred choice over VQE if high-quality solutions are required.

Given the further benefits of EVQE we encountered, namely its good parallelizability, its stable optimisation progress that enables its consistent termination, its ability to consistently deal with noise, and its low-depth ansatz circuits, we believe that evolving ansatz VQE algorithms can be useful quantum heuristics where good solution qualities are required.

Future Work

For future work, we believe there are multiple promising avenues of research.

¹QUEASARS is licenced under the Apache Licence 2.0.

²The QUEASARS GitHub repository can be found at: <https://github.com/DLR-RB/QUEASARS>

9. Conclusion and Future Work

While we have shown that the high computational effort per generation in Evolving Ansatz VQE algorithms can be well worth it in certain scenarios, it would still be helpful to reduce the computational effort per generation as far as possible. Since evaluating the fitness in EVQE entails the optimisation of the last layer of an individual, which involves many expectation value evaluations, reducing the number of costly fitness evaluations per generation could drastically reduce the computational effort per generation of EVQE. One possible approach to this can be found in the work of Ansoategui et al. [AST09]. They split the population of the evolutionary algorithm into two pools of individuals with separate genders: the competitive and non-competitive genders. New individuals are randomly assigned to the gender. For the competitive gender, the fitness function is calculated as normal, whereas for the non-competitive gender, individuals are selected randomly without calculating their fitness. Ansoategui et al. have shown that this drastically reduces the number of costly fitness evaluations needed by the evolutionary algorithm, as only half the population needs to be evaluated per generation [AST09].

Another good avenue for future research would be to investigate improvements that have been proposed for the VQE algorithm. Such improvements include the application of filtering operators [ARF⁺22] or the use of quantum natural gradients [KB22], which can also be approximated within SPSA [GZCW21]. Many more improvements can be found in literature reviews for the VQE method [TCC⁺22] [BCLK⁺22]. These improvements could then be investigated twice. On the one hand, since we only benchmarked EVQE against the basic variant of VQE, it could stand to reason that improved versions of VQE might outperform EVQE even when searching for high-quality solutions. On the other hand, since the optimisation subroutine of EVQE is basically a short VQE optimisation run, improvements that have been proposed for VQE should also be applicable for the optimisation subroutines of EVQE. It would then be interesting to see whether, when using these improvements for both VQE and EVQE, EVQE still outperforms VQE on larger problem instances.

Finally, once access to quantum computing hardware has become more ubiquitous, scaling benchmarks over many more problem instances and problem sizes should be carried out to investigate whether the promise of evolving ansatz VQE methods holds true even on real quantum hardware. This would then also allow the usage of the runtime performance metric, as the overhead of quantum simulation would not be an issue any more. This would also enable a fair comparison of EVQE against classical optimisation heuristics.

A. Additional Figures

A.1. Example QAOA Ansatz



Figure A.1.: This figure shows one automatically generated QAOA ansatz layer for the JSSP instance shown in Figure 5.1.

A.2. Hyperparameter Optimisation Training Instances

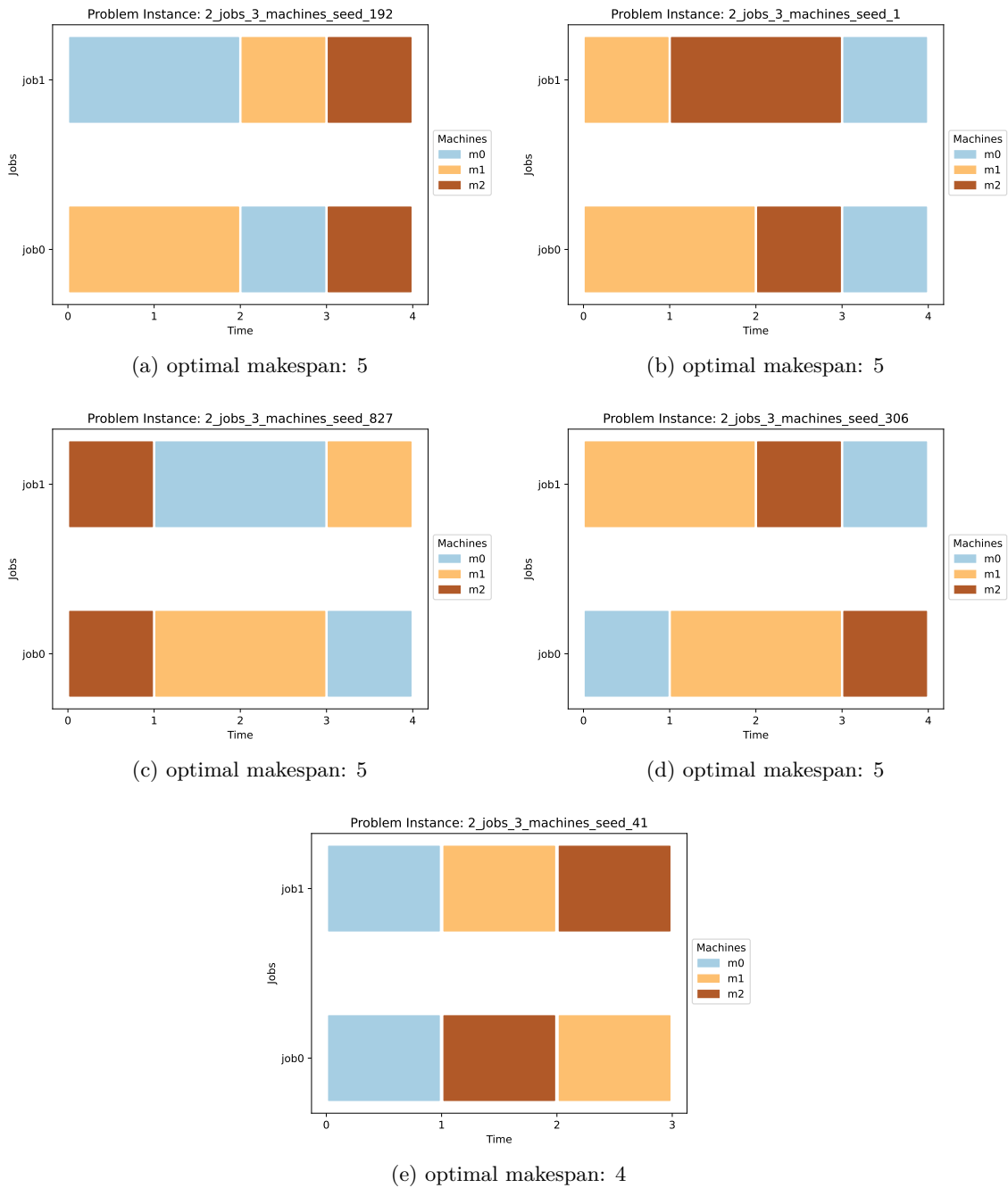


Figure A.2.: This figure shows all training JSSP instances used in the hyperparameter optimisation. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 12 qubits to be solved.

A.3. 12 Qubit Benchmarking Instances

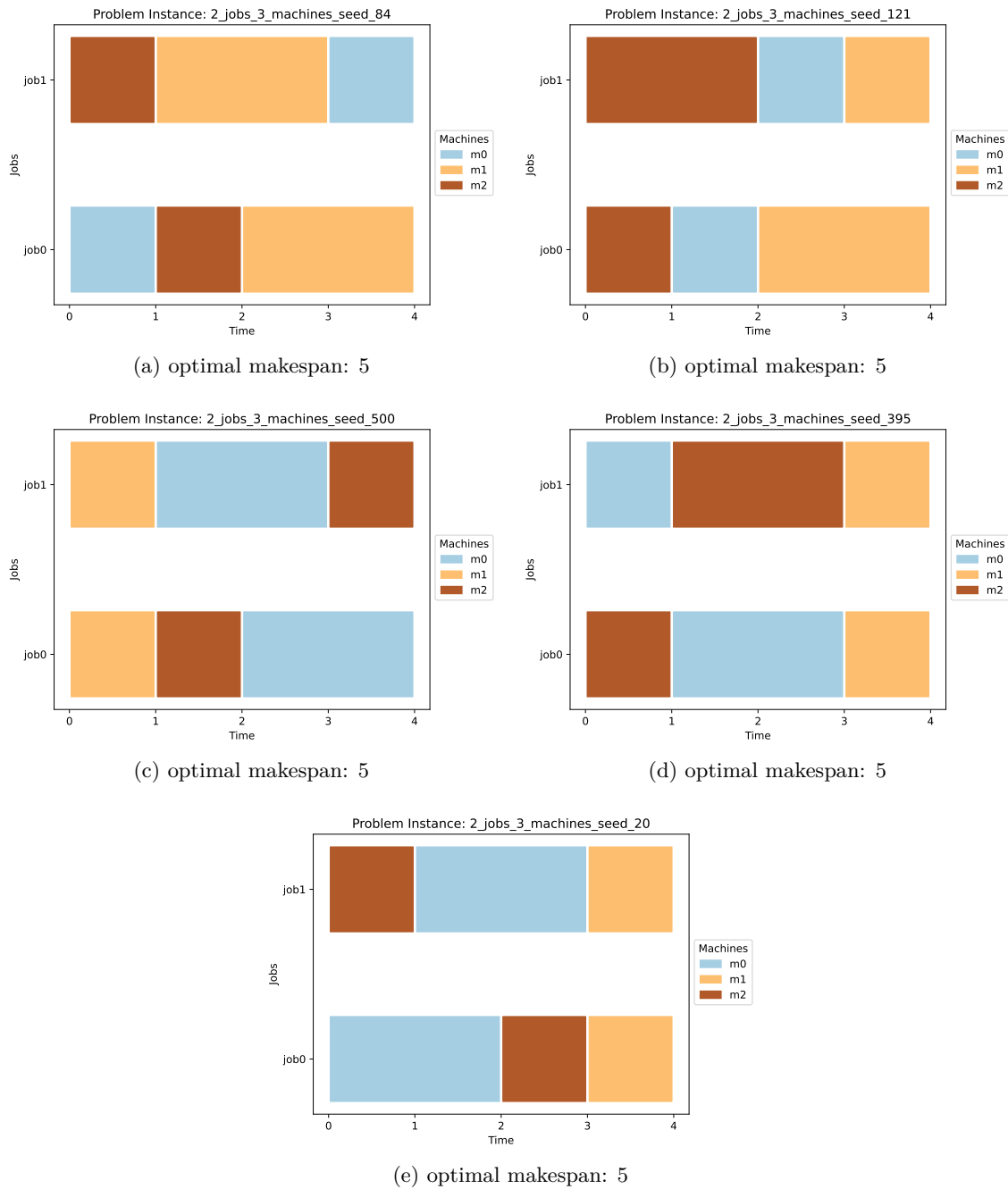


Figure A.3.: This figure shows all benchmarking JSSP instances used for benchmarking with 12 qubits. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 12 qubits to be solved.

A.4. 15 Qubit Benchmarking Instances

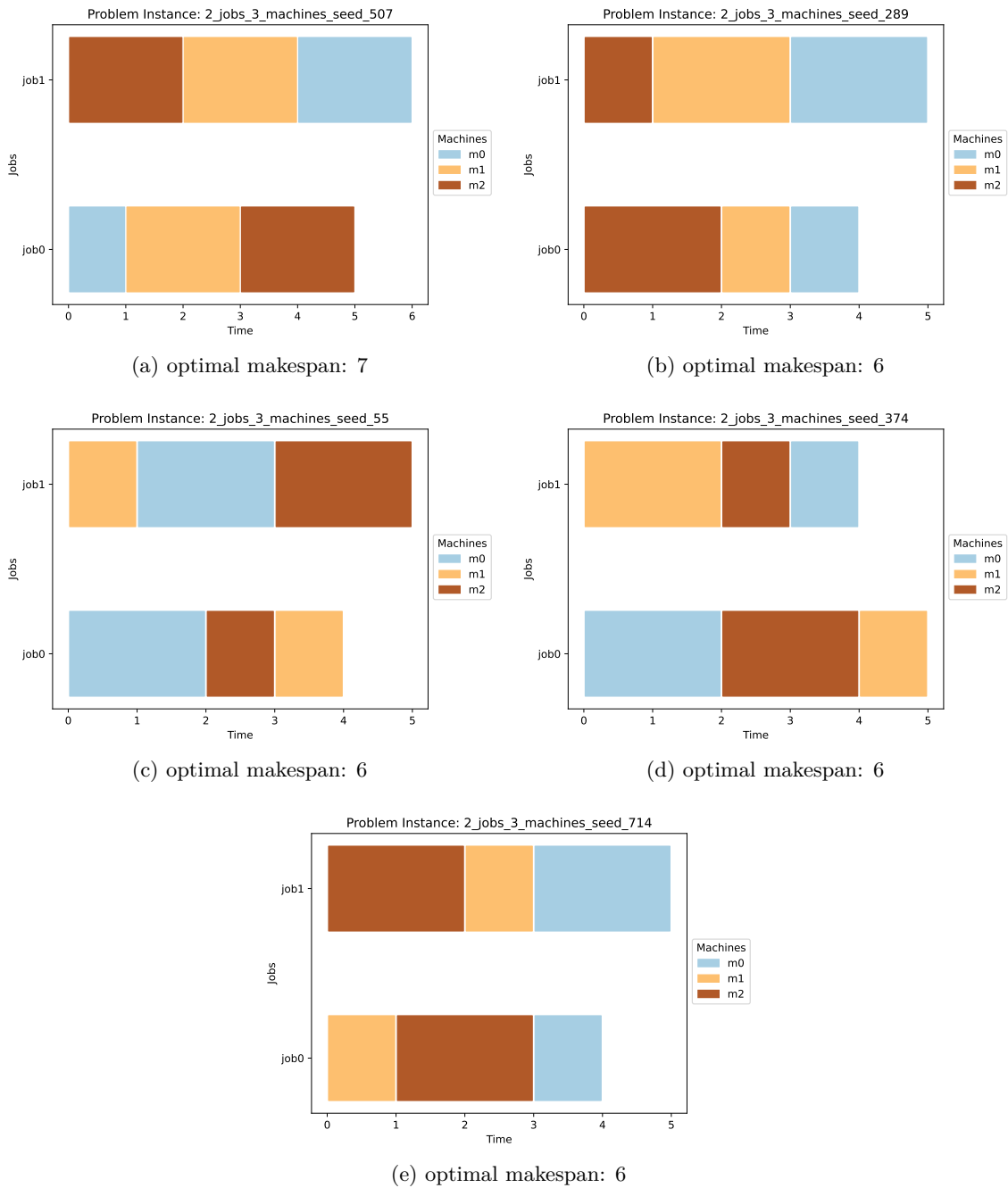


Figure A.4.: This figure shows all benchmarking JSSP instances used for benchmarking with 15 qubits. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 15 qubits to be solved.

A.5. 18 Qubit Benchmarking Instances

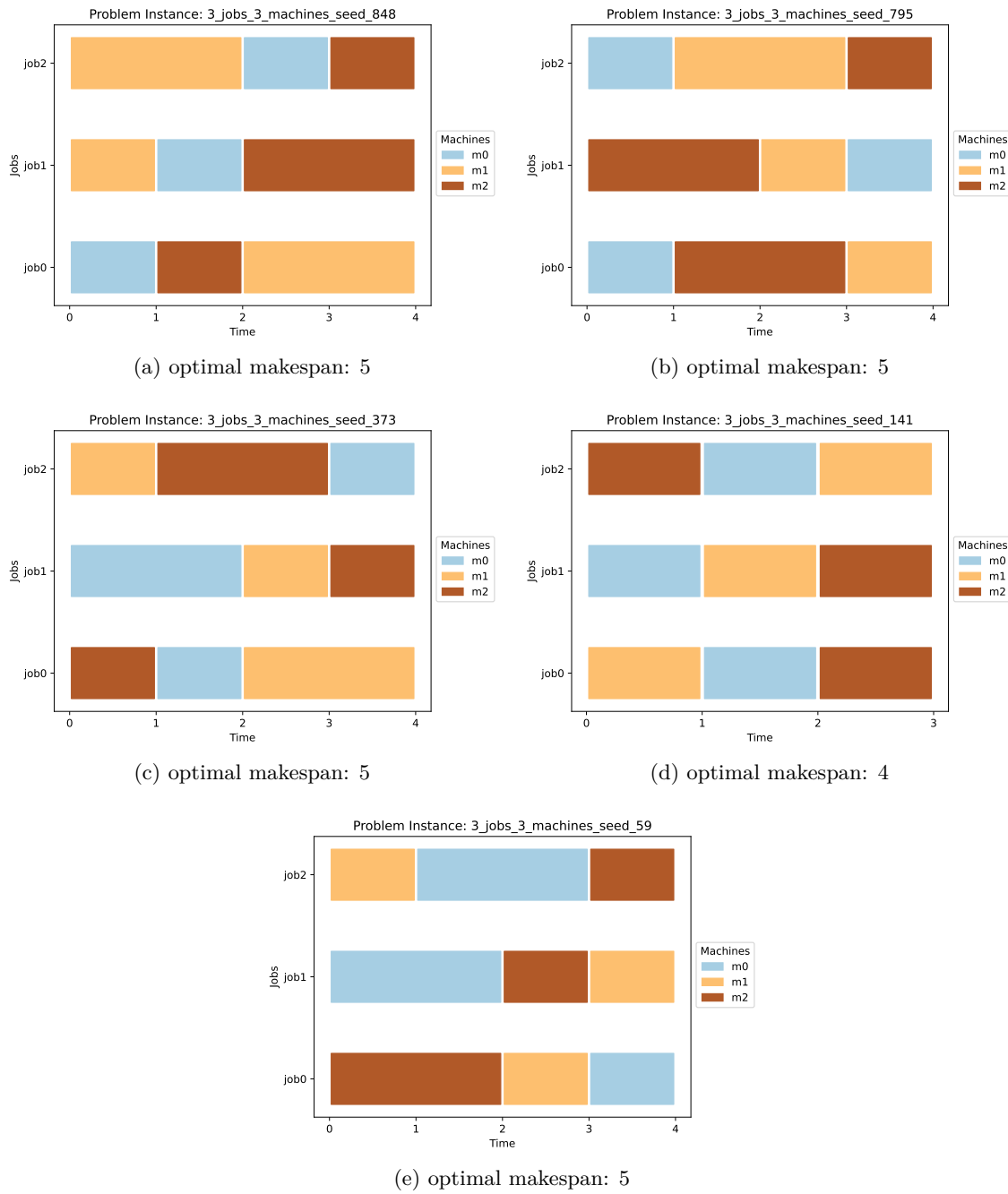


Figure A.5.: This figure shows all benchmarking JSSP instances used for benchmarking with 18 qubits. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 18 qubits to be solved.

A.6. 21 Qubit Benchmarking Instances

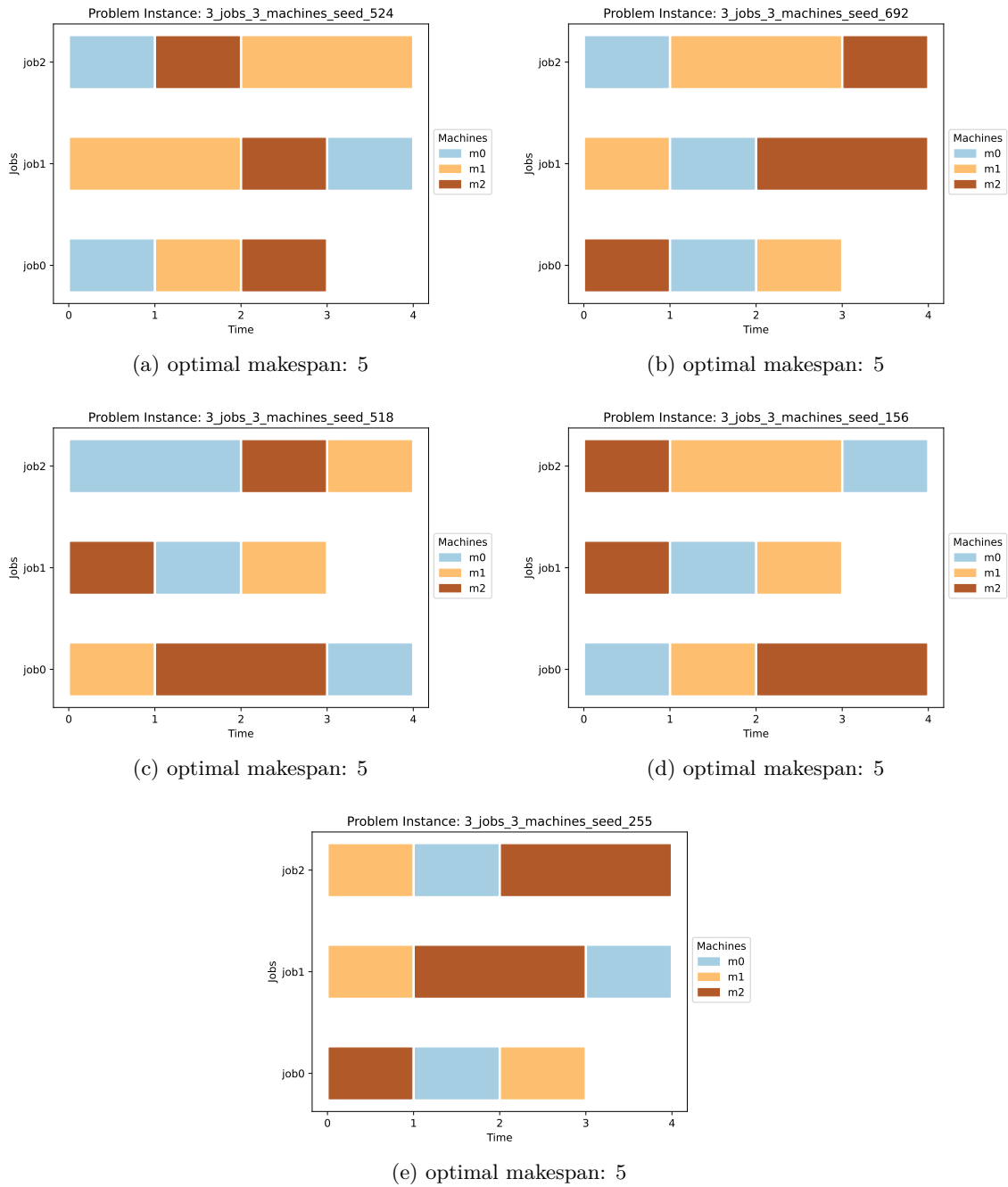


Figure A.6.: This figure shows all benchmarking JSSP instances used for benchmarking with 21 qubits. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 21 qubits to be solved.

B. Hyperparameter Values

B.1. QAOA Hyperparameter Values

QAOA Configurations	0	1	2	3	4
encoding penalty weight	492.8	307.4	241.5	408.3	536.1
precedence penalty weight	492.8	307.4	241.5	408.3	536.1
overlap penalty weight	429.8	307.4	241.5	408.3	536.1
early start term importance	0.23	0.33	0.25	0.02	0.06
perturbation	0.45	0.28	0.1	0.43	0.42
learning_rate	0.35	0.47	0.43	0.36	0.36
resamplings	1	1	2	2	2
trust_region	False	False	False	True	True
blocking	True	True	True	True	True
allowed_increase	414.2	223.4	300.1	359.6	355.2
ansatz layer repetitions	2	3	2	3	3

Figure B.1.: This table shows all hyperparameter values for all QAOA hyperparameter configurations found by SMAC3. The hyperparameter names are colour-coded to represent their origin. Light blue hyperparameters belong to the Hamiltonian encoding process, light yellow hyperparameters belong to SPSA, and light red hyperparameters belong to QAOA itself. The QAOA hyperparameter configuration, which we chose for our benchmarks, is highlighted in light green.

B.2. VQE Hyperparameter Values

Configuration	0	1	2	3	4	5	6	7
encoding penalty weight	600.4	714.7	681.8	838.8	677.3	840.7	697.2	704.6
precedence penalty weight	600.4	700.1	681.8	838.8	677.3	840.7	697.2	704.6
overlap penalty weight	600.4	714.7	681.8	793.1	677.3	805.7	697.2	704.6
early start term importance	0.48	0.45	0.16	0.17	0.18	0.15	0.06	0.37
perturbation	0.08	0.12	0.17	0.17	0.21	0.21	0.20	0.21
learning_rate	0.31	0.28	0.29	0.3	0.18	0.26	0.18	0.19
resamplings	3	1	2	4	2	2	2	3
trust_region	True	True	True	True	True	True	True	True
blocking	False	False	False	False	False	False	False	False
allowed_increase	-	-	-	-	-	-	-	-
ansatz layer repetitions	2	2	2	2	2	2	3	3

Figure B.2.: This table shows all hyperparameter values for all VQE hyperparameter configurations found by SMAC3. The hyperparameter names are colour-coded to represent their origin. Light blue hyperparameters belong to the Hamiltonian encoding process, light yellow hyperparameters belong to SPSA, and light red hyperparameters belong to VQE itself. The VQE hyperparameter configuration, which we chose for our benchmarks, is highlighted in light green.

B.3. EVQE Hyperparameter Values

Configuration	0	1	2	3	4	5	6	7	8
encoding penalty weight	459.4	318.6	231.0	193.6	971.7	820.8	858.3	288.5	111.1
precedence penalty weight	113.2	274.9	231.0	193.6	337.8	623.8	517.9	288.5	111.1
overlap penalty weight	386.4	318.6	231.0	193.6	598.5	588.4	621.8	288.5	111.1
early start term importance	0.33	0.19	0.19	0.45	0.18	0.30	0.29	0.43	0.34
maxiter	44	33	37	41	42	38	38	31	19
perturbation	0.35	0.35	0.36	0.32	0.37	0.50	0.44	0.24	0.37
learning_rate	0.48	0.43	0.48	0.40	0.48	0.47	0.47	0.34	0.13
resamplings	1	1	2	3	3	3	3	1	1
trust_region	True	True	True	True	True	True	True	False	False
blocking	False	False	False	False	False	False	False	False	False
allowed_increase	-	-	-	-	-	-	-	-	-
termination behaviour	1	2	3	4	4	6	6	3	3
last_avg	1	1	2	1	4	4	4	1	4
genetic distance	1	1	2	2	2	1	2	3	2
alpha penalty	0.28	0.15	0.04	0.11	0.31	0.39	0.36	0.18	0.20
beta penalty	0.09	0.02	0.07	0.03	0.35	0.34	0.35	0.11	0.21
likelihood of topological search	0.75	0.79	0.80	0.66	0.53	0.61	0.61	0.67	0.55
likelihood of parameter search	0.32	0.39	0.26	0.12	0.33	0.33	0.34	0.08	0.22
likelihood of layer removal	0.06	0.02	0.08	0.10	0.13	0.13	0.13	0.14	0.15
tournament size	3	2	3	3	3	3	3	2	3

Figure B.3.: This table shows all hyperparameter values for all EVQE hyperparameter configurations found by SMAC3. The hyperparameter names are colour-coded to represent their origin. Light blue hyperparameters belong to the Hamiltonian encoding process, light yellow hyperparameters belong to SPSA, and light red hyperparameters belong to EVQE itself. The EVQE hyperparameter configuration, which we chose for our benchmarks, is highlighted in light green.

List of Figures

2.1.	This figure shows a visualisation of the operation order in each job for an example JSSP instance, with 3 jobs on 3 machines.	3
2.2.	This figure shows a comparison of valid scheduling results with different makespans for the JSSP instance shown in Figure 2.1. The schedule shown in (a) is optimal. The schedule shown in (b) is suboptimal.	4
2.3.	This figure shows a quantum circuit consisting of a quantum register of two qubits (a), a classical register to store results (a), quantum gates (b), and measurement instructions (c).	5
2.4.	This figure shows a Bloch sphere representation of the state $ 0\rangle$	6
2.5.	This figure shows a qubit's state before (a) and after (b) applying a Pauli-X gate.	7
2.6.	Qubit state before (a) and after (b) applying a Hadamard gate.	8
2.7.	This figure shows the simulated measurement results (noiseless, 500 shots) of the circuit in Figure 2.3.	10
2.8.	This figure shows the general workflow of VQA algorithms, which is an iterative loop. In it, a quantum state $ \psi(\theta)\rangle$ is created based on the parameter values θ . This state is then evaluated using objective function O . Based on the resulting objective value, the classical optimiser improves the parameter values θ	12
2.9.	This figure shows a parameterised version of the circuit in Figure 2.3 that uses U3 gates.	12
2.10.	This figure shows the simulated measurement results (noiseless, 500 shots) of the circuit in Figure 2.9 for different parameter values. The parameter values that were applied are shown below the individual figures in the order $[\theta_1, \phi_1, \lambda_1, \theta_2, \phi_2, \lambda_2]$	13
2.11.	This table shows the eigenstates and eigenvalues of the example Hamiltonian (Equation 2.32) with the corresponding QUBO variable assignments.	15
2.12.	This table demonstrates the one-hot encoding for a variable $y \in \{0, 1, 2\}$	16
2.13.	This table demonstrates the domain wall encoding for a variable $y \in \{0, 1, 2, 3\}$, with $ $ visualising the domain wall. The variable bits are preceded and followed by an imagined bit in grey.	17
2.14.	This figure shows the general workflow of evolutionary algorithms, in which selection, variation, and replacement are used to, over many generations, improve the fitness of the individuals within the population.	22
3.1.	This figure shows the hardware-efficient VQA ansatz used in Amaro et al.'s comparison of VQA algorithms with two layer repetitions.	25
3.2.	This figure shows an EVQE individual with two circuit layers. The separation between layers is indicated by the grey barriers.	28

3.3.	This figure illustrates the gate block structure used in the MoG-VQE algorithm [CSU ⁺ 20]. Combinations of multiple such gate blocks form an ansatz.	29
3.4.	This figure illustrates the ansatz structure of the QNEAT algorithm. Taken from Giovagnoli et al.'s paper on QNEAT [GTMS23] (CC BY 4.0 License ¹). In the first layer of the individual, all possible gates are placed, showcasing the structure of a layer. In the other layers, not all gates have been placed.	29
4.1.	This figure shows a graphical representation of the EVQE algorithm's cyclical evolutionary optimisation procedure.	32
4.2.	This figure illustrates the differences between genes, genomes, and gene instances in EVQE [RHP ⁺ 19]. (a) and (b) show two different genes γ_s and γ_t , each representing a circuit layer, without fixed parameter values. (c) and (d) show the genomes of two different individuals, i and j . They each consist of two gene instances, highlighted by the grey boxes.	33
4.3.	This figure illustrates the stages of the basic random gene generation. (a): First, the qubits are either designated for a CU3 (red pentagon) or a U3 (blue pentagon) gate with a 50% chance each. Then the CU3s are randomly placed (b). Finally, the U3 gates are placed (c).	34
4.4.	This figure illustrates how a random gene is generated to follow the gene shown in (a) without placing redundant gates. As shown in (b), following an identity or U3 gate in (a), the qubits are directly designated as CU3 qubits (red pentagons). Following a CU3 gate in (a), the qubits are randomly designated as either U3 (blue pentagons) or CU3 qubits (c). After the random gate placement, no gate could be placed on qubit 3, leaving it with an identity gate, as shown in (d).	35
4.5.	This figure shows an individual before (a) and after (b) topological search was applied.	40
4.6.	This figure shows an individual before (a) and after (b) layer removal for one layer was applied.	40
5.1.	This figure shows a small JSSP problem instance using 2 jobs and 2 machines. Its minimum makespan is 3.	47
5.2.	This figure visualises the full energy landscape for the small JSSP instance. The red line denotes $w_{opt} = 100$. Any state with less energy than that is valid. All other states are invalid. It can be seen that the vast majority of states are invalid.	47
5.3.	This figure visualises the energy landscape for the small JSSP instance if all energies larger than $w_{opt} = 100$ are filtered out. The remaining energies are energies for fully valid states. The global minima at $x = 5$ and $x=80$ both have an identical energy of 22.9.	48
5.4.	This figure shows the solutions to the small JSSP instance with makespan 3. These correspond to the global minima at (a) $x = 5$ and (b) $x = 80$ of the QUBO's energy landscape. Both represent ideal solutions with makespan 3, differing only in their ordering of the jobs.	48
6.1.	This figure shows a problematic JSSP problem instance with 2 jobs and 3 machines.	49

6.2. This figure demonstrates the issues of the EVQE algorithm during an optimisation run for the problem instance shown in Figure 6.1. (a) shows the expectation value of the best individual for each generation. (b) shows the measurement result for the best individual found during the evolution. As it can be seen, EVQE was incapable of finding better states than the $|0\rangle^{\otimes n}$ state. 50

6.3. This table shows eigenstates with their eigenvalues (energy) for the Hamiltonian of the JSSP instance shown in Figure 6.1. Specifically, these are eigenstates with energies lower than the energy of the zero state $|0\rangle^{\otimes n}$, which need the fewest bit flips to be reached from the zero state. For reference, it also contains the energy of the zero state and the range of energies for all states, which only need up to three bitflips. 51

6.4. This figure shows two EVQE optimisation runs for the JSSP instance shown in Figure 6.1. (a) uses an initialisation approach with two random gene instances and random parameters for each initial individual. (b) uses an initialisation approach by prepending Hadamard gates to each individual's ansatz. (a) converges faster, presumably due to its ability to explore a range of good initial states. Note that the y-axis scales are different. (a) finds a state with an expectation value below 100 in generation 2, whereas (b) only finds such a state in generation 6. 52

6.5. This table shows selection probabilities for a population of five fictional individuals when using proportional selection as explained in Section 4.3.3. In particular, it shows the issue of low selection pressure if the fitness values do not differ greatly. See, for instance, the individual with fitness 30, which is only slightly likelier to be selected as a parent than the individual with fitness 32. 53

6.6. This figure illustrates the tournament selection process, with a tournament size of $k = 2$, where five parents are selected from a population of five individuals. The numbers represent the fitness values of the individuals. Five tournaments are held to select five parents. The fittest individual for each tournament that is selected as a parent, is highlighted in red. 53

6.7. This table shows selection probabilities for a population of five fictional individuals when using tournament selection with replacement and a tournament size of $k = 2$. When compared to the selection likelihoods for the proportional selection shown in Table 6.5, one can see that for small differences in fitness values, tournament selection still applies a reasonable selection pressure. . . . 54

List of Figures

7.1. This figure shows the optimisation history of an example run of the EVQE algorithm. Each point in the plot shows the best expectation value found during a generation of the evolutionary algorithm and the total number of expectation value evaluations used at the end of that generation. The red horizontal line shows the energy boundary for valid states E_{bval} , while the blue horizontal line shows the energy boundary for optimal states E_{bopt} . As can be seen in the graph, EVQE first provably found valid solutions in the third generation (highlighted in red), first provably found optimal solutions in the fifth generation (highlighted in blue), found its best expectation value in the 7th generation (highlighted in yellow), and terminated after 9 generations (highlighted in brown). This amounts to the following amounts of expectation value evaluations: $NEXP_{val} = 2387$, $NEXP_{opt} = 6102$, $NEXP_{best} = 7844$, $NEXP_{term} = 9810$ 58

7.2. These figures show the number of randomly generated problem instances, which satisfy our filtering criteria, distributed over the number of qubits that they require: (a) shows the number of problem instances with 6 operations; (b) shows the number of problem instances with 9 operations; and (c) shows the number of problem instances with 12 operations. 60

7.3. This figure shows the problem instance used for the optimisation run on real quantum hardware. Its optimal makespan is 4, and it needs 8 qubits to optimise with a makespan limit of 5. 71

8.1. This figure shows the average cost values for the best QAOA hyperparameter configurations found by SMAC3. Lower values are better for both the inverted state quality and the expectation value evaluations. The error bars show the 0.1 and 0.9 quantiles of the cost values, respectively. The triangles denote the absolute minimum and absolute maximum of the ISQ observed for each configuration. The individual configurations are numbered to ease referencing them. 74

8.2. This figure shows the average cost values for the best VQE hyperparameter configurations found by SMAC3 over all their optimisation runs. Lower values are better for both the inverted state quality and the expectation value evaluations. The error bars show the 0.1 and 0.9 quantiles of the cost values, respectively. The triangles denote the absolute minimum and absolute maximum of the ISQ observed for each configuration. The individual configurations are numbered to ease referencing them. 75

8.3. This figure shows the average cost values for the best EVQE hyperparameter configurations found by SMAC3 over all their optimisation runs. Lower values are better for both the inverted state quality and the expectation value evaluations. The error bars show the 0.1 and 0.9 quantiles of the cost values, respectively. The triangles denote the absolute minimum and absolute maximum of the ISQ observed for each configuration. The individual configurations are numbered to ease referencing them. 76

8.4. This figure shows the number of expectation value evaluations needed for the algorithms to terminate over multiple problem sizes. Since for each problem size, each algorithm was run 25 times, box plots are used to aggregate the individual results. The lower and upper edges of the boxes show the 0.25 and 0.75 quantiles, respectively. The horizontal line within the boxes shows the median, and the whiskers extending from the box show the minimum and maximum values. 77

8.5. These figures show one optimisation run for VQE (a), QAOA (b), and EVQE (c) for the third 18-qubit JSSP benchmarking instance shown in Figure A.5. The red line denotes the boundary value E_{bval} , and the blue line denotes the boundary value E_{bopt} . VQE finds optimal solutions, but its expectation values vary strongly between optimisation iterations. QAOA does not seem to find good solutions. It also experiences swings in expectation value. Furthermore, the long distances between reported expectation values indicate that SPSA is having difficulties finding good parameter value updates, as it rejects many update steps. 78

8.6. These figures show the likelihood of measuring a valid solution P_{val} (a) or an optimal solution P_{opt} (b) when measuring the best quantum state found by the VQAs over multiple problem sizes. Since for each problem size, each algorithm is run 25 times, box plots are used to aggregate the results. 80

8.7. These figures show the success rate for the VQAs over multiple problem sizes of finding quantum states in which valid (a) or optimal (b) solutions can be measured with a likelihood higher than 0.01. 80

8.8. These figures show how many expectation value evaluations the VQAs need to hit certain milestones during the optimisation process for multiple problem sizes. Since for each problem size, each algorithm was run 25 times, these values are aggregated using box plots. In (a), it is shown how many expectation value evaluations the VQAs need to reach expectation values below the boundary E_{bval} . In (b), it is shown how many expectation value evaluations the VQAs need to reach expectation values below the boundary E_{bopt} . In (c), it is shown how many expectation value evaluations the VQAs need to reach the quantum state with the lowest expectation value seen during the optimisation procedure. 82

8.9. These figures show the depth of the ansatz circuit over all problem sizes and algorithms, aggregated using box plots. The ansatz circuit depth is shown in (a), and the number of controlled gates is shown in (b). 83

8.10. These figures show how changing the population size of EVQE influences the performance of EVQE. To that end, they represent the results of running EVQE with the population sizes 5, 10, and 20 for each problem size. In (a), the computational effort needed by these EVQE variants over the problem sizes is shown. In (b), it is shown how many of the EVQE optimisation runs are successful at finding optimal solutions. Success is defined as measuring an optimal solution from the quantum state found by EVQE with a likelihood higher than 0.01. In (c), the depth of the best ansatz and in (d), the number of controlled gates in the best ansatz found by EVQE during each optimisation run is shown. 84

8.11.	These figures show how EVQE, VQE, and QAOA compare under varying shot noise scenarios. Shotless refers to the noise-free state vector simulation. In (a), it is shown how many expectation value evaluations the VQAs need to reach a quantum state that contains optimal solutions. In (b) it is shown how likely it is to measure an optimal solution from the quantum states found by the VQAs.	85
8.12.	These figures show the results for the EVQE optimisation run on IBM's <i>ibm_nazca</i> 127-qubit quantum computer. In (a), the best expectation value found during each generation is shown. The red horizontal line denotes the boundary value E_{bval} , and the blue horizontal line denotes the boundary value E_{bopt} . EVQE finds the best possible expectation value during the first generation. As a result, it cannot improve the expectation value further during subsequent generations. In (b), the measurement distribution of classical states, retrieved by measuring the ansatz of the best individual multiple times, is shown. For readability, the bitstrings retrieved by the measurement are referred to by their decimal representation. State 5 with the measurement probability of 73% translates to the schedule with the optimal makespan 4, shown in (c).	87
A.1.	This figure shows one automatically generated QAOA ansatz layer for the JSSP instance shown in Figure 5.1.	93
A.2.	This figure shows all training JSSP instances used in the hyperparameter optimisation. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 12 qubits to be solved.	94
A.3.	This figure shows all benchmarking JSSP instances used for benchmarking with 12 qubits. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 12 qubits to be solved.	95
A.4.	This figure shows all benchmarking JSSP instances used for benchmarking with 15 qubits. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 15 qubits to be solved.	96
A.5.	This figure shows all benchmarking JSSP instances used for benchmarking with 18 qubits. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 18 qubits to be solved.	97
A.6.	This figure shows all benchmarking JSSP instances used for benchmarking with 21 qubits. For each instance, the Hamiltonian is generated with a makespan limit that is 1 greater than their optimal makespan. In that configuration, each training instance needs 21 qubits to be solved.	98

- B.1. This table shows all hyperparameter values for all QAOA hyperparameter configurations found by SMAC3. The hyperparameter names are colour-coded to represent their origin. Light blue hyperparameters belong to the Hamiltonian encoding process, light yellow hyperparameters belong to SPSA, and light red hyperparameters belong to QAOA itself. The QAOA hyperparameter configuration, which we chose for our benchmarks, is highlighted in light green. 99
- B.2. This table shows all hyperparameter values for all VQE hyperparameter configurations found by SMAC3. The hyperparameter names are colour-coded to represent their origin. Light blue hyperparameters belong to the Hamiltonian encoding process, light yellow hyperparameters belong to SPSA, and light red hyperparameters belong to VQE itself. The VQE hyperparameter configuration, which we chose for our benchmarks, is highlighted in light green. 100
- B.3. This table shows all hyperparameter values for all EVQE hyperparameter configurations found by SMAC3. The hyperparameter names are colour-coded to represent their origin. Light blue hyperparameters belong to the Hamiltonian encoding process, light yellow hyperparameters belong to SPSA, and light red hyperparameters belong to EVQE itself. The EVQE hyperparameter configuration, which we chose for our benchmarks, is highlighted in light green. . . 101

Bibliography

- [AHY22] AJAGEKAR, Akshay ; HAMOUD, Kumail A. ; YOU, Fengqi: Hybrid Classical-Quantum Optimization Techniques for Solving Mixed-Integer Programming Problems in Production Scheduling. In: *IEEE Transactions on Quantum Engineering* 3 (2022), S. 1–16. <http://dx.doi.org/10.1109/TQE.2022.3187367>. – DOI 10.1109/TQE.2022.3187367
- [AK22] ANSCHUETZ, Eric R. ; KIANI, Bobak T.: Quantum variational algorithms are swamped with traps. In: *Nature Communications* 13 (2022), Nr. 1, 7760. <http://dx.doi.org/10.1038/s41467-022-35364-5>. – DOI 10.1038/s41467-022-35364-5. – ISSN 2041–1723
- [ARF⁺22] AMARO, David ; ROSENKRANZ, Matthias ; FITZPATRICK, Nathan ; HIRANO, Koji ; FIORENTINI, Mattia: A case study of variational quantum algorithms for a job shop scheduling problem. In: *EPJ Quantum Technology* 9 (2022), Nr. 1, 1–20. <http://dx.doi.org/10.1140/epjqt/s40507-022-00123-4>. – DOI 10.1140/epjqt/s40507-022-00123-4. – ISSN 2196–0763
- [AST09] ANSÓTEGUI, Carlos ; SELLMANN, Meinolf ; TIERNEY, Kevin: A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. Version: 2009. http://dx.doi.org/10.1007/978-3-642-04244-7_14. In: GENT, Ian P. (Hrsg.): *Principles and Practice of Constraint Programming - CP 2002* Bd. 5732. Berlin, Heidelberg : Springer Nature, 2009. – DOI 10.1007/978-3-642-04244-7_14. – ISBN 978-3-642-04243-0, S. 142–157
- [BBC⁺23] BESTUZHEVA, Ksenia ; BESANÇON, Mathieu ; CHEN, Wei-Kun ; CHMIELA, Antonia ; DONKIEWICZ, Tim ; VAN DOORNALEN, Jasper ; EIFLER, Leon ; GAUL, Oliver ; GAMRATH, Gerald ; GLEIXNER, Ambros ; GOTTWALD, Leona ; GRACZYK, Christoph ; HALBIG, Katrin ; HOEN, Alexander ; HOJNY, Christopher ; VAN DER HULST, Rolf ; KOCH, Thorsten ; LÜBBECKE, Marco ; MAHER, Stephen J. ; MATTER, Frederic ; MÜHMER, Erik ; MÜLLER, Benjamin ; PFETSCH, Marc E. ; REHFELDT, Daniel ; SCHLEIN, Steffan ; SCHLÖSSER, Franziska ; SERRANO, Felipe ; SHINANO, Yuji ; SOFRANAC, Boro ; TURNER, Mark ; VIGERSKE, Stefan ; WEGSCHEIDER, Fabian ; WELLNER, Philipp ; WENINGER, Dieter ; WITZIG, Jakob: Enabling Research through the SCIP Optimization Suite 8.0. In: *ACM Transactions on Mathematical Software* 49 (2023), Nr. 2, S. 1–21. <http://dx.doi.org/10.1145/3585516>. – DOI 10.1145/3585516. – ISSN 0098–3500
- [BC95] BARNES, J. W. ; CHAMBERS, JOHN B.: Solving the job shop scheduling problem with tabu search. In: *IIE Transactions* 27 (1995), Nr. 2, S. 257–263. <http://dx.doi.org/10.1080/07408179508936739>. – DOI 10.1080/07408179508936739. – ISSN 0740–817X

- [BCCS22] BERAHAS, Albert S. ; CAO, Liyuan ; CHOROMANSKI, Krzysztof ; SCHEINBERG, Katya: A Theoretical and Empirical Comparison of Gradient Approximations in Derivative-Free Optimization. In: *Foundations of Computational Mathematics* 22 (2022), Nr. 2, 507–560. <http://dx.doi.org/10.1007/s10208-021-09513-z>. – DOI 10.1007/s10208-021-09513-z. – ISSN 1615–3383
- [BCLK⁺22] BHARTI, Kishor ; CERVERA-LIERTA, Alba ; KYAW, Thi H. ; HAUG, Tobias ; ALPERIN-LEA, Sumner ; ANAND, Abhinav ; DEGROOTE, Matthias ; HEIMONEN, Hermann ; KOTTMANN, Jakob S. ; MENKE, Tim ; MOK, Wai-Keong ; SIM, Sukin ; KWEK, Leong-Chuan ; ASPURU-GUZIK, Alán: Noisy intermediate-scale quantum algorithms. In: *Reviews of Modern Physics* 94 (2022), Nr. 1. <http://dx.doi.org/10.1103/RevModPhys.94.015004>. – DOI 10.1103/RevModPhys.94.015004. – ISSN 0034–6861
- [BCV⁺23] BILKIS, M. ; CEREZO, M. ; VERDON, Guillaume ; COLES, Patrick J. ; CINICIO, Lukasz: A semi-agnostic ansatz with variable structure for variational quantum algorithms. In: *Quantum Machine Intelligence* 5 (2023), Nr. 2, 1–22. <http://dx.doi.org/10.1007/s42484-023-00132-1>. – DOI 10.1007/s42484-023-00132-1. – ISSN 2524–4914
- [BD95] B. L. MILLER ; D. GOLDBERG: Genetic Algorithms, Tournament Selection, and the Effects of Noise. In: *Complex Systems* (1995)
- [BGK⁺95] BARR, Richard S. ; GOLDEN, Bruce L. ; KELLY, James P. ; RESENDE, Mauricio G. C. ; STEWART, William R.: Designing and reporting on computational experiments with heuristic methods. In: *Journal of Heuristics* 1 (1995), Nr. 1, 9–32. <http://dx.doi.org/10.1007/BF02430363>. – DOI 10.1007/BF02430363. – ISSN 1572–9397
- [BM85] BARKER, Jeffrey R. ; MCMAHON, Graham B.: Scheduling the General Job-Shop. In: *Management Science* 31 (1985), Nr. 5, S. 594–598. <http://dx.doi.org/10.1287/mnsc.31.5.594>. – DOI 10.1287/mnsc.31.5.594. – ISSN 0025–1909
- [BMWV⁺23] BONET-MONROIG, Xavier ; WANG, Hao ; VERMETTEN, Diederick ; SENJEAN, Bruno ; MOUSSA, Charles ; BÄCK, Thomas ; DUNJKO, Vedran ; O’BRIEN, Thomas E.: Performance comparison of optimization methods on variational quantum algorithms. In: *Physical Review A* 107 (2023), Nr. 3. <http://dx.doi.org/10.1103/PhysRevA.107.032407>. – DOI 10.1103/PhysRevA.107.032407. – ISSN 2469–9926
- [BNR⁺20] BARKOUTSOS, Panagiotis K. ; NANNICINI, Giacomo ; ROBERT, Anton ; TAVERNELLI, Ivano ; WOERNER, Stefan: Improving Variational Quantum Optimization using CVaR. In: *Quantum* 4 (2020), 256. <http://dx.doi.org/10.22331/q-2020-04-20-256>. – DOI 10.22331/q-2020-04-20-256
- [BOM15] BRABAZON, Anthony ; O’NEILL, Michael ; MCGARRAGHY, Seán: *Natural Computing Algorithms*. 1st ed. 2015. Berlin, Heidelberg : Springer Berlin Heidelberg, 2015 (Natural Computing Series). <http://dx.doi.org/10.1007/>

978-3-662-43631-8. <http://dx.doi.org/10.1007/978-3-662-43631-8>. – ISBN 9783662436318

- [Bur17] BURTON, VIRGIL L., III: Job Shop. In: *Encyclopedia of Small Business* Bd. 2. Farmington Hills, MI : Gale, 2017, S. 639–641
- [CAB⁺21] CERESO, M. ; ARRASMITH, Andrew ; BABBUSH, Ryan ; BENJAMIN, Simon C. ; ENDO, Suguru ; FUJII, Keisuke ; MCCLEAN, Jarrod R. ; MITARAI, Kosuke ; YUAN, Xiao ; CINCIO, Lukasz ; COLES, Patrick J.: Variational quantum algorithms. In: *Nature Reviews Physics* 3 (2021), Nr. 9, S. 625–644. <http://dx.doi.org/10.1038/s42254-021-00348-9>. – DOI 10.1038/s42254-021-00348-9
- [ÇB15] ÇALIŞ, Banu ; BULKAN, Serol: A research survey: review of AI solution strategies of job shop scheduling problem. In: *Journal of Intelligent Manufacturing* 26 (2015), Nr. 5, 961–973. <http://dx.doi.org/10.1007/s10845-013-0837-8>. – DOI 10.1007/s10845-013-0837-8. – ISSN 1572-8145
- [CFC22] CARUGNO, Costantino ; FERRARI DACREMA, Maurizio ; CREMONESI, Paolo: Evaluating the job shop scheduling problem on a D-wave quantum annealer. In: *Scientific Reports* 12 (2022), Nr. 1, 6539. <http://dx.doi.org/10.1038/s41598-022-10169-0>. – DOI 10.1038/s41598-022-10169-0. – ISSN 2045-2322
- [CS20] CHAUHAN, Amit K. ; SANADHYA, Somitra K.: Quantum resource estimates of grover’s key search on aria. In: *Security, Privacy, and Applied Cryptography Engineering: 10th International Conference, SPACE 2020, Kolkata, India, December 17–21, 2020, Proceedings 10*, 2020, S. 238–258
- [CSU⁺20] CHIVILIKHIN, D. ; SAMARIN, A. ; ULYANTSEV, V. ; IORSH, I. ; OGANOV, A. R. ; KYRIENKO, O.: *MoG-VQE: Multiobjective genetic variational quantum eigensolver*. <https://arxiv.org/pdf/2007.04424.pdf>. Version: 2020
- [CWMH20] CLAUDINO, Daniel ; WRIGHT, Jerimiah ; MCCASKEY, Alexander J. ; HUMBLE, Travis S.: Benchmarking Adaptive Variational Quantum Eigensolvers. In: *Frontiers in Chemistry* 8 (2020), 606863. <http://dx.doi.org/10.3389/fchem.2020.606863>. – DOI 10.3389/fchem.2020.606863. – ISSN 2296-2646
- [ES15] EIBEN, A.E ; SMITH, James E.: *Introduction to Evolutionary Computing*. 2nd ed. 2015. Berlin, Heidelberg : Springer Berlin Heidelberg, 2015 (Natural Computing Series). <http://dx.doi.org/10.1007/978-3-662-44874-8>. <http://dx.doi.org/10.1007/978-3-662-44874-8>. – ISBN 9783662448748
- [FB91] FALKENAUER, E. ; BOUFFOUX, S.: A genetic algorithm for job shop. In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. Los Alamitos, Calif. and Piscataway, NJ : IEEE Computer Society Press and Order from IEEE Service Center, 1991. – ISBN 0-8186-2163-X, S. 824–829
- [FGG14] FARHI, Edward ; GOLDSTONE, Jeffrey ; GUTMANN, Sam: A Quantum Approximate Optimization Algorithm. In: *MIT-CTP* (2014). <https://arxiv.org/pdf/1411.4028.pdf>

- [FGS⁺94] FINNILA, A. B. ; GOMEZ, M. A. ; SEBENIK, C. ; STENSON, C. ; DOLL, J. D.: Quantum annealing: A new method for minimizing multidimensional functions. In: *Chemical Physics Letters* 219 (1994), Nr. 5-6, 343–348. [http://dx.doi.org/10.1016/0009-2614\(94\)00117-0](http://dx.doi.org/10.1016/0009-2614(94)00117-0). – DOI 10.1016/0009-2614(94)00117-0. – ISSN 00092614
- [GGB⁺23] GRIMSLEY, Harper R. ; BARRON, George S. ; BARNES, Edwin ; ECONOMOU, Sophia E. ; MAYHALL, Nicholas J.: Adaptive, problem-tailored variational quantum eigensolver mitigates rough parameter landscapes and barren plateaus. In: *npj Quantum Information* 9 (2023), Nr. 1, 1–8. <http://dx.doi.org/10.1038/s41534-023-00681-0>. – DOI 10.1038/s41534-023-00681-0
- [GEBM19] GRIMSLEY, Harper R. ; ECONOMOU, Sophia E. ; BARNES, Edwin ; MAYHALL, Nicholas J.: An adaptive variational algorithm for exact molecular simulations on a quantum computer. In: *Nature Communications* 10 (2019), Nr. 1, 3007. <http://dx.doi.org/10.1038/s41467-019-10988-2>. – DOI 10.1038/s41467-019-10988-2. – ISSN 2041-1723
- [GJS76] GAREY, M. R. ; JOHNSON, D. S. ; SETHI, Ravi: The Complexity of Flowshop and Jobshop Scheduling. In: *Mathematics of Operations Research* 1 (1976), Nr. 2, S. 117–129. <http://dx.doi.org/10.1287/moor.1.2.117>. – DOI 10.1287/moor.1.2.117. – ISSN 0364-765X
- [Gro96] GROVER, Lov K.: A fast quantum mechanical algorithm for database search. In: MILLER, Gary L. (Hrsg.): *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*. New York : ACM, 1996. – ISBN 0897917855, S. 212–219
- [GS06] GUPTA, Amit K. ; SIVAKUMAR, Appa I.: Job shop scheduling techniques in semiconductor manufacturing. In: *The International Journal of Advanced Manufacturing Technology* 27 (2006), Nr. 11-12, 1163–1169. <http://dx.doi.org/10.1007/s00170-004-2296-z>. – DOI 10.1007/s00170-004-2296-z. – ISSN 1433-3015
- [GTMS23] GIOVAGNOLI, Alessandro ; TRESP, Volker ; MA, Yunpu ; SCHUBERT, Matthias: QNEAT: Natural Evolution of Variational Quantum Circuit Architecture. In: SILVA, Sara (Hrsg.) ; PAQUETE, Luís (Hrsg.): *GECCO'23 companion*. New York : The Association for Computing Machinery, op. 2023. – ISBN 9798400701207, S. 647–650
- [Gur24] GUROBI OPTIMIZATION, LLC: *Gurobi Optimizer Reference Manual*. <https://www.gurobi.com>. Version: 2024
- [GZCW21] GACON, Julien ; ZOUFAL, Christa ; CARLEO, Giuseppe ; WOERNER, Stefan: Simultaneous Perturbation Stochastic Approximation of the Quantum Fisher Information. In: *Quantum* 5 (2021), 567. <http://dx.doi.org/10.22331/q-2021-10-20-567>. – DOI 10.22331/q-2021-10-20-567

- [HHLB11] HUTTER, Frank ; HOOS, Holger H. ; LEYTON-BROWN, Kevin: Sequential Model-Based Optimization for General Algorithm Configuration. In: COELLO COELLO, Carlos A. (Hrsg.): *Learning and Intelligent Optimization*. Berlin, Heidelberg : Springer Berlin Heidelberg and Imprint: Springer, 2011 (Theoretical Computer Science and General Issues). – ISBN 978-3-642-25566-3, 507–523
- [Hom22] HOMEISTER, Matthias: *Quantum Computing verstehen: Grundlagen - Anwendungen - Perspektiven*. 6., erweiterte und überarbeitete Auflage. Wiesbaden, Germany : Springer Vieweg, 2022 (Computational Intelligence). <http://dx.doi.org/10.1007/978-3-658-36434-2>. <http://dx.doi.org/10.1007/978-3-658-36434-2>. – ISBN 9783658364342
- [HSCC22] HOLMES, Zoë ; SHARMA, Kunal ; CEREZO, M. ; COLES, Patrick J.: Connecting Ansatz Expressibility to Gradient Magnitudes and Barren Plateaus. In: *PRX Quantum* 3 (2022), Nr. 1, S. 010313. <http://dx.doi.org/10.1103/PRXQuantum.3.010313>. – DOI 10.1103/PRXQuantum.3.010313
- [IBM22] IBM: *User's Manual for CPLEX*. <https://www.ibm.com/docs/en/icos/22.1.1?topic=optimizers-users-manual-cplex>. Version: 2022
- [JC23] JIANG, Jehn-Ruey ; CHU, Chun-Wei: Classifying and Benchmarking Quantum Annealing Algorithms Based on Quadratic Unconstrained Binary Optimization for Solving NP-Hard Problems. In: *IEEE Access* 11 (2023), S. 104165–104178. <http://dx.doi.org/10.1109/ACCESS.2023.3318206>. – DOI 10.1109/ACCESS.2023.3318206
- [JM+98] JAIN, Anant S. ; MEERAN, Sheik u.a.: A state-of-the-art review of job-shop scheduling techniques. In: *Workin paper, Department of applied physics, electronic and mechanical engineering. University of Dundee, Dundee, Scotland* (1998)
- [JNRV20] JAQUES, Samuel ; NAEHRIG, Michael ; ROETTELER, Martin ; VIRIDIA, Fernando: Implementing Grover oracles for quantum key search on AES and LowMC. In: *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, 2020, S. 280–310
- [Juk11] JUKNA, Stasys: *Extremal combinatorics: With applications in computer science*. 2nd ed. Heidelberg and New York : Springer, 2011 (Texts in theoretical computer science). <http://dx.doi.org/10.1007/978-3-642-17364-6>. <http://dx.doi.org/10.1007/978-3-642-17364-6>. – ISBN 978-3-642-17363-9
- [KB16] KU, Wen-Yang ; BECK, J. C.: Mixed Integer Programming models for job shop scheduling: A computational analysis. In: *Computers & Operations Research* 73 (2016), 165–173. <http://dx.doi.org/10.1016/j.cor.2016.04.006>. – DOI 10.1016/j.cor.2016.04.006. – ISSN 0305-0548

- [KB22] KOCZOR, Bálint ; BENJAMIN, Simon C.: Quantum natural gradient generalized to noisy and nonunitary circuits. In: *Physical Review A* 106 (2022), Nr. 6. <http://dx.doi.org/10.1103/PhysRevA.106.062416>. – DOI 10.1103/PhysRevA.106.062416. – ISSN 2469–9926
- [KN98] KADOWAKI, Tadashi ; NISHIMORI, Hidetoshi: Quantum annealing in the transverse Ising model. In: *Physical Review E* 58 (1998), Nr. 5, 5355–5363. <http://dx.doi.org/10.1103/PhysRevE.58.5355>. – DOI 10.1103/PhysRevE.58.5355
- [KPS⁺23] KUROWSKI, Krzysztof ; PECYNA, Tomasz ; SLYSZ, Mateusz ; RÓŻYCKI, Rafał ; WALIGÓRA, Grzegorz ; W GLARZ, Jan: Application of quantum approximate optimization algorithm to job shop scheduling problem. In: *European Journal of Operational Research* 310 (2023), Nr. 2, 518–528. <http://dx.doi.org/10.1016/j.ejor.2023.03.013>. – DOI 10.1016/j.ejor.2023.03.013. – ISSN 0377–2217
- [KWS⁺20] KUROWSKI, Krzysztof ; W GLARZ, Jan ; SUBOCZ, Marek ; RÓŻYCKI, Rafał ; WALIGÓRA, Grzegorz: Hybrid Quantum Annealing Heuristic Method for Solving Job Shop Scheduling Problem. Version:2020. http://dx.doi.org/10.1007/978-3-030-50433-5_39. In: KRZHIZHANOVSKAYA, Valeria V. (Hrsg.) ; ZÁVODSZKY, Gábor (Hrsg.) ; LEES, Michael H. (Hrsg.) ; DONGARRA, Jack (Hrsg.) ; SLOOT, Peter (Hrsg.) ; BRISSOS, Sérgio (Hrsg.) ; TEIXEIRA, João (Hrsg.): *Computational Science - ICCS 2020* Bd. 12142. Cham : Springer, 2020. – DOI 10.1007/978-3-030-50433-5_39. – ISBN 978-3-030-50432-8, S. 502–515
- [LHWK21] LIAQAIT, Raja A. ; HAMID, Shermeen ; WARSI, Salman S. ; KHALID, Azfar: A Critical Analysis of Job Shop Scheduling in Context of Industry 4.0. In: *Sustainability* 13 (2021), Nr. 14, 7684. <http://dx.doi.org/10.3390/su13147684>. – DOI 10.3390/su13147684. – ISSN 2071–1050
- [LKB77] LENSTRA, J. K. ; KAN, A. H. G. R. ; BRUCKER, P.: Complexity of Machine Scheduling Problems. Version:1977. [http://dx.doi.org/10.1016/S0167-5060\(08\)70743-X](http://dx.doi.org/10.1016/S0167-5060(08)70743-X). In: HAMMER, P. L. (Hrsg.) ; JOHNSON, E. L. (Hrsg.) ; KORTE, B. H. (Hrsg.) ; NEMHAUSER, G. L. (Hrsg.): *Studies in Integer Programming* Bd. 1. Elsevier, 1977. – DOI 10.1016/S0167-5060(08)70743-X, 343–362
- [LMW19] LARSON, Jeffrey ; MENICKELLY, Matt ; WILD, Stefan M.: Derivative-free optimization methods. In: *Acta Numerica* 28 (2019), 287–404. <http://dx.doi.org/10.1017/S0962492919000060>. – DOI 10.1017/S0962492919000060. – ISSN 1474–0508
- [Loc22] LOCKWOOD, Owen: *An Empirical Review of Optimization Techniques for Quantum Variational Circuits*. <http://arxiv.org/pdf/2202.01389>. Version: 2022
- [Lom65] LOMNICKI, Z. A.: A “Branch-and-Bound” Algorithm for the Exact Solution of the Three-Machine Scheduling Problem. In: *Journal of the Operational*

- Research Society* 16 (1965), Nr. 1, 89–100. <http://dx.doi.org/10.1057/jors.1965.7>. – DOI 10.1057/jors.1965.7. – ISSN 1476–9360
- [Luc14] LUCAS, Andrew: Ising formulations of many NP problems. In: *Frontiers in Physics* 2 (2014), 74887. <http://dx.doi.org/10.3389/fphy.2014.00005>. – DOI 10.3389/fphy.2014.00005. – ISSN 2296–424X
- [Man60] MANNE, Alan S.: On the Job-Shop Scheduling Problem. In: *Operations Research* 8 (1960), Nr. 2, S. 219–223. <http://dx.doi.org/10.1287/opre.8.2.219>. – DOI 10.1287/opre.8.2.219. – ISSN 0030–364X
- [MFP⁺22] MIHÁLIKOVÁ, Ivana ; FRIÁK, Martin ; PIVOLUSKA, Matej ; PLESCH, Martin ; SAIP, Martin ; ŠOB, Mojmir: Best-Practice Aspects of Quantum-Computer Calculations: A Case Study of the Hydrogen Molecule. In: *Molecules* 27 (2022), Nr. 3. <http://dx.doi.org/10.3390/molecules27030597>. – DOI 10.3390/molecules27030597. – ISSN 1420–3049
- [MKM⁺22] MARIUS LINDAUER ; KATHARINA EGGENSBERGER ; MATTHIAS FEURER ; ANDRÉ BIEDENKAPP ; DIFAN DENG ; CAROLIN BENJAMINS ; TIM RUHKOPF ; RENÉ SASS ; FRANK HUTTER: SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. In: *Journal of Machine Learning Research* 23 (2022), Nr. 54, 1–9. <https://www.jmlr.org/papers/v23/21-0888.html>. – ISSN 1533–7928
- [Nan19] NANNICINI, Giacomo: Performance of hybrid quantum-classical variational heuristics for combinatorial optimization. In: *Physical review. E* 99 (2019), Nr. 1-1, S. 013304. <http://dx.doi.org/10.1103/PhysRevE.99.013304>. – DOI 10.1103/PhysRevE.99.013304
- [NW06] NOCEDAL, Jorge ; WRIGHT, Stephen J.: *Numerical Optimization*. Second edition. New York, NY : Springer Science+Business Media, LLC. and Springer e-books, 2006 (Mathematics and Statistics (Springer-11649)). <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6315051>. – ISBN 9780387400655
- [PJSPP21] PELLOW-JARMAN, Aidan ; SINAYSKIY, Ilya ; PILLAY, Anban ; PETRUCCIONE, Francesco: A comparison of various classical optimizers for a variational quantum linear solver. In: *Quantum Information Processing* 20 (2021), Nr. 6, 1–14. <http://dx.doi.org/10.1007/s11128-021-03140-x>. – DOI 10.1007/s11128-021-03140-x. – ISSN 1573–1332
- [PMS⁺14] PERUZZO, Alberto ; MCCLEAN, Jarrod ; SHADBOLT, Peter ; YUNG, Man-Hong ; ZHOU, Xiao-Qi ; LOVE, Peter J. ; ASPURU-GUZIĆ, Alán ; O'BRIEN, Jeremy L.: A variational eigenvalue solver on a photonic quantum processor. In: *Nature communications* 5 (2014), 4213. <http://dx.doi.org/10.1038/ncomms5213>. – DOI 10.1038/ncomms5213
- [PSR21] PLEWA, Julia ; SIENKO, Joanna ; RYCERZ, Katarzyna: Variational Algorithms for Workflow Scheduling Problem in Gate-Based Quantum Devices. In: *COMPUTING AND INFORMATICS* 40 (2021), Nr. 4, 897–929. http://dx.doi.org/10.31577/cai_2021_4_897. – DOI 10.31577/cai_2021_4_897

- [Qis23] QISKIT CONTRIBUTORS: *Qiskit: An Open-source Framework for Quantum Computing*. <http://dx.doi.org/10.5281/zenodo.2573505>. Version: 2023
- [RDD23] RAJWAR, Kanchan ; DEEP, Kusum ; DAS, Swagatam: An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. In: *Artificial Intelligence Review* 56 (2023), Nr. 11, 1–71. <http://dx.doi.org/10.1007/s10462-023-10470-y>. – DOI 10.1007/s10462-023-10470-y. – ISSN 1573-7462
- [RHP⁺19] RATTEW, Arthur G. ; HU, Shaohan ; PISTOIA, Marco ; CHEN, Richard ; WOOD, Steve: *A Domain-agnostic, Noise-resistant, Hardware-efficient Evolutionary Variational Quantum Eigensolver*. <https://arxiv.org/pdf/1910.09694.pdf>. Version: 2019
- [RJ16] RABELO, Luis C. ; JONES, Albert: Job Shop Scheduling. Version: 2016. http://dx.doi.org/10.1007/978-1-4419-1153-7_490. In: GASS, Saul I. (Hrsg.) ; FU, Michael C. (Hrsg.): *Encyclopedia of operations research and management science*. New York, New York and Boston, Massachusetts : Springer Berlin Heidelberg and Credo Reference, 2016. – DOI 10.1007/978-1-4419-1153-7_490. – ISBN 978-1-4419-1137-7, S. 817–830
- [SEBB22] STORK, Jörg ; EIBEN, A. E. ; BARTZ-BEIELSTEIN, Thomas: A new taxonomy of global optimization algorithms. In: *Natural Computing* 21 (2022), Nr. 2, 219–242. <http://dx.doi.org/10.1007/s11047-020-09820-4>. – DOI 10.1007/s11047-020-09820-4. – ISSN 1572-9796
- [SMM23] SINGH, Harshdeep ; MAJUMDER, Sonjoy ; MISHRA, Sabyashachi: Benchmarking of different optimizers in the variational quantum algorithms for applications in quantum chemistry. In: *The Journal of Chemical Physics* 159 (2023), Nr. 4. <http://dx.doi.org/10.1063/5.0161057>. – DOI 10.1063/5.0161057. – ISSN 0021-9606
- [Spa98] SPALL, J. C.: An overview of the simultaneous perturbation method for efficient optimization. In: *Johns Hopkins APL technical digest* 19 (1998), Nr. 4, S. 482. – ISSN 0270-5214
- [SWGA23a] SCHWORM, Philipp ; WU, Xiangqian ; GLATT, Moritz ; AURICH, Jan C.: Responsiveness to sudden disturbances in manufacturing through dynamic job shop scheduling using Quantum Annealing. In: *Procedia CIRP* 120 (2023), 511–516. <http://dx.doi.org/10.1016/j.procir.2023.09.028>. – DOI 10.1016/j.procir.2023.09.028. – ISSN 2212-8271
- [SWGA23b] SCHWORM, Philipp ; WU, Xiangqian ; GLATT, Moritz ; AURICH, Jan C.: Solving flexible job shop scheduling problems in manufacturing with Quantum Annealing. In: *Production Engineering* 17 (2023), Nr. 1, 105–115. <http://dx.doi.org/10.1007/s11740-022-01145-8>. – DOI 10.1007/s11740-022-01145-8. – ISSN 1863-7353
- [SWK⁺24] SCHWORM, Philipp ; WU, Xiangqian ; KLAR, Matthias ; GLATT, Moritz ; AURICH, Jan C.: Multi-objective Quantum Annealing approach for solving

- flexible job shop scheduling in manufacturing. In: *Journal of Manufacturing Systems* 72 (2024), 142–153. <http://dx.doi.org/10.1016/j.jmsy.2023.11.015>. – DOI 10.1016/j.jmsy.2023.11.015. – ISSN 0278–6125
- [Tai93] TAILLARD, E.: Benchmarks for basic scheduling problems. In: *European Journal of Operational Research* 64 (1993), Nr. 2, 278–285. [http://dx.doi.org/10.1016/0377-2217\(93\)90182-M](http://dx.doi.org/10.1016/0377-2217(93)90182-M). – DOI 10.1016/0377-2217(93)90182-M. – ISSN 0377-2217
- [Tal09] TALBI, El-Ghazali: *Metaheuristics: From design to implementation*. Hoboken, N.J. : John Wiley & Sons, 2009. – ISBN 9780470496909
- [TAM⁺22] TASSEFF, Byron ; ALBASH, Tameem ; MORRELL, Zachary ; VUFFRAY, Marc ; LOKHOV, Andrey Y. ; MISRA, Sidhant ; COFFRIN, Carleton: On the Emerging Potential of Quantum Annealing Hardware for Combinatorial Optimization. In: *LA-UR-22-* (2022). <http://arxiv.org/pdf/2210.04291>
- [TCC⁺22] TILLY, Jules ; CHEN, Hongxiang ; CAO, Shuxiang ; PICOZZI, Dario ; SETIA, Kanav ; LI, Ying ; GRANT, Edward ; WOSSNIG, Leonard ; RUNGGER, Ivan ; BOOTH, George H. ; TENNYSON, Jonathan: The Variational Quantum Eigensolver: A review of methods and best practices. In: *Physics Reports* 986 (2022), S. 1–128
- [TSB⁺21] TANG, Ho L. ; SHKOLNIKOV, V. O. ; BARRON, George S. ; GRIMSLEY, Harper R. ; MAYHALL, Nicholas J. ; BARNES, Edwin ; ECONOMOU, Sophia E.: Qubit-ADAPT-VQE: An Adaptive Algorithm for Constructing Hardware-Efficient Ansätze on a Quantum Processor. In: *PRX Quantum* 2 (2021), Nr. 2, S. 020310. <http://dx.doi.org/10.1103/PRXQuantum.2.020310>. – DOI 10.1103/PRXQuantum.2.020310
- [TZS24] TOMA, Lilia ; ZAJAC, Markus ; STÖRL, Uta: *Solving Distributed Flexible Job Shop Scheduling Problems in the Wool Textile Industry with Quantum Annealing*. <http://arxiv.org/pdf/2403.06699>. Version: 2024
- [VGS⁺20] VIKSTÅL, Pontus ; GRÖNKVIST, Mattias ; SVENSSON, Marika ; ANDERSSON, Martin ; JOHANSSON, Göran ; FERRINI, Giulia: Applying the Quantum Approximate Optimization Algorithm to the Tail-Assignment Problem. In: *Physical Review Applied* 14 (2020), Nr. 3, S. 034009. <http://dx.doi.org/10.1103/PhysRevApplied.14.034009>. – DOI 10.1103/PhysRevApplied.14.034009
- [VMR15] VENTURELLI, Davide ; MARCHAND, Dominic J. J. ; ROJO, Galo: *Quantum Annealing Implementation of Job-Shop Scheduling*. <http://arxiv.org/pdf/1506.08479>. Version: 2015
- [WCA⁺24] WANG, Samson ; CZARNIK, Piotr ; ARRASMITH, Andrew ; CEREZO, M. ; CINCIO, Lukasz ; COLES, Patrick J.: Can Error Mitigation Improve Trainability of Noisy Variational Quantum Algorithms? In: *Quantum* 8 (2024), 1287. <http://dx.doi.org/10.22331/q-2024-03-14-1287>. – DOI 10.22331/q-2024-03-14-1287

- [WFC⁺21] WANG, Samson ; FONTANA, Enrico ; CEREZO, M. ; SHARMA, Kunal ; SONE, Akira ; CINCIO, Lukasz ; COLES, Patrick J.: Noise-induced barren plateaus in variational quantum algorithms. In: *Nature Communications* 12 (2021), Nr. 1, 6961. <http://dx.doi.org/10.1038/s41467-021-27045-6>. – DOI 10.1038/s41467-021-27045-6. – ISSN 2041-1723
- [WK20] WANG, Yulun ; KRSTIC, Predrag S.: Prospect of using Grover’s search in the noisy-intermediate-scale quantum-computer era. In: *Physical Review A* 102 (2020), Nr. 4, S. 042609. – ISSN 2469-9926
- [XSRH22] XIONG, Hegen ; SHI, Shuangyuan ; REN, Danni ; HU, Jinjin: A survey of job shop scheduling problem: The types and models. In: *Computers & Operations Research* 142 (2022), 105731. <http://dx.doi.org/10.1016/j.cor.2022.105731>. – DOI 10.1016/j.cor.2022.105731. – ISSN 0305-0548
- [YRBS22] YARKONI, Sheir ; RAPONI, Elena ; BÄCK, Thomas ; SCHMITT, Sebastian: Quantum annealing for industry applications: introduction and review. In: *Reports on progress in physics. Physical Society (Great Britain)* 85 (2022), Nr. 10, 104001. <http://dx.doi.org/10.1088/1361-6633/ac8c54>. – DOI 10.1088/1361-6633/ac8c54
- [ZDZ⁺19] ZHANG, Jian ; DING, Guofu ; ZOU, Yisheng ; QIN, Shengfeng ; FU, Jianlin: Review of job shop scheduling research and its new perspectives under Industry 4.0. In: *Journal of Intelligent Manufacturing* 30 (2019), Nr. 4, 1809–1830. <http://dx.doi.org/10.1007/s10845-017-1350-2>. – DOI 10.1007/s10845-017-1350-2. – ISSN 1572-8145
- [ZLRG08] ZHANG, Chao Y. ; LI, PeiGen ; RAO, YunQing ; GUAN, ZaiLin: A very fast TS/SA algorithm for the job shop scheduling problem. In: *Computers & Operations Research* 35 (2008), Nr. 1, 282–294. <http://dx.doi.org/10.1016/j.cor.2006.02.024>. – DOI 10.1016/j.cor.2006.02.024. – ISSN 0305-0548
- [ZTB⁺20] ZHU, Linghua ; TANG, Ho L. ; BARRON, George S. ; CALDERON-VARGAS, F. A. ; MAYHALL, Nicholas J. ; BARNES, Edwin ; ECONOMOU, Sophia E.: *An adaptive quantum approximate optimization algorithm for solving combinatorial problems on a quantum computer*. <http://arxiv.org/pdf/2005.10258>. Version: 2020
- [ZWC⁺20] ZHOU, Leo ; WANG, Sheng-Tao ; CHOI, Soonwon ; PICHLER, Hannes ; LUKIN, Mikhail D.: Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices. In: *Physical Review X* 10 (2020), Nr. 2, S. 021067. <http://dx.doi.org/10.1103/PhysRevX.10.021067>. – DOI 10.1103/PhysRevX.10.021067
- [Zyg18] ZYGELMAN, Bernard: *A first introduction to quantum computing and information*. ham : Springer Nature Switzerland, 2018. – ISBN 3319916289