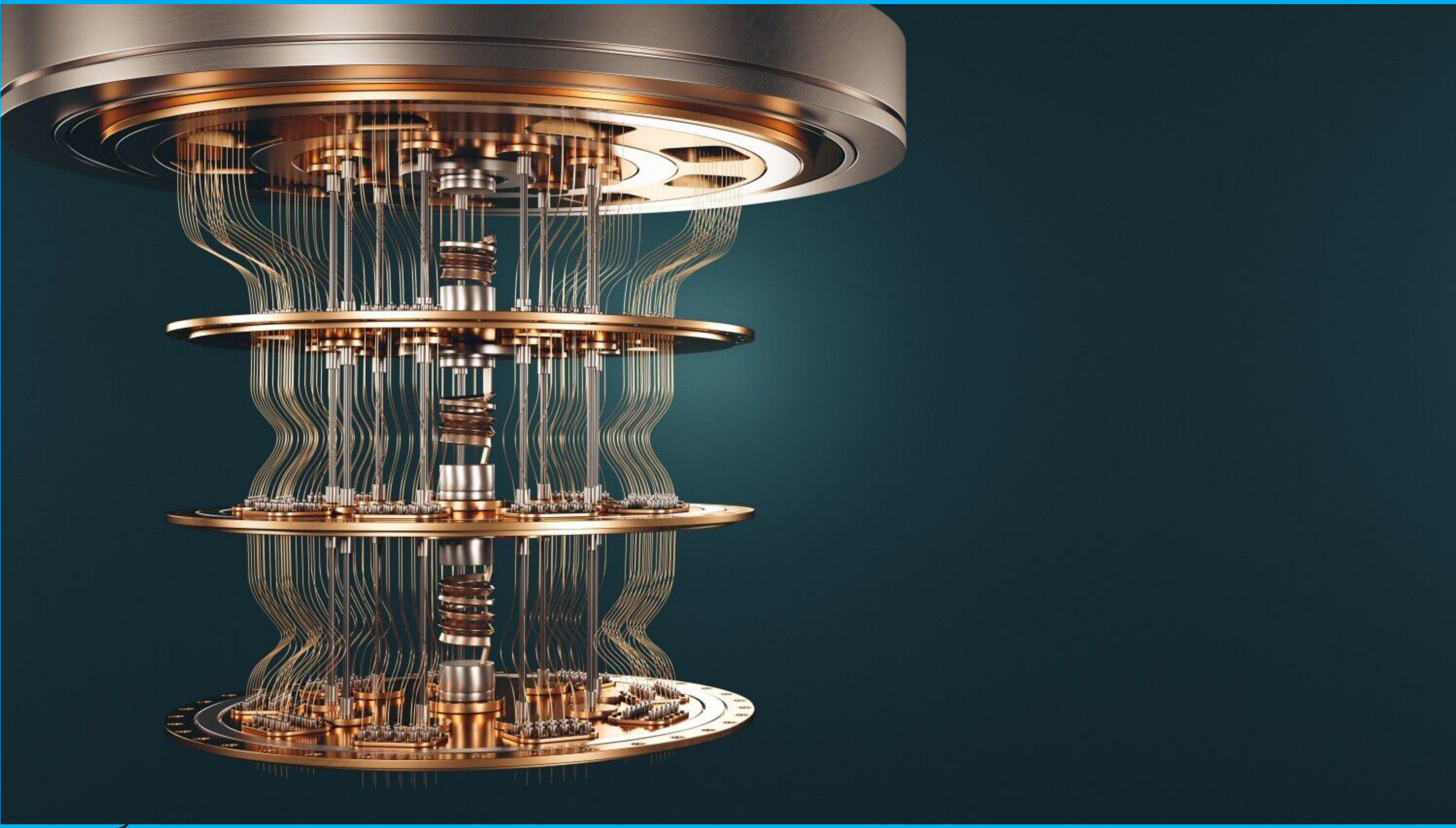
QPack: QAOA as scalable application-level quantum benchmark

Koen Mesman



QPack: QAOA as scalable application-level quantum benchmark

by

Koen Mesman

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday September 24, 2021 at 11:00 AM.

Student number:	4359011	
Project duration:	November 1, 2020 – September 24, 2021	
Thesis committee:	Dr. ir. Z. Al-Ars,	Technische Universiteit Delft, supervisor
	Dr. rer. nat. M. Moller,	Technische Universiteit Delft
	Ir. A. Sarkar,	Technische Universiteit Delft

Preface

I would like to express my gratitude to both my supervisors Matthias Moller and Zaid Al-Ars for their enthusiasm and continued support. I also want to thank Aritra Sarkar, Anneriet Krol and Smaran Adarsh for their feedback and engaging discussions. Special thanks to Huub Donkers for his implementation of the QPack benchmark using the XACC library and providing the simulator results, as well as his interest and enthousiasm for the project. Working with the TUDelft ABS quantum team has been inspiring and very enjoyable. Furthermore, I would like to thank César Rodríguez from the Strangeworks team for providing special access to IBM's Montreal and Nairobi quantum hardware, as well as the use of and support for IBM's runtime environment. Finally, I want to thank my wife and my family for their confidence in me and their love and support.

Koen Mesman
Delft, September 2021

Contents

1
Towards a standard quantum benchmark

Currently, quantum computing is making large steps to becoming a mature technology. Many companies are developing their own quantum hardware for both research and preparing for practical deployment. The main challenges in the past years have been an abundance of practical applications for the few-qubit Noisy Intermediate-Scale Quantum (NISQ) quantum hardware and scaling the qubits and depth of the quantum hardware. For these reasons, quantum computing was not yet at the maturity level needed to create a useful application level quantum benchmark.

Many quantum benchmarks have been developed, but these either aim at qubit level assessment or are lacking in practical usability. For a benchmark to be useful, the practical application of the quantum hardware needs to be reflected. For this a practical application is required and this was not yet achievable with current quantum hardware scales. Benchmarks at the component level (individual qubits, quantum logic gates) have been developed widely and are useful for the development of the hardware. This is, however, more aimed at basic quantum research rather than application [21]. This also serves a fundamentally different goal than a performance measure of quantum computers. Many different forms of quantum hardware are being used, without a clear dominating implementation. Each of these have different dynamics and metrics, and can therefore not be generalized [97]. For this reason, making a benchmark for low-level hardware aspects will not properly reflect its performance compared to other implementations of the quantum hardware.

Furthermore, while benchmarks for single qubit operations have been developed [32, 36, 70, 102], these performances do not accurately reflect quantum operations on larger scales. Noise levels, for example, are an important metric in single-gate performance. The noise, however, varies per gate and due to qubit entanglement will propagate unpredictably throughout the system [38, 43, 97]. This makes it impossible to use single gate noise performance to extrapolate for the entire system, while for classical computers this would have been possible. These difficulties in determining the performance will require abstraction from the low level performance and an application level benchmark is needed to properly verify the performance.

Other benchmarks such as Random Benchmarking (RB) or quantum volume [20] are well known quantum benchmarks, but provide limited insight into practical use of quantum computers. RB is an example of a volumetric benchmark, which determines to which circuit depth the quantum hardware can hold the required fidelity for a set number of qubits. RB can determine the fidelity for a random sequence of quantum gates, which should give a general idea of how the hardware performs. By using RB, or any other volumetric benchmark, the quantum volume can be determined. However, this gives little insight on the performance of a practical algorithm on the quantum hardware. Another challenge in developing a benchmark for quantum computing, is that there is no consensus on which metrics should be benchmarked to reflect performance properly.

In this thesis, we present QPack: a benchmark designed to evaluate the modern quantum computing stack. The focus will be placed on practical, application-level benchmarking. This will be done according to common application benchmarking metrics as generally used in classical computing, but also evaluated quantum specific performances. The quantum performance will not be comparable to single-qubit bench-

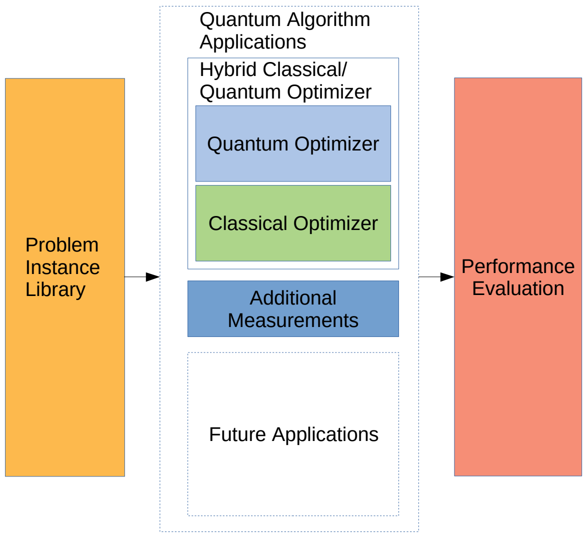


Figure 1.1: Schematic outline of the QPack benchmark.

marking, but will evaluate the quantum computer in its entirety. The benchmark has three main components, as presented in Figure 1.1:
Problem instance library: A set of problems are provided to ensure practical relevance of the benchmark and a diverse means of evaluation. Multiple problems targeted by our benchmark are discussed in Chapter 3.
Quantum algorithm applications: A quantum algorithm must be chosen which can be applied to various problems and is applicable on current quantum hardware and quantum simulators. With current quantum hardware in mind, a hybrid classical-quantum optimizer is chosen as our first quantum algorithm. The selected quantum optimizer is QAOA (Quantum Approximate Optimization Algorithm) [42]. The selection of this optimizer is further elaborated in Chapter 2. The options and selection of the classical optimizer are discussed in Chapter 4. Measurements of these classical optimizers are done as well, which is further elaborated in Section 4.3. The QPack benchmark is envisioned to grow as more quantum algorithms become applicable on quantum hardware. Some of these options will be discussed in Chapter 4
Performance evaluation: The performance of quantum computers and simulators need to be evaluated according to a selection of metrics. Further discussion on these metrics is presented in Chapter 4.

In this thesis, a selection will be made on which quantum algorithm is best suited for benchmarking NISQ era quantum computers. Furthermore, a selection between different implementations to practical problems will be made (i.e. variations of the algorithm to find the solution to a specific mathematical problem). These selected implementations will be used to benchmark NISQ quantum computers, as well as simulators, on relevant metrics. Part of the thesis will be determining the set of metrics on which the quantum computers will be evaluated. This thesis will also cover the strategy on how these metrics can be evaluated, and which method is chosen for the QPack benchmark.

The rest of this thesis is organized as follows. A general overview of near-term hybrid quantum algorithms is presented in Chapter 2. A brief theoretical background on complexity theory is given in Chapter 3. We also propose the concept of quantum technology readiness level (QTRL) to classify the development status of quantum algorithms and their readiness to be used in the field. Chapter 4 discusses the implementation of the benchmark, as well as the details of various engineering choices regarding the implementation. The results of this implemented benchmark are presented and discussed in Chapter 5. The drawn conclusions and suggestions for future improvements are presented in Chapter 6.

2

Quantum algorithm selection

In this chapter, a sort review of various hybrid quantum optimization algorithms are given. Hybrid quantum algorithms were chosen as these are promising to be effective for relatively few qubits []. For this reason, the quantum approximate optimization algorithm (QAOA), the variational quantum eigensolver (VQE) and the quantum adiabatic algorithm (QAA) are briefly explained. The chosen algorithm will be explained in further detail. Finally, emerging trends and concerns on implementing the algorithm (parameter tuning, etc.) will be discussed.

2.1. Near term hybrid quantum optimization algorithms

Quantum algorithms have the potential to outperform classical algorithms, also called quantum supremacy. Many quantum algorithms have been developed and algorithms such as Shor's algorithms are proven to perform better than their classical counterpart. Currently the development of quantum hardware is not yet at the scale to support such algorithms. In order to achieve practical quantum algorithms, with quantum supremacy in mind, algorithms have emerged to fit the current NISQ technology. With this technology, algorithms for 50-100 qubits could be supported. In this section, a selection of similar near term algorithms are discussed which are candidates for near term implementation.

Quantum approximate optimization algorithm (QAOA)

The first algorithm discussed in this section is QAOA [42], introduced by Farhi et al. in 2014. This algorithm can be used to approximate a set of NP-hard optimization problems with relatively few qubits. NP-hard optimization problems are a large bottleneck for current classical computing systems, as these cannot be efficiently calculated using classical computers. The type of optimization problems that QAOA can be applied to consist of a set of parameters with a set of constraints where the parameters need a specific configuration for finding the optimal solution. QAOA is a layered quantum algorithm that theoretically increases in accuracy with the increase of the number of iterations (layers) denoted by parameter p. This has many practical applications, ranging from multiprocessor scheduling to warehouse-order picking and flight-timetable planning. The specifics of these applications will be further discussed in Appendix 3 and Section 3.4.

Variational quantum eigensolver (VQE)

The variational quantum eigensolver (VQE) is a quantum algorithm that finds the eigenvalues of a Hamiltonian H, representing the system to be solved [98]. It is used to minimize the objective function $\langle\psi(\theta)|\, H\, |\psi(\theta)\rangle$. The minimum eigenvalue sought is represented by the ground state of the system. The state $|\psi(\theta)\rangle$ can be varied by changing θ to converge to the minimum, by means of a classical optimizer. This is a minimum eigenvalue problem and is a constraint satisfaction problem (CSP), which is generally NP-hard [86]. It is also one of the algorithms, like QAOA, that is believed to be one of the first algorithms to run effectively on a near-term quantum computer.

Quantum adiabatic algorithm (QAA)

The quantum adiabatic algorithm (QAA) is able to find the optimal solution to a satifyability problem when given enough time by finding the ground state of the slowly varying Hamiltionian $H(t)$ [41]. QAA uses the time-dependent Hamiltonian $H(t) = (1 - t/T)B + (t/T)C$. The algorithm starts in the highest energy state of B, and seeks the highest energy state of C. This makes use of the Perron-Frobenius theorem that the difference in energies between the top state and the one below is greater than 0 for $t < T$ [42]. This means that if T is sufficiently large, the optimal solution can be found by *slowly* varying t for $0 \leq t \leq T$ [41]. A downside of QAA is that success probability is not a monotonic function of T. This means that the success probability can drop, whereas for QAOA the success probability always increases with p [30, 42]. It is also shown that QAA can be trapped in a local minimum, which is not the case for QAOA [42]. It is also worth mentioning that QAA is not a gate-based algorithm.

2.2. Introduction to QAOA

As QAOA can be implemented to a larger variety of applications compared to VQE and has an advantage over QAA, QAOA was selected for the QPack benchmark. To understand this algorithm better for implementation, further details are discussed.

QAOA is designed to find approximate solutions for combinatorial optimization problems with the help of quantum computing. To go in further detail, the algorithm will be explained using the equations given by Farhi et al. In combinatorial optimization problems, the goal is to maximize or minimize the number of clauses (m) satisfied. A clause (ψ) is a Boolean requirement, for example for a n-bit Boolean string $\mathbf{z} = z_1, ...z_n$:

$$\psi_{\text{example}} : z_1 \wedge z_2 \tag{2.1}$$

To satisfy the clause of the example z_1 and z_2 must satisfy $z_1 = z_2 = 1$.

The QAOA algorithm depends on a parameter $p \geq 1$, which determines the quality of the approximation. Quality is given as an approximation ratio. This ratio is defined as either the number of clauses satisfied by the QAOA algorithm divided by the number of clauses satisfied in the optimal assignment:

$$r = \frac{M_p}{C_{\max}(z)} \tag{2.2}$$

or as the number of clauses satisfied by the QAOA algorithm divided by the number of clauses:

$$r = \frac{M_p}{m} \tag{2.3}$$

The latter is considered to be stronger [74]. The depth of the quantum circuit required to implement this algorithm grows linearly with p times the number of constraints m in the worst case [42].

The equation that needs to be maximized (cost Hamiltonian) is as follows:

$$C(z) = \sum_{\psi \in \Psi} C_\psi(z) \tag{2.4}$$

Here, $C_\psi(z) = 1$ if clause $\psi \in \Psi$ is satisfied, and 0 otherwise. In the quantum approach we define the Boolean string as a vector $|z\rangle$ in the computational basis $\{|0\rangle, |1\rangle\}$. A general guide to form such cost Hamiltonians from a cost function has been presented by Choi et al. [26]. The operator for equation (2.4) then becomes

$$U(C, \gamma) = e^{-i\gamma C} = \prod_{\psi \in \Psi} e^{-i\gamma C_\psi} \tag{2.5}$$

The terms of the product commute as these are diagonal in the computational basis and each term's locality is the locality of clause ψ. The unitary is dependent on angle vector $\boldsymbol{\gamma}$ and can be restricted to $[0, 2\pi]$ since C has integer values [42]. Choosing the number of elements γ is divided in, will impact precision and performance. To improve the precision of the QAOA algorithm, multiple cycles of the gates will be applied. The number of cycles is denoted as p. For integer $p \geq 1$, $\boldsymbol{\gamma}$ is defined as $\boldsymbol{\gamma} \equiv \{\gamma_1, ..., \gamma_p\}$, where each γ_i with $1 \leq i \leq p$ is an angle within the range $[0, 2\pi]$.

To sum the outcomes of equation (2.4), the so-called mixer Hamiltonian B or H_B is introduced:

$$B = \sum_{j=1}^{n} \sigma_j^x \tag{2.6}$$

Here, the Pauli X-gates operate on a single qubit $|z_j\rangle$. For integer $p \geq 1$, $\boldsymbol{\beta}$ is defined as $\boldsymbol{\beta} \equiv \{\beta_1, ..., \beta_p\}$, where each β_i with $1 \leq i \leq p$ represents a angle β for a single cycle. The general definition of the unitary applied to a single qubit then becomes:

$$U(B,\beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x} \tag{2.7}$$

Here, σ_j^x is the Pauli-X operator on qubit j. The unitary $U(B,\boldsymbol{\beta})$ depends on angle vector $\boldsymbol{\beta}$, of which each $\beta_i \in \boldsymbol{\beta}$ runs from 0 to π. The vector $|z\rangle$ is placed in superposition:

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_z |z\rangle \tag{2.8}$$

The angle dependent quantum state can then be written as

$$|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle = U(B,\beta_p)U(C,\gamma_p)...U(B,\beta_1)U(C,\gamma_1)\,|s\rangle \tag{2.9}$$

The expectation of C then becomes:

$$F_p(\boldsymbol{\gamma},\boldsymbol{\beta}) = \langle\boldsymbol{\gamma},\boldsymbol{\beta}|\,C\,|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle \tag{2.10}$$

The maximum expectation for p QAOA cycles (M_p) is then defined as:

$$M_p = \max_{\boldsymbol{\gamma},\boldsymbol{\beta}} F_p(\boldsymbol{\gamma},\boldsymbol{\beta}) \tag{2.11}$$

Since increasing parameter p will increase the quality of the solution, in other words a equal or higher expectation will be found, we can state the following:

$$M_p \geq M_{p-1} \tag{2.12}$$

With the intent that $\lim_{p\to\infty} M_p = \max_z C(z)$, which is proven by Farhi et al. [42].

To give an intuition on how the algorithm operates, an example will be given. In this example, a system with 2 (qu)bits per clause is examined. The general clause will be worked out for bits j and k:

$$C = \sum_{<jk>} C_{<jk>} \tag{2.13}$$

This is derived from equation (2.4). The expectation from equation (2.10) becomes:

$$F_p(\boldsymbol{\gamma},\boldsymbol{\beta}) = \sum_{jk} \langle s|\,U^\dagger(C,\gamma_1)\cdot ... \cdot U^\dagger(B,\beta_p)C_{<jk>}U(B,\beta_p)\cdot ... \cdot U(C,\gamma_1)\,|s\rangle \tag{2.14}$$

Here $U^\dagger$ denotes the conjugate transpose of the unitary operator U.

The operator for a single clause then becomes:

$$U^\dagger(C,\gamma_1)...U^\dagger(B,\beta_p)C_{<jk>}U(B,\beta_p)...U(C,\gamma_1) \tag{2.15}$$

The contribution for $p = 1$ can then be written as:

$$U^\dagger(C,\gamma_1)U^\dagger(B,\beta_1)C_{<jk>}U(B,\beta_1)U(C,\gamma_1) \tag{2.16}$$

Using an example for C, used in MaxCut applications (Appendix 3.3.2), it can be shown which operators contribute to this equation. MaxCut and other applications will be discussed later in greater detail. For this example, the definition for the maximized equation becomes:

$$C_{<jk>} = \frac{1}{2}(-\sigma_j^z\sigma_k^z + 1) \tag{2.17}$$

Here, the Pauli-Z operator is denoted as σ^z. Now substituting equation (2.7), gives rise to the following:

$$U^\dagger(C,\gamma_1)e^{i\beta_1(\sigma_1^x+...+\sigma_n^x)}\frac{1}{2}(-\sigma_j^z\sigma_k^z + 1)e^{-i\beta_1(\sigma_1^x+...+\sigma_n^x)}U(C,\gamma_1) \tag{2.18}$$

All σ^z terms in the second exponent except for σ_j^z and σ_k^z can be moved in front of the $\frac{1}{2}(-\sigma_j^z\sigma_k^z+1)$ term as they do not interact with qubits j and k, i.e. the operators commute ($\hat{A}\hat{B} = \hat{B}\hat{A}$). Because these operators can be moved to the front, they cancel out with their Hermitian conjugate. For increasing p, $U(B,\beta)$ will contain the qubits connected one step further from the original clause. For example, if qubit j shares a clause with qubit l, the operators acting on qubit j will be included as well. By expanding this for increasing p, more qubits are considered for each clause, effectively increasing the quality of the approximation. To expand this, the unitary $U(C,\gamma)$ must be understood. This unitary operator C_α is 1 if the condition is met and 0 otherwise. In the case of a graph (e.g. MaxCut), this means that the conditions are a connectivity matrix. This makes the term 1 if a connection between the qubits exists. The unitary of γ_1 will contain the σ^z for j and k, but also all qubits which are connected. This has as consequence that $U(B,\beta_2)$ will keep σ_j and σ_k but also for all qubits connected to these qubits. This way, increasing p will increase the qubits in the graph.

2.3. Practical concerns and emerging trends for QAOA

Practical concerns

In order to apply QAOA algorithms to practical applications, some hard challenges must be achieved first. One of the major challenges is that even while QAOA uses much less resources than, e.g., Grover's algorithm, it still requires hundreds of qubits in order to achieve better performance than classical algorithms. Specifically for QAOA, the depth of a circuit can be a huge bottleneck if it does not make use of an optimized implementation [29]. Furthermore, the limited precision of QAOA will require a large parameter p to achieve optimal results. This parameter is expected to grow almost linearly with the problem size, but finding the optimal value for p is one of the big challenges for QAOA to become practical [107]. Another challenge is the tuning of parameters $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. The original paper of QAOA [42], proposes an optimal configuration for the $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ parameters, but is considered computationally expensive [29, 33]. Instead, optimal parameters can be found for specific problem instances [29]. For specific problems, a value for p can be chosen in such a way that the approximation is better than the classical counter part. For example, Crooks shows that for $p \geq 8$, the QAOA for MaxCut could outperform the Goemans-Williamson algorithm [29], which is currently considered the most efficient classical algorithm for MaxCut. This way, a lower bound can be set on the p parameter.

Another challenge is that while QAOA might perform good in theory and on a simulator, on real hardware the performance might be considerably worse [115] or could make an implementation unfeasible [105]. The relatively large amount of noise in current quantum hardware results in better performance for algorithms with lower depth [78] and will likely be a strong factor in implementing QAOA in practice.

Emerging trends

An emerging trend is to optimize the parameters for QAOA using machine learning [8, 68, 114]. Results of Khairy et al. show that training for an 8-qubit system, the optimality ratio (equation 2.2) can be improved 1.16 to 8.61 compared to the Nelder-Mead optimizer, depending on the MaxCut graph solved and QAOA depth p (depths 1, 2 and 4 were tested). Wauters et al. [114] aims to solve this problem by suggesting that a strategy learned from a small system, can be applied to a larger system to achieve similar speedup. Their results using local optimizations on larger systems show this is achievable. Alam et al. [8] applies machine learning to a classical optimizer that iteratively runs the QAOA algorithm with new parameters. Their implementations show an average reduction of iterations of 44.9% and is also a good solution for run-time optimization.

In terms of integration, Qiskit and Rigetti actively include QAOA in their repositories. Qiskit currently only supports Ising-type problems such as MaxCut [33], while Rigetti also has support for, e.g., the partitioning problem and makes this publicly available on Github. Rewriting the problem to a format that can be optimized using QAOA is a challenge in itself as well. In recent papers, Quadratic Unconstrained Binary Optimization (QUBO) [49] is used to map problems to QAOA. QUBO is a compelling approach as this allows the QAOA algorithm to be unaltered, while only adjusting the classical problem formulation. This requires some additional work, but would otherwise be needed to properly design the QAOA algorithm. QUBO is used by, e.g., Qiskit [39] and XACC [80], allowing a single QAOA function definition, with a standardised problem formulation. In terms of performance, no clear comparison has been published comparing the QUBO implementation and the straightforward QAOA implementation.

Another downside of QAOA could be the limited range of problems that it can solve. While all problems discussed in Section 3.4 are considered important challenges, the quantum algorithm is still limited to very similar problems. A generalization of QAOA is presented as the Quantum Alternating Operator Ansatz, which covers a wider range of problems [54]. As discussed in Section 3.4, some algorithms are not well suited for QAOA and other quantum algorithms will likely perform better results. The constraint density is also shown to impact the performance of QAOA [5]. This means that performance will depend not only on the size of a graph, but also the number of constraints compared to the size of the graph.

3 Complexity theory

This section will explore potential fields, where QAOA could be implemented to achieve significant speedup over classical algorithms. A brief introduction to the theoretical problems will be given togeher with their practical applications,i.e. how the algorithm can be applied to a real-world problem as opposed to the mathematical problem. For each problem, the current QAOA implementations will be reviewed in terms of complexity and approximation accuracy.

3.1. A brief recap of complexity theory

Computational problems can be sorted by their time complexity. This refers to the computation time required to solve the problem for n inputs. Often, big-O (Order of magnitude) notation is used to describe the time required for the worst-case scenario, while big-Ω notation is used for best-case scenarios, i.e. the lower bound of the magnitude [71]. If both the worst-case and best-case scenarios match, big-Θ notation can be used. A graph of common time complexities is given in Figure 3.1. The time complexity gives a rough indication on how long an algorithm will take to perform a computation for n inputs. More importantly, it gives an indication of the rate that the computation time increases with respect to the number of data inputs. Slower growth of time with respect to data inputs indicates better scalability of the program.

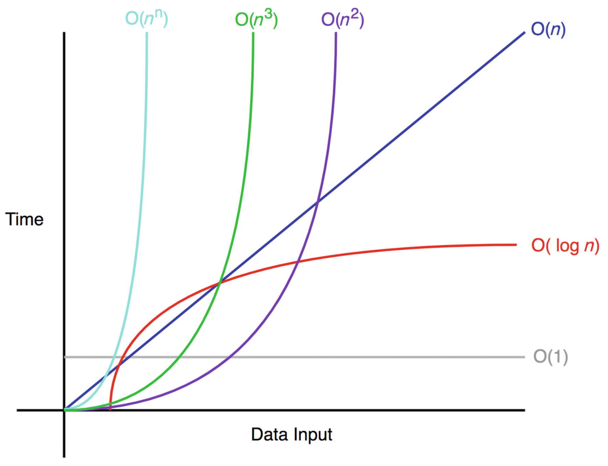


Figure 3.1: A graph of common time complexities [13].

Problems can be subdivided in classes, depending on their complexity. A visualization of the complexity classes is given in Figure 3.2. The complexity classes consist of:
P: Problem A is of polynomial time complexity, if it can be computation time scales polynomial, i.e. in the range from $O(1)$ to $O(n^p)$, with $p \in \mathbb{N}$ fixed [59]. Problems in this complexity can be solved with classical computers.
NP: **NP** stands for non-deterministic polynomial time and is used to classify decision problems. These

problems initially determine a solution in a non-deterministic way (a guess), and is then verified in polynomial time [11]. We can therefore define a problem A, if A requires its solution to be guessed and requires an oracle to be verified in polynomial time. Whether **NP = P**, is one of the famous millennium prize problems and is yet to be proven [24].

NP-complete: Problem A is considered **NP-complete** if A is in **NP** and if all **NP** problems can be reduced to A in polynomial time. This means that problem A can be used as subroutine for any **NP** problem, with the number of subroutine calls polynomial [69]. If the problem is only known to be in **NP**, it is considered **NP-hard** [59].

BPP: Problem A is in Bounded-error probabilistic polynomial time complexity if A is a decision problems, to which a randomized algorithm can find the solution in polynomial time with probability $p \geq \frac{1}{2}$ [59]. The problem set includes **P** and parts of **NP**. The relation between **BPP** and **P** is unclear. Cases have been presented for which **BPP = P** [60], but is not proven for all cases [51]. Goldreich discusses how with derandomization **P = BPP** [51], but further detail on this is considered beyond the scope of this thesis. The relation between **BPP** and **NP** is also unclear. However, oracles have been presented for which **BPP** is not contained in **NP** [109].

BQP: Bounded-error quantum polynomial time is the quantum analogue to the BPP complexity class [59, 90]. Problem A is in **BQP** if a quantum algorithm has a probability of $p \geq \frac{1}{2}$ to find the correct answer in polynomial time [59]. As for the relation to **BPP**, it has been proven that **BPP** $\subseteq$ **BQP**, and that **BPP** $\neq$ **BQP** [17].

EQP: Problem A is in exact quantum polynomial time complexity if a solution to A can be found in polynomial time by a quantum algorithm with probability $p = 1$ [59]. The relation between **BQP** and **EQP** is still an open question, but is discussed in detail by Fortnow et al. [44].

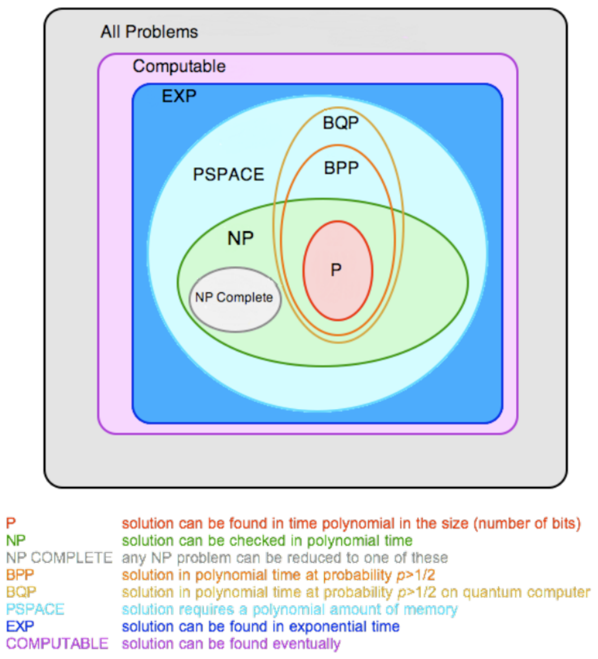

All Problems
Computable
EXP
PSPACE
BQP
BPP
NP
P
NP Complete
P solution can be found in time polynomial in the size (number of bits)
NP solution can be checked in polynomial time
NP COMPLETE any NP problem can be reduced to one of these
BPP solution in polynomial time at probability p>1/2
BQP solution in polynomial time at probability p>1/2 on quantum computer
PSPACE solution requires a polynomial amount of memory
EXP solution can be found in exponential time
COMPUTABLE solution can be found eventually

Figure 3.2: A Venn-diagram of the complexity classes [90].

In 1971, Stephen Cook proved that the satisfiability problem is **NP-complete** [28]. The year after that, Richard Karp expanded the list of **NP-complete** problems to the well known list of Karp's 21 **NP-complete** problems [67]. An overview of the problems is given in Figure 3.3. This list was established by showing that these 20 additional problems can be reduced to the satisfiability problem in polynomial time. An problem can only be **NP-complete** if it is a decision problem. Finding the optimal solution for such a problem is **NP-hard**. The solutions to these problems have been approximated by classical algorithms to a limited accuracy (in polynomial time). QAOA improves on these algorithms by making the guess more accurate while remaining within polynomial time complexity. From here on, some of the algorithms used to find the optimal solutions to Karp's 21 problems will be discussed, including practical applications. Not all 21 problems will be discussed, as only the problems will be considered where a QAOA implementation has been designed to improve on the classical counterpart. Other **NP-hard** problems, not directly related to Karp's 21 problems, will be included as well. The satisfiability problem is a problem where Boolean clauses need to be satisfied. All of Karp's problems can be reduced to a satisfiability problem in polynomial time. The satisfiability problem is commonly denoted as Max-Sat. A k-Max-Sat is a satisfiability problem where each clause is limited to k variables. The QAOA algorithm is applied to the *approximation* of NP-hard problems, to find the optimal solution of **NP** problems are **NP-hard**. This means that, while a solution might be found with QAOA, this only has a probability of finding the solution and is therefore contained in **BQP** complexity. Note that QAOA can not approximate a solution for all **NP-hard** problems, but a subset of similar structured problems. As discussed above, theoretically QAOA could find a solution with probability $P = 1$. But since this is not demonstrated to be achievable in polynomial time, it cannot be considered as **EQP** complexity.

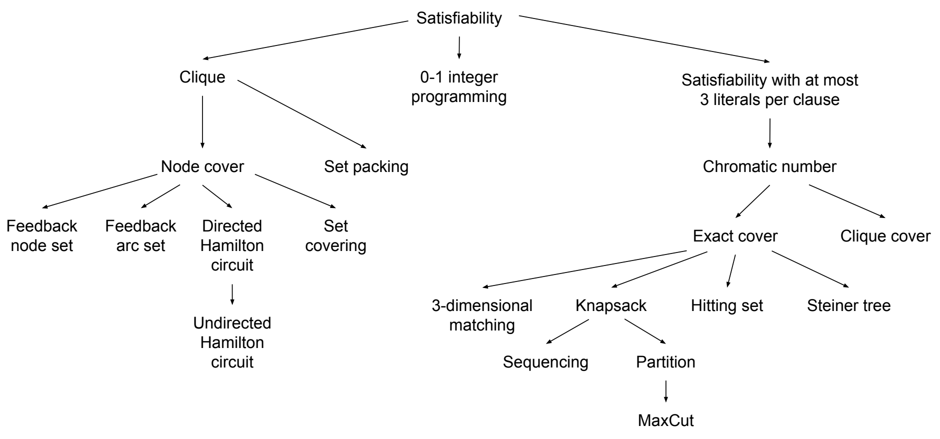


Figure 3.3: Karp's 21 problems, with the reduction relations given as a tree diagram [67].

3.2. The QAOA implementations to the problems

This section explores QAOA algorithms applied to various NP-hard problems.

Graph partition

In the graph partition problem, a graph (V,E) is divided into partitions with the objective to minimize edges between the partitions in order to simplify graph analysis. This problem can be applied to VLSI circuit design as the first step of the physical design [65]. A QAOA implementation has been presented by Ruan et al. [100], with a general QAOA scheme for linear inequality constraints.

Problems with a linear equality constraint have the characteristic that their constraints are represented as:

$$\mathbf{A}\cdot\mathbf{x} = \mathbf{b} \tag{3.1}$$

With $\mathbf{A}$ being a $m\times n$ matrix for m constraints and n components, $\mathbf{x}$ the solution vector of length n and $\mathbf{b}$ a vector of length m. Such a problem translates to a constraint function of [100] :

$$f(\mathbf{x}) = \sum_k \mathbf{c}_k \cdot \mathbf{x}_k \tag{3.2}$$

In this scheme, the B operator is defined as follows:

$$B = \sum_{\mathbf{x},\mathbf{x}'\in\Omega,\mathbf{d}(\mathbf{x},\mathbf{x}')=\mathbf{2}} |x\rangle\langle x'| + |x'\rangle\langle x| \tag{3.3}$$

Here $\mathbf{d}(\mathbf{x},\mathbf{x}')$ is the Hamming distance between solution $\mathbf{x}$ and $\mathbf{x}'$. Hamming distance is the number of bits that differ from the original. This way of encoding the B operator is derived from a random quantum walk, where positions and orders of "1's" are swapped. In the case of the graph partition problem, the B operator is rewritten by Hadfield and Hen [54, 57] to encode the constraints as:

$$B = -\sum_i^n (\sigma_i^x\sigma_{i+1}^x + \sigma_i^y\sigma_{i+1}^y) \tag{3.4}$$

Both solutions differ in outcome. The probability distribution from equation (3.3) is more accurate than the implementation from (3.4), and does not generate redundant subgraphs [100].

Packing (knapsack)

The packing problem, or knapsack problem, aims to fill a resource to a maximum extent. As example, given a knapsack with a limited capacity, fill it with as much elements as the capacity allows. Ruan et al. [100] implement QAOA on this problem for multiprocessors with a similar solution as mentioned in the previous section. For a execution time t of task, the Hamiltonian for this implementation is given by:

$$C = -\sum_{x=0}^{2^{mn}-1} \max\{\sum_{k=in}^{(i+1)n-1} t_{k \mod n} x_k\} |x\rangle \langle x| \text{ for } i = 0, ..., (m-1) \tag{3.5}$$

De la Grand'rive and Hullo also published a paper on this particular problem, with a different approach aimed at an implementation for battery revenue optimization for micro-grids [33]. In their paper, a 'relaxed' approach is presented, using a linear penalty as opposed to a quadratic penalty to improve upon the adiabatic approach [27]. The algorithm proposed, consists of four subroutines: Cost calculation, Constraint testing, Penalty dephasing and Reinitialization. In terms of time complexity, the classical algorithm for the knapsack problem runs in $O(\frac{n^3}{\epsilon})$ for ϵ-estimation [33]. The implementation presented can run with a qubit depth of $O(p(\log_2(n))^3)$, meaning that it achieves a p-polylogarithmic complexity in n. The number of ancillary qubits required for this algorithm is $O(n\log_2(n)$. Another paper published on this problem is by Roch et al. [99] and uses a cross entropy hyper parameter optimization. With this method, the penalty values are optimized. An enhancement for $p = 1$ and $p = 2$ of more than 100% is claimed compared to the method of Hadfield et al. [54]. Ruan et al. [100] expanded on the packing problem by adding arbitrary constraints. Since the previous method of changing the Hamming distance does not apply, they present a *star graph* method. Their results are claimed to be similar to classical optimization algorithms in terms of accuracy, but not much about complexity or performance is presented.

MaxCut

The details of the MaxCut problem have been discussed in detail in Section 3.3.2. The optimization can be applied for circuit layout design [14], data clustering [91] or implementing the Ising model. The MaxCut problem for QAOA has been studied extensively [29, 74, 115, 119]. It has been shown that the approximation of QAOA achieves better results than the classical Goemans-Williamson [50] for $p \geq 8$ [29]. For the general MaxCut algorithm, the gates required are $O(N^2P)$ and can be run in $O(NP)$ time assuming that $O(N)$ gates can be run in parallel for N nodes, and P the number of QAOA layers. The classical Goemans-Williamson algorithm, in contrasts, requires a run time of $O(Nm)$ for m edges [29].

Ising model

The Ising model can be approximated by the MaxCut algorithm [62]. Implementations for the Ising model can be found in physics for simulation of e.g. phase separation, lattice-based liquid-gas model or spin glasses. Applications for this model can also be found in biology for the protein folding problem [82]. The model is also supported in Qiskit [39], alongside the popular MaxCut algorithm. The Ising model has in recent publishing been compared for both classical and QAOA algorithms [89]. It becomes apparent that for large clause density, QAOA shows reachability deficits [5] and is therefore no improvement on the classical algorithms. For a large enough search space, the QAOA algorithm can reach a Grover scaling of $O(\sqrt{N})$ for N nodes.

Set packing

In the set packing problem a set of subsets is given. A minimal selection of the subsets must be chosen to cover all elements of the set [108]. A visual representation of this problem is shown in Figure 3.4.

This problem is also covered by Ruan et al. [100] by adjusting the Hamming distance to $\mathbf{d}(\mathbf{x}, \mathbf{x}') = 1$. The complexity achieved with their method is $O(n)$ for n subsets.

Vertex cover

In the vertex cover, all edges of a graph (V, E) must be covered by selecting the minimum amount of nodes. This problem is implemented by Ruan et al. with an alteration on the previous method [100]. The vertex cover problem has implementations in, e.g., monitoring. Examples are security camera placement or network link monitoring.

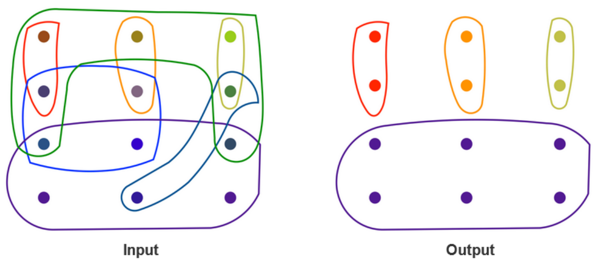


Figure 3.4: Visualization of the set packing problem [108].

Dominating set problem

The dominating set problem is very similar to the vertex covering problem, but instead of covering edges, there must be an edge to each vertex from a vertex from the selected set. The way both of these problems are used in practice is very similar as well. The cost is defined in 2 parts: the number of nodes connected to a covered set is maximized (T_k) and the number of nodes used as covering (all nodes connected to this node are covered) is minimized (D_k) [52]. The nodes are encoded as $z_k = 1$ if a node is used for covering. This translates to the following cost functions:

$$T_k(z) = \begin{cases} 1, & \text{if } z_k \text{ connected to a node } z_i \text{ for which } z_i = 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

$$D_k(z) = \begin{cases} 1, & \text{if } z_k = 0 \\ 0, & \text{if } z_k = 1 \end{cases} \tag{3.7}$$

$$C(\mathbf{z}) = \sum_{k=0}^{n} D_k(z) + \sum_{k=0}^{n} T_k(z) \text{ for n nodes} \tag{3.8}$$

The cost function C is to be maximized.
An implementation of QAOA and an analysis of the algorithm is presented by Nicholas Guerrero [52]. In the circuit presented, for n vertices the following is required [52]:

- $18n^2 - 6n$ single qubit gates
- $16n^2 - 12n$ controlled Pauli-X gates
- $2n$ qubits (of which n are ancillary)

Traveling salesman problem

The traveling salesman problem is a well known problem where a salesman needs to visit all cities (vertices) which are connected by roads (weighted edges). A special case of the problem, the undirected Hamiltonian circuit, is also one of Karp's 21 problems. A (naive) classical approach would check all routes and would have a complexity of $O(n!)$ [16]. An implementation based on Grover's algorithm has shown a quadratic speedup to $O(\sqrt{(}n!))$ [16], but such an implementation would require a lot of quantum resources and would currently not be applicable to quantum hardware. A QAOA solution has been presented by Radzihovsky et al. [96] and by Henry and Dudas [31]. The respective source codes for implementation on a Rigetti (pyquil and the Rigetti QVM) quantum computer are available at `https://github.com/murphyjm/cs269q_radzihovsky_murphy_swofford` [96] and `https://github.com/danielhenry1/QAOA_TSP` [31]. Radzihovsky et al. show that the number of gates in the cost Hamiltonian scales with n^2 for n cities, and a circuit depth of $2n$ for even n and $2(n-1)$ for uneven n. Ruen et al. also published an implementation using n^2 qubits. The traveling salesman problem has a large range of applications, such as computer wiring, drilling PCB and order-picking in warehouses [79]. A recent implementation of QAOA on the traveling salesman problem is presented by Sarkar et al. [105]. In this implementation, the QAOA is applied for quantum accelerated de Novo DNA sequence reconstruction. An application and QAOA simulation are presented, but does not feature a hardware implementation as the currently available quantum hardware are not sufficient to support meaningful results [105].

Tail assignment problem

The tail assignment problem deals with the scheduling of a set of flights to a set of aircraft to create a feasible flight schedule, while keeping the cost of operation as low as possible. This problem is an optimization of a packing problem with additional constraints. Vikstal et al. propose an implementation of QAOA for this problem [112]. In their publication, they show that for QAOA depth $p = 2$, an outcome accuracy of 99.9% can be achieved, but requires 74 measurements for 25 route instances. This case performs better than quantum annealing. Using more measurements instead of increasing p is computationally less expensive.

Facility placement problem

The facility placement problem aims to place a node (facility) with the least transportation cost (this in itself is a Weber problem [22]), while considering other constraints. A QAOA implementation is presented by Quiñones and Junqueira [95]. This paper, however, shows that the QAOA implementation has a much higher cost and lower optimal solution probability than a VQE for the investigated case. This might indicate that this type of problem is not suited for QAOA.

QAOA for Grover's algorithm

An interesting approach has been presented in [63], where the principals of QAOA have been applied to Grover's algorithm. Grover's unstructured search algorithm is a well known quantum algorithm for which the oracle can find an entry in an unstructured database. Much like the implementations discussed before, it uses p iterations to improve fidelity/accuracy. This implementation could potentially make the algorithm that is proven to perform better than classical algorithms (quadratic speed-up), executable on relatively few qubits. This would allow it to run on near-future devices. Some critical issues, however, exist with the implementation presented in [63]. QAOA has limited accuracy, while Grover's algorithm can only be applied practically (e.g. database searching) if an exact solution can be found. The authors mitigate this problem by checking with multiple rounds whether the found solution is correct. This might require the algorithm to run for many iterations before the solution is found. Furthermore, the chance of success of the algorithm is only 50%, making it impractical for implementation. Error-correction schemes might make this implementation worthwhile, but it is unclear whether such a solution can be found. Quantum error correction in adiabatic models is still a challenge [117], but could make this application implementable.

3.3. Details on max-kSAT and max-kXOR

The general optimization problems discussed in this section will be the combinatorial satisfaction problems (CSP) max-kSAT and max-kXOR. Here, k denotes the exact number of variables contained in a clause. The special cases of CSP will be discussed: the one with *typicality* and the MaxCut problem. These problems represent general applications from which other practical applications can be derived. These practical applications are discussed in further detail in Appendix 3.4, to determine which optimization problem solutions are most developed and suited for the benchmark.

3.3.1. Max-kSAT and max-kXOR

In max-kSAT the objective function needs to be maximized, as discussed in Section 2. Max-kSAT is the general set of problems where the number of Boolean constraints satisfied is maximized. Individual constraints can have their own weight. Giving a weight to constraints can be beneficial for certain problems. For now the unweighted problems are examined, i.e. weight = 1. There exist problems where not all constraints can be satisfied. In these cases the maximum number of constraints, but not all need to be satisfied. The problem can take the form of any constraint that applies to k parameters. Currently, the most common max-kSAT problem discussed, is the max-kXOR. For this variant of max-kSAT problems, the objective function is defined as [74]:

$$C = \frac{1}{2} \pm \frac{1}{2} \prod_{i=1}^{k} x_i \tag{3.9}$$

3.3.2. MaxCut

This is by far the most discussed problem with regard to QAOA [8, 29, 42, 53, 74, 115, 119]. The MaxCut problem is a max-2XOR problem, that can be visually represented as a graph. The individual parameters are presented as vertices, while the clauses can be represented as edges. The objective of MaxCut is to find as

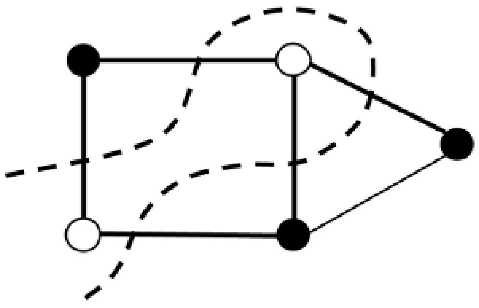


Figure 3.5: A MaxCut problem visualized as a graph [75].

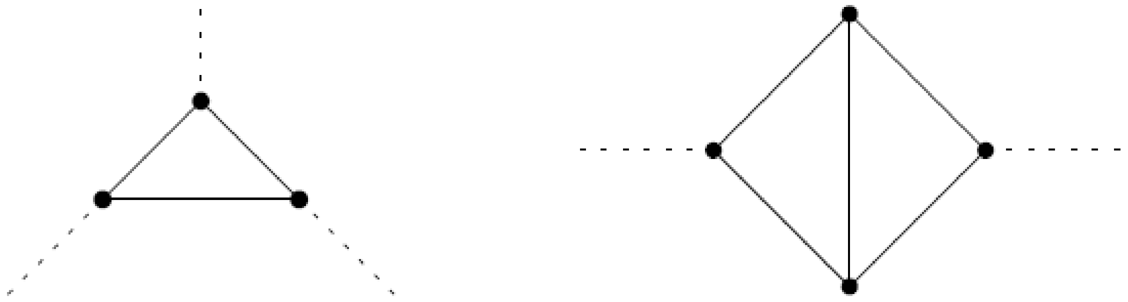


Figure 3.6: Possible sub-graphs for 3-regular graphs [42].

may edges $< jk >$as possible, where $j = (1 - k)$. A visualization is shown in Figure 3.5.

MaxCut is a known NP-complete problem [47, 67] and therefore serves as a good example of what can be accomplished with QAOA. The performance of MaxCut can be increased with classical pre-processing. If the input graph has a regular shape, e.g. with a bounded degree, the graph can be divided into sub-graphs that occur multiple times. This way, only the unique graphs need to be analyzed, and their expectation value multiplied with their corresponding occurrence [42]. An example is the 3-regular graph, which only has two possible sub-graphs, as shown in Figure 3.6. With pre-processing, only the occurrence of the different sub-graphs need to be found, and the QAOA needs to find the expectation value of the 2 possible graphs. This however implies that the classic algorithm needs to find the occurrences of the sub-graphs with high performance. How this carries over to practical applications is not yet explored, and will be very application dependent. A mayor concern is that finding the subgraphs, known as the clique problem, is in itself one of Karp's NP-complete problems [67].

MaxCut can be applied to challenges as circuit layout design [15], data clustering [91], network design and statistical physics [113]. This shows that MaxCut can be applied in a large variety of industry level optimization problems.

3.4. Quantum algorithm technology readiness level

The QAOA implementations of the above discussed problems will be summarized and compared according to their technology readiness level (TRL). This is done, as not every paper provides an in-depth analysis of the proposed implementation. Providing an in-depth analysis of every implementation would be beyond the scope of this thesis. TRL was first introduced by NASA [76] to evaluate the state of development of technologies. The NASA TRL has 9 levels, from proposing basic principals (1) to flight proven (9). Currently, more adaptions have been made for both European projects and commercial projects [56].

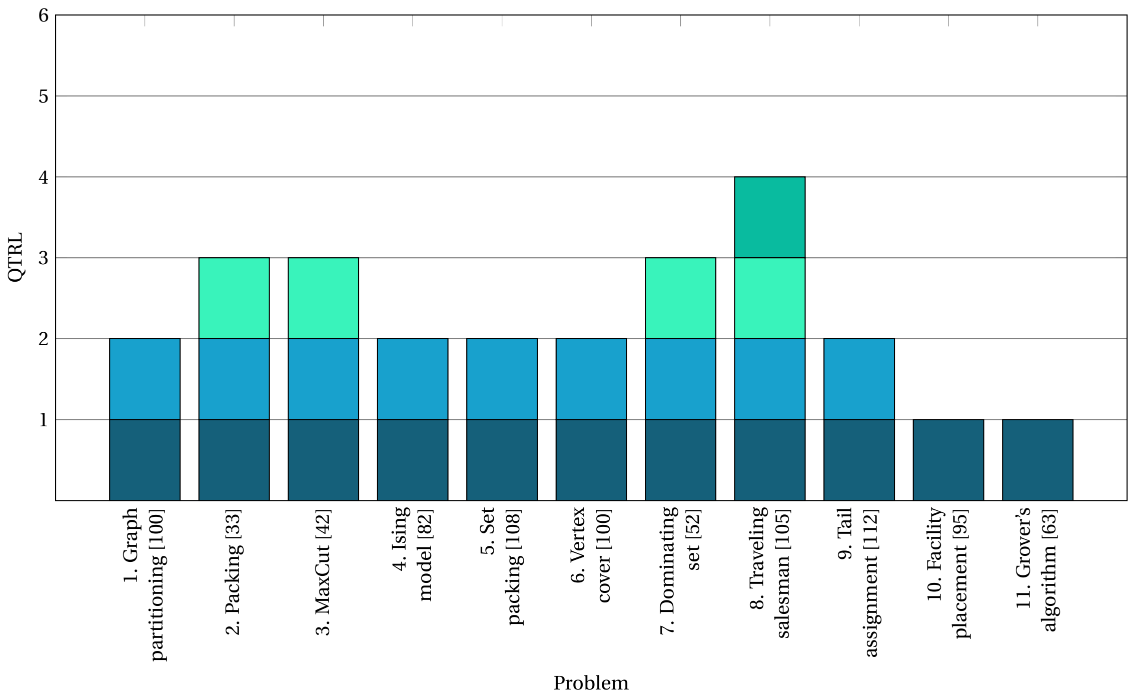


Figure 3.7: A quantum technology readiness level (QTRL) evaluation of the presented QAOA implementations.

In order to evaluate the maturity of quantum algorithm implementations appropriately, we propose a novel quantum TRL (QTRL). This QTRL is inspired by the TRL as presented by NASA, but adapted to quantum algorithms. The QTRL is in this chapter used to evaluate the presented QAOA implementations, but can be used to evaluate quantum algorithms in general. The QTRL are ordered as follows:

1. **Theoretical proof of concept is presented**: The quantum algorithm to a specific problem is fully worked out and a theoretical proof is given that this implementation approximates or yields the optimal solution to the problem. *In the example of QAOA, this means that both the Cost and Mixer Hamiltonian are presented to the specific problem, as well as proof of the approximation.*

2. **Demonstrated on quantum simulator**: The quantum algorithm is implemented on a quantum simulator and provides meaningful results. *For QAOA, this means that the simulation should return values that to reasonable extend approximate a valid a solution.*

3. **Cost and performance analysis**: An in-depth analysis of the implementation is done, showing the required resources and complexity. With resources is explicitly meant: qubits, circuit depth, number of each used gate.

4. **Simulated application**: The problem is implemented to a practical utility (application) with a classical algorithm that uses the quantum algorithm. The system should work with the combination of a classical computer and a quantum simulator. The full system should provide meaningful/valid results.

5. **Full stack application**: The full stack is presented with an implementation using quantum hardware and can be applied to problems in practice. The application will likely need optimization or clever implementation to out perform classical implementations.

6. **Quantum supremacy**: The problem can be solved faster on a classical-quantum hybrid full stack compared to a classical implementation, using the quantum algorithm. *For QAOA, this is currently an aim for the future, as no such implementations have been presented yet. Current efficient algorithms, such as QAOA for MaxCut, still require hundreds of qubits to achieve quantum supremacy. With the current state of the NISQ hardware, systems that can support such large numbers of qubits are not available.*

The QTRL of the problems is presented in Figure 3.7. For each of the presented algorithms, a proof of concept for the QAOA implementation has been presented. Except for the facility placement problem and the QAOA for Grover's algorithms, the algorithms have been tested on a quantum simulator, with published results. On only four of the algorithms, a cost and performance analysis has been published in literature: Packing [108], MaxCut [42], Dominating set [52] and Traveling salesman [105]. These are viable implementations for QPack. We have decided to focus on 3 of the possible canidates as the benchmarks, namely MaxCut problem, Dominating set problem and traveling salesman problem. Most notably, the Traveling salesman problem has been presented on a full stack, applied to a practical problem [105]. It is therefore the only algorithm at time of writing that can be considered QTRL-4. Some notes on the results are:

- The MaxCut algorithm has been presented on a full stack quantum computer [55], but not in the context of a practical application.
- Publications on the Ising model have analytic results on complexity, but it is considered not feasible and no resource analysis is done [5, 82]. As such it cannot be considered QTRL-3.
- Analysis has been done on the complexity of the set packing implementation, but no in-depth results on resources have been published [100].
- The traveling salesman has been implemented in full stack and on a practical application, but is not ready to work on available quantum hardware [105].

4
Benchmarking

As mentioned before, a popular metric to measure the performance of quantum computers is its quantum volume: the number of qubits and the quantum circuit depth it can run. The authors of the quantum volume benchmark mention in total four metrics to which the performance of a quantum computer can be evaluated [18]:

- Number of physical qubits
- Number of gates that can be applied before errors occur (quantum circuit depth)
- Connectivity of the quantum computer
- Number of operations that can be run in parallel

The number of physical qubits is only of interest if a required circuit depth, as such these metrics can be combined in the quantum volume. As argued before, quantum volume on itself does not reflect practical performance properly. Furthermore, the qubit connectivity and number of operations that can be run in parallel certainly have value as metrics, however, it can again be argued whether these metrics have practical value. For example, a fully connected quantum computer is good on paper, but if it cannot scale up it cannot be put in practice. From the perspective of this thesis, it is considered that execution with limited connectivity will be handled by the compiler (additional qubit SWAPs to enable execution on limited connectivity). As a result, the connectivity will affect the number of quantum operations and with that the run time and outcome accuracy (increasing SWAP-gates and in turn circuit depth will give rise to more noise, decreasing outcome accuracy). An important characteristic of QPack, is that it will use non-native problems, meaning that the problem does not correspond to the qubit layout. This means more resources are required to compile the problem to the quantum computer's hardware layout, which significantly impacts performance [55], but is essential for practical application. Similarly, the number of operations run in parallel will decrease the critical path of the circuit and reduce the runtime. With this, both metrics can be abstracted by evaluating the runtime. As quantum computers have progressed to a size that can run practical algorithms such as QAOA and VQE (variational quantum eigensolver), a higher abstraction level using application performance would be the proper method to evaluate the performance of current quantum computers.

A very recent benchmark QASMbench [73] has been presented by Li et al. This benchmark uses the OpenQASM assembly representation and has been used to benchmark several IBM quantum computers. While this benchmark is valuable to benchmark quantum chips, it is too low level to benchmark the quantum computing stack. Understandably, such benchmarks are of use to the development of quantum chips. However, the benchmarking of the full stack will be required with long-term benchmarking in mind. Especially with the current popularity of hybrid quantum algorithms, the quantum chip as well as its classical counterpart and the interaction between them must all be included in performance benchmarking.

Another proposed benchmark is the BB84 protocol, used in quantum communication [120]. The BB84 protocol uses only two qubits and can therefore be applied to the majority of currently available quantum hardware. It can, however, be argued that the algorithm in itself does not show a large variety in gates and that hardware can be optimized to these operations. It can therefore be considered a useful benchmark, if combined with other quantum algorithms.

A paper by Michielsen et al. discusses various quantum algorithms as candidates for benchmarking [81]. This set of benchmark problems in its implementation has shown informative results of IBM experience systems, but is again aimed at low level benchmarking rather than the full stack. The algorithms are purely quantum, whereas hybrid algorithms reflect current usage of NISQ quantum computers better, and show more insight to the quantum stack rather than the quantum computing chip in itself.

Earlier this year, Atos has announced the Q-score benchmark [12] that evaluates to which problem size a quantum computer can run an optimization algorithm for the MaxCut problem, to which the details are provided in Section 3.3.2. This QAOA-centric benchmark is a good first approach to establish meaningful quantum benchmarks inspired directly from applications. Atos' Q-score benchmark targets the following metrics:

- **Success-rate, accuracy**: How well is the algorithm executed
- **Performance**: How fast is the algorithm executed
- **Scaling**: How well does the algorithm perform on different problem sizes

We believe that further metrics of practical relevance should be included to create an even more insightful benchmark. First, we suggest a more fine-grained splitting of the total runtime into the different stages: (1) compilation, (2) classical computation, (3) communication between classical and quantum and (4) quantum computation. Each of these four stages targets a different aspect of the quantum computing stack and will give both quantum hardware providers and algorithm developers insight into the computational bottlenecks.

Another concern of the Q-score benchmark is its lack of universality. While it claims to run on every current quantum computer, it only uses a single problem: MaxCut. The danger of using a single problem is that quantum computers could be tailored to this specific problem but perform poorly with others. An extreme case is the D-Wave quantum computer, which is specialized to quantum annealing by design. This is by no means a criticism towards D-Wave, but illustrates how a single-problem benchmark could give an incomplete picture of a quantum computer's performance in general. QAOA is a versatile algorithm that can be applied on a large variation of problems with various practical applications. To avoid single problem tailoring, we suggest multiple different problems to benchmark the quantum hardware.

4.1. Benchmarking metrics

To create a standardized benchmark, a set of metrics needs to be agreed upon. In order to properly evaluate the performance of a quantum hardware implementation in practice, the following aspects need to be considered:

1. Run-time
   (a) Classical computation
   (b) Classical to quantum communication
   (c) Quantum computation
2. Outcome accuracy / best approximation error
3. Performance scaling

Run-time

The run-time of an application is an obvious parameter to measure. Classically, apart from the functionality, this is the main parameter measured as it separates implementations on their practical performance. Since current NISQ quantum hardware is only practical in classical-quantum hybrid configurations, both the classical and the quantum hardware need to be benchmarked. While the quantum speedup should be most significant to the performance, for QAOA optimization it was found that the classical optimizer plays an important role to the performance of the QAOA optimization [19, 105]. Determining the performance of the classical hardware is therefore as essential as evaluating the quantum hardware. For this reason the following run-times need to be observed:

- Overall run-time
- Run-time classical hardware
- Connection between classical and quantum hardware
- Preprocessing/placing and routing (quantum circuit compilation)
- Queuing time
- Run-time quantum hardware

The communication between quantum and classical hardware might very well be significant to the overall performance, as QAOA requires many calls to the quantum hardware. Since quantum computing is a Compute as a Service (CaaS) in current implementations, the way the connection is organized as well as the relative slow internet connection can be crucial. In cases that the implementation is organized with a local classical computer sending instructions to the remote quantum computer, most time will be lost by queuing and classical to quantum connection. Queuing time indicates time spend waiting for access to a quantum system. In regular IBM quantum computer calls, the call is put in a queue for access for each quantum circuit call (shots for the same call are executed consecutively). A more mature implementation would be both a remote classical computer and a remote quantum computer, which share a faster local connection. An example of such a quantum stack is the IBM experience run-time. In this case, the classical code is sent in combination with the quantum circuit, as a quantum job. This job in itself is queued, but not each individual call to the quantum computer. Rigetti offers a QaaS as well, but trough an API to log in to their QCS. Rigetti also allows the user to use a remote classical computer to communicate faster to the quantum computer. The service, in contrast to IBM, requires the user to schedule a time slot with access to the quantum computer. This in itself has its pros and cons.
By explicitly measuring these detailed timestamps, much more crucial information is learned about the bottleneck of an implementation. As the benchmark does not optimize quantum instructions to suit specific hardware, the performance of the compiler will be reflected as well and will translate to the number of instructions generated and how the quantum circuit is scheduled. Other bottlenecks on the implementation could include: number of calls to quantum hardware from classical hardware and time required per quantum instruction.

Outcome accuracy / Best approximation error

QAOA can only approximate an optimal solution. While the quality of the approximation depends on parameter p and the classical optimization strategy, it will also depends on the hardware. High quality quantum hardwarew can produce accurate results, but if the optimizer is not accurate the end results will be inaccurate as well, and vice versa. The qubit quality will determine the noise in a system and will determine p [7] which can significantly decrease the runtime performance. Furthermore, gate error and finite coherence time can introduce significant bounds on p, limiting not only compensation for noise, but also place significant bounds on scaling. Another effect on the accuracy could be $|0\rangle$ bias [52]. This means that the states are more likely to be measured as a logical 0 rather than a logical 1, skewing the outcome and effecting the accuracy. The approximation can be evaluated in multiple approaches.

1. The benchmark will be run on the hardware with optimal parameter p, determined iteratively. This way the maximum achievable accuracy can be found by comparing it to the solution of an exact algorithm. This has as downside that multiple runs will be required to find the correct parameter, and increasing p will increase the run-time, which might not be properly reflected this way. If the run-time of the implementation is however presented for only the optimal parameter, it will likely better reflect how the optimization would be implemented in practice. This way a decision will be made for the trade-off between speed and accuracy, favoring accuracy. The difficult part is that, in the case that there are no bounds on p, p might grow to infinity, approaching exact calculation. Recent research show that increasing p will not always improve accuracy [85] and will have a practical limit.

2. The benchmark will be run on hardware for, e.g., $p \in \{1,2,3,4\}$ and the results will be compared. This way the run-time will not be determined by the choice of p, but the accuracy will reflect the quality of the quantum hardware. This will require more interpretation, as the accuracy shown will not be the optimal accuracy, but the accuracy that is achieved for low p. The difficulty in this approach lies in determining how p scales for larger problem sizes. The size of p is expected to grow with the problem size, but at which rate is not yet determined. Some research suggests that QAOA will only be practical for low depth (e.g. p=2) which might eliminate this issue.
   From both approaches, the latter appears to be the most fair and with fewer issues, but demands an explanation on how to interpret the results.

3. Alternatively, an approach can be taken where the accuracy is evaluated for a predetermined run-time. This however requires that the run-time will not be constant for each problem size as the run-time is expected to grow. Determining a starting point for the run time as well as how this will progress for the problem size is nontrivial.

Of the three presented approaches, approach one is preferred.

Performance scaling

With concerns as bounds on p and scaling issues, it is important to give insight on how the benchmark performance scales for larger problem sizes. By determining the run-time for increasing problem size, the scaling performance can be reviewed. Similarly, it can be evaluated on how the accuracy or success probability decreases for larger problem sizes.

While this benchmark is focused on quantum hardware, it could be extended to quantum simulators, evaluating run-time and memory consumption. This could give valuable insight to developers to find the optimal simulator to work with.

Benchmarking modern high performance computing architectures has a large focus on energy consumption. This is expected become an important metric for quantum computing in the future as well. Energy consumption is, however, complex to benchmark as most of the measurements need to be done on location, and the division over classical and quantum computing adds another layer of complexity. In current quantum hardware there is a large variety of hardware implementation, making it even more difficult to standardize the measurement of energy consumption. For obvious reasons, this metric is not included in the benchmark, but quantum hardware developers are encouraged to accompany benchmark result with any estimate of energy consumption.

4.2. Benchmark configuration

In this chapter the details on the benchmark configuration are discussed. This includes a discussion on selecting a classical optimizer for finding the optimal $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ parameter, as well as details on the quantum implementation of the QAOA algorithms.

4.2.1. Parameter optimization

This section explores the strategies to optimally determine the $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ parameters. The optimization of these parameters is considered to be the main bottleneck of QAOA [105], and the selection of the optimizer could make or break the application. The optimization is a single objective real-parameter optimization and is done classically. Many algorithms have been developed for this problem. Critical parameters for choosing a classical optimizer are as follows:

- number of parameters to optimize
- size of the problem instance
- smoothness of the objective function
- number of local optima

Gradient-based optimization algorithms
Gradient-based optimization uses the gradient of the problem function to find a (local) optimum. Gradient-based optimizers converge generally faster than gradient-free ones and would therefore be favorable [9]. However, gradient-based optimization only works well on smooth functions and is prone to getting stuck in local optima. Due to the dependency on smooth functions, any function that is either discontinuous or noisy will cause problems for the optimization. In the case that a smooth function is available, the problem of local minima could be averted by choosing multiple starting points and running the optimization multiple times or in parallel if possible. This, however, increases the computational complexity and choosing an efficient algorithm for the starting value is not trivial. Often random points are chosen, but a more efficient algorithm based on the Discrete Fourier Transform (DFT) is presented by [119] but is not open-source.

Gradient-free optimization algorithms
Gradient-free algorithms do not depend on the gradient of the function, as the name suggests. This means that these algorithms can be applied to objective functions that are noisy, discrete or discontinuous [10]. Gradient-free methods can be more resilient with respect to local minima, depending on the algorithm. The downside of these algorithms is that these generally take more computation time or more iterations.

Problem instances
The objective function is evaluated with the above mentioned critical aspects, in order to find the optimization algorithm that fits best. The objective functions are considered in general and not per individual problem. The reason for this will become apparent in the evaluation.

Number of parameters: For all problems, the number of parameters are $2p$: β_p and γ_p for each QAOA iteration p. Optionally, p can be taken as a parameter as well, but since publications show that QAOA is most effective in low depth, i.e. low p, this parameter is expected not to contribute significantly. Generally a value for p is chosen beforehand (e.g. Crooks [29] shows that $p \geq 8$ is required to outperform the Goemans-Williamson algorithm) after which β and γ are optimized. Theoretically, an optimizer would increase p to infinite as this would give the best results. In practice, p can optionally be included as well, as performance is likely to decrease with higher circuit dept on quantum hardware.

Problem size: The problem size is expected to grow very large, as scalability needs to be considered. This does not differ per problem type.

Smoothness of objective function: The smoothness of the objective function is expected to differ per problem type. However, due to the probabilistic nature of the results, as well as noise in quantum hardware, the objective function is never smooth in practice.

Local optima: The presence of local optima is dependent on each individual problem, but due to noise in quantum hardware, local optima might still occur. Local optima might not be as strong as local minima inherent to the objective function. This could give room to optimizers that have some trouble escaping from strong local optima.

Selection of optimizer
Considering how all QAOA problem instances will suffer from noise, discontinuities and local optima, no gradient-based optimizer can be applied. From the gradient-free algorithms the following are considered:

Nelder-Mead Simplex: Nelder-Mead is efficient for non-smooth objective functions. It is, however, not very efficient for 10 or more design variables [58]. Considering that only $2 \times p$ variables are optimized, the Nelder-Mead optimizer fits well for low-depth QAOA. Nelder-Mead is also more resilient with regard to local optima compared to other local optimizers.

Genetic Algorithms: Genetic algorithms generally require many function evaluations, but are applicable to multi-objective optimizations [58]. Genetic algorithms are not well-suited, since QAOA problems only have one objective function. The large number of iterations are therefore inefficient.

Simulated Annealing (SA): Simulated annealing has more freedom than other gradient-free optimization algorithms, but comparison to other algorithms shows that the computation per iteration of SA is less efficient. Alternatively, SA could be accelerated with VQE, but the performance is not yet adequately explored and will be more demanding for the quantum hardware. Potentially, this could be used in the future when quantum hardware is more mature and scalable.

Particle swarm optimization (PSO): Particle swarm optimization can converge fast, has a short computational time and has very few parameters to adjust [58]. The downsize is that it is prone to get trapped in local optima, especially with complex problems [3]. Since large problem instances need to be considered with regards to scalability, PSO might not be suited for QAOA. PSO could be considered for problem instances with no inherent local optima (i.e. local optima due to noise). Unfortunately, it could be difficult to tune initial parameters for PSO. Considering that QAOA should be applied to multiple different problems, tuning the parameters could create a bottleneck. PSO could be considered an option for similar problems and a custom PSO parameterization. A custom PSO could be considered for the Ising model, with seemingly no inherent local optima [88], but not for the more popular and more developed MaxCut [68].

Symmetric Rank-1 Update Method (SR1): The SR1 method is not a gradient-free algorithm, but is included for special cases. SR1 uses a first-order derivative, as opposed to a second-order derivative like the Newton method. This means that the smoothness requirement on the objective function is less strict. In the case that the hardware has sufficiently low noise and multiple measurements per iteration are performed, the SP1 method might be applicable. The main difficulty, is that it is subjective to how smooth an objective function needs to be for SP1 to perform well, as no clear comparison is done between noise levels in objective functions and performance of the algorithm. It would also require the objective function to be evaluated, meaning it would be unclear per implementation if the algorithm works well. Quasi-Newton methods such as SP1 and the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) only work for finding local minima. This means that if multiple local minima exist in the solution landscape, the global minima might not be found. With this observation in mind, SR1 can only be applied to objective functions with only one minimum and would require extensive evaluation of the objective function. Another Quasi-Newton method that is commonly used is the BFGS algorithm, but as long as the problem field has a limit constraint (in this case $\gamma, \beta \in [0, 2\pi]$), SR1 converges faster.

More state-of-the-art algorithms exist, as competitions in the field of single objective real-parameters optimization are organized by IEEE yearly (CEC Real-Parameter Optimization Competitions) but since the winning algorithms have not been found as either open-source or a library, the winning algorithms are not considered for application. Regardless, the best algorithm for low dimensions was found to be UMOEA [121].

From the evaluated optimizers, Nelder-Mead is considered the best fit for QAOA. Numerical comparison of classical optimizers [77] show that in terms of performance, BFGS and Nelder-Mead differ very little in runtime and find identical results in sufficiently smooth functions. This raises the argument that if the objective function is sufficiently smooth, Nelder-Mead could be used as well and no custom optimizer needs to be applied. This saves development time for the application. PSO or SR1 could be considered only for very special cases. For the general case, UMOEA could be implemented to possibly further improve performance.

4.3. Measurements

In this section, measurements of the different classical optimizer options and the different QAOA implementations are presented. These measurements were done using Qiskit qasm simulator, to show relative performances of classical parameter optimizers and QAOA applications.

4.3.1. Measurements of classical optimizers

In section 4.2.1, a preliminary study of various classical optimizers was done. In the current section, various promising classical optimization algorithms will be applied for increasing problem size to compare the runtime performance and accuracy. Apart from Nelder-Mead and BFGS, which are both discussed in Section 4.2.1, other optimization algorithms available in open-source libraries are presented as well. The list of tested optimizers is:

1. Local optimizers
   (a) Nelder-Mead (nm)
   (b) BFGS (Broyden–Fletcher–Goldfarb–Shanno)
   (c) NEWUOA (New Unconstrained Optimization Algorithm) [93]
   (d) BOBYQA (Bound Optimization By Quadratic Approximation) [94]

   (e) COBYLA (Constrained Optimization By Linear Approximation) [92]

2. Global optimizers
   (a) Dual annealing [116]
   (b) SHGO (simplicial homology global optimization) [37]
   (c) ISRES (Improved Stochastic Ranking Evolutionary Strategy) [103]
   (d) MLSL / MLSL_LDS (Multi-Level Single-Linkage, low-discrepancy sequence) [66]
   (e) DIRECT (DIviding RECTangles) [64]
   (f) StoGO (Stochastic Global Optimization) [118]

These algorithms were implemented in Python using either SciPy or the NL-Opt library [1]. Some of these algorithms are not included in the following results, as either no convergence was found (due to various reasons) or if the optimization had such a long run-time that they cannot be considered a contender in comparison to other algorithms. The algorithms included in this list are:

- Dual annealing
- NEWUOA
- BOBYQA
- ISRES
- MLSL / MLSL_LDS
- DIRECT
- StoGO

Perhaps with different implementations, these algorithms could be used for QAOA optimization. In the case of dual annealing, it was clear that convergence could be found, but takes significantly longer than other tested optimization algorithms. The MLSL implementation suffers from searching for every local optima (which can be very significant with noisy QAOA results). A variant of MLSL might work for QAOA, but with the current implementation no convergence could be found in reasonable time.

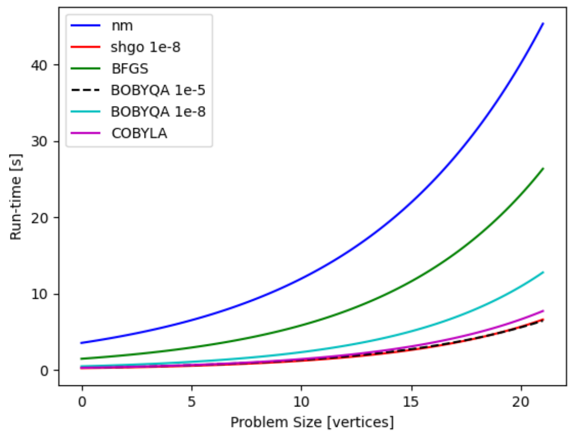


Figure 4.1: Data of run-time measurements fitted to exponential functions. Any number in the legend indicates the convergence tolerance, which is default if not indicated.

The run-time measurements were done on MaxCut problems with 3 to 23 nodes. The QAOA algorithm is implemented using Qiskit for the Qasm simulator [2] The MaxCut problems are randomly configured, but

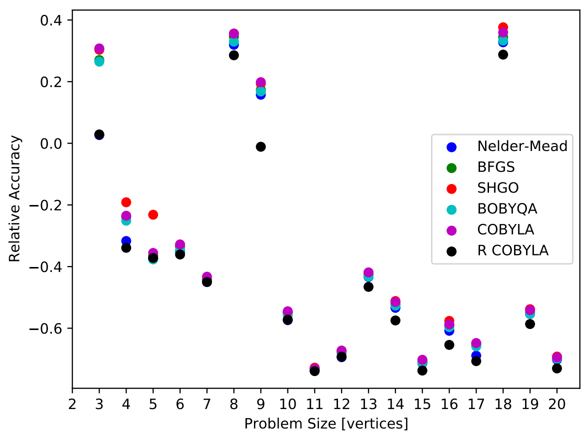


Figure 4.2: Optimizer accuracy compared to Dual-Annealing for the same MaxCut problem instance. Here a lower relative accuracy shows a more accurate algorithm. A negative value shows that the tested algorithm outperforms the Dual-Annealing algorithm. R COBYLA here indicates the COBYLA algorithm with multiple randomized starting points.

have the same number of edges as nodes to maintain the problem complexity. The measured run-times are shown in Figure 4.1. The data is fitted to an exponential function, as the obtained data shows a clear exponential growth. In this figure, it can be seen that the Nelder-Mead algorithms which was expected to perform best, is in fact the slowest from the selected algorithms in practice. SHGO, COBYLA and BOBYQA perform much better in terms of run-time, with SHGO being the fastest.

For the accuracy measurements, the different algorithms solve the same MaxCut problem for different sizes and the best expectation values are compared. Preferably, a fine grid search (brute force) is implemented as reference, but this proved to be too time consuming to collect data, even for small problems. In Figure 4.2, the found expected values are compared to the value found by the Dual-Annealing algorithm. Dual-Annealing was chosen, since it was a slower global optimizer, but allowed for reasonable computing times and expecting that its accuracy would outperform the faster algorithms. The difference in accuracy is calculated as

$$relative\ accuracy = \frac{DA\ expectation\ value - tested\ algorithm\ expectation\ value}{DA\ expectation\ value}$$

. The results however show that Dual-Annealing does not always find better solutions and is often outperformed by other contenders. The β and γ parameters are not compared, as different local minima might find values very close to the optimal value. The results show clearly that COBYLA using randomized starting points outperforms other algorithms, indicating that a hybrid approach finds the best solutions. This, however, significantly increases the search time. For the single algorithm approaches, Nelder-Mead finds the best expectation values, presenting a trade-off between accuracy and time. It should be noted that for larger problem sizes, the expectation values are much closer and it is expected that for scaling purposes it is better to choose a faster algorithm such as COBYLA, BOBYQA of SHGO. Some parameters such as the number of the QAOA iterations or the convergence tolerance are not optimized as this is expected to give minimal change to the results with respect to the comparison.

Considering the found results in terms of accuracy and run-time, a hybrid approach could be applied. Heuristics such as random starting points combined with a local optimizer are often applied, to counteract the local optimizer of getting trapped in local optima. Another implementation is the use of a (fast) global optimizer with a more accurate local optimizer. In the case of optimizing QAOA parameters, using a fast algorithm such as SHGO combined with, e.g., Nelder-Mead or BFGS would be an interesting approach. Furthermore, many combinations are possible of both local and global optimizers. This approaches however did not provide enough evidence to indicate better performance compared to random starting points. The increase of starting points in itself proved to increase the accuracy, but the method did not affect the accuracy significantly.

4.3.2. Measurements of applications

A common downside of computer benchmarks is that hardware vendors are tempted to tailor their systems to excel in the particular benchmarks. We have therefore included multiple applications to the QPack benchmark. Next to the MaxCut problem (MCP), the dominating set problem (DSP) and the traveling salesman problem (TSP) have been included. In this section, the details on the QAOA implementation will be discussed, as well as the effect on the measured performance.

MaxCut

The MaxCut application has been examined in detail in Section 3.3.2. The QAOA implementation requires the following gates for n vertices and m edges(or clauses) and p QAOA iterations:

Gate type	# gates
Hadamard	n
CNOT	$2m \cdot p$
RZ	$m \cdot p$
RX	$n \cdot p$

From the applications explored, MaxCut QAOA requires the least gates and is therefore expected to run best on NISQ devices. MaxCut results in the smallest circuit depth and does not require additional ancilla qubits. The implementation of the MaxCut circuit and the cost function evaluation are given below, in qiskit and Python.

```
# circuit definition for QAOA MaxCut
def max_cut_circ(params, graph, p, backend):
    beta = params[0:p]
    gamma = params[p:2*p]

    v, edge_list = graph
    vertice_list = list(range(0, v, 1))

    # initialise quantum circuit
    c = ClassicalRegister(v)
    q = QuantumRegister(v)
    qc = QuantumCircuit(q, c)

    # apply hadamard to each qubit
    for qubit in range(v):
        qc.h(qubit)

    #apply cost and mixer uinitary for each layer of p
    for iteration in range(p):
        for e in edge_list:
            qc.cnot(e[0], e[1])
            qc.rz(-gamma[p-1], e[1])
            qc.cnot(e[0], e[1])
        for qb in vertice_list:
            qc.rx(2*beta[p-1], qb)
    qc.measure(q, c)
    job = execute(qc, backend, shots=qaoa_shots)
    result = job.result()
    out_state = result.get_counts()
    prob = list(out_state.values())
    states = list(out_state.keys())
    exp = 0
    for k in range(len(states)):
        exp += eval_cost(states[k], graph)*prob[k]
    return [exp/qaoa_shots]
```

```
# cost evaluation for MaxCut
def eval_mcp(out_state, graph):
    v = graph[0]
    edges = graph[1]
    c = 0
    # convert values from outstate
    bin_val = [int(i) for i in list(out_state)]
bin_val = [-1 if x == 0 else 1 for x in bin_val]
# evaluate cost
for e in edges:
    c += 0.5 * (1 - int(bin_val[e[0]]) * int(bin_val[e[1]]))
return c
```

Dominating set

The QAOA for DSP is implemented according to the algorithm provided by Nicholas Guerrero [52]. This implementation uses logical OR gates, which can be implemented using CNOT gates and ancilla qubits. The number of required ancillas and CNOT per logical OR depends on the number of input qubits k. The number of required ancillas is $k-1$. The implementation by Guerrero uses the logical OR to control a RY gate. This OR-controlled RY gate requires one additional ancilla as target qubit. As the ancillas can be re-used, the ancillas per circuit add up to k ancillas, with $2 \leq k \leq n$. The logical 2-OR uses 3 CNOT gates, as depicted in Figure 4.3. These 2-OR gates can be combined to a n-input logical OR. For n inputs, this n-OR is implemented as:

```
qc = QuantumCircuit(2*n)
qc.append(ORGate, [0, 1, n])
for i in range(2, n):
    qc.append(ORGate, [i, n+i-2, n+i-1])
qc.crz(gamma, 2*n-2, 2*n-1)
for i in range(n, 2, -1):
    qc.append(ORGate, [n-i-1, 2*n-2-i, 2*n-1-i])
qc.append(ORGate, [0, 1, n])
```

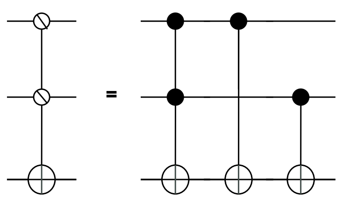


Figure 4.3: Implementation of a 2-input logical OR using CNOT gates.

The circuit can be implemented using the following resources (at most):

Gate type	# gates
Hadamard	n
X	$2m \cdot p$
CNOT	$((6n-5) \cdot m) \cdot p$
RZ	$m \cdot p$
RX	$n \cdot p$

n qubits are inititialised in superposition, using n Hadamard gates. Inverted controlled RZ-gates and OR-controlled RZ-gates are used to implement the m clauses [52]. The n RX gates are then applied to the n qubits. Excluding the initialization, this is repeated for p iterations.

The DSP QAOA implementation uses similar, but significantly more resources than the MCP implementation. The DSP implementation uses $(6n-7) \cdot m \cdot p$ more CNOT gates and $2n \cdot p$ additional Pauli-X gates. This requires quantum computers to support larger circuit depth, but also more ancilla qubits and better CNOT mapping. The additional CNOT requirement will reflect the qubit connectivity of the quantum hardware implementation, as well as the handling of qubit SWAPS (if required) by the hardware scheduler. The code implementation for the DSP circuit and cost evaluation are given below, again using Python and qiskit. Some of the larger gates, which are not native to qiskit, are omitted in the code below. These are presented in Appendix A.

```
def dsp_circuit(params, graph, p, backend):
```

```
# quantum circuit for dominating set
    beta = params[0:p]
    gamma = params[p:2*p]
    v, edge_list = graph
    vertice_list = list(range(0, v, 1))
    # initialise a list of the connections (list of nodes, connected to the node)
    connections = []
    for i in range(v):
        connections.append([i])
    for t in edge_list:
        connections[t[0]].append(t[1])
        connections[t[1]].append(t[0])
    ancillas = 0
    for con in connections:
        if len(con) > ancillas:
            ancillas = len(con)
    n = v+ancillas         # add ancillas
    qc = QuantumCircuit(n, v)

    # initialise with hadamard to each qubit
    for qubit in range(v):
        qc.h(qubit)
        # inverted  controlled rotation Z gate
        qc.x(qubit)
        qc.crz(-gamma[0], qubit, n-1)
        qc.x(qubit)
    # cost unitary and mixer unitary for QAOA layers (p)
    for iteration in range(p):
        f = 0
        f_anc = v
        for con in connections:
            c_len = len(con)
            # n-qubit controller rotation Z gate (OR_nrz)
            OR_range = con.copy()
            for k in range(c_len-1):
                OR_range.append(v+k)
            cOR_rz = OR_nrz(c_len, gamma[p-1])
            OR_range.append(n-1)
            qc.append(cOR_rz, OR_range)

        for qb in vertice_list:
            qc.rx(-2*beta[p-1], qb)
    qc.measure(range(v), range(v))
    job = execute(qc, backend, shots=qaoa_shots)
    result = job.result()
    out_state = result.get_counts()
    return out_state
```

```
# cost evaluation dominating set
def dsp_cost(params, v, e, p, backend):
    graph = [v, e]
    beta = params[0:p]
    gamma = params[p:2*p]

    out_state = dsp_circuit(params, graph, p, backend)

    edge_list = e
    vertice_list = list(range(0, v, 1))
    connections = []
    for i in range(v):
        connections.append([i])
    for t in edge_list:
        connections[t[0]].append(t[1])
        connections[t[1]].append(t[0])
    total_count = 0
    total_cost = 0
    for p_it in out_state:
        for t in out_state:
            count = int(out_state.get(t))
            total_count += count
            bin_len = "{0:0" + str(v) + "b}"  # string required for binary formatting
            bin_val = t.format(bin_len)
            bin_val = [int(i) for i in bin_val]
            # cost T
            T = 0 (+1 if node not connected/dominated)
            for con in connections:
                tmp = 0
                for k in con:
                    tmp = tmp or bin_val[k]
                    if tmp:
                        T += 1
                        break
            # cost D (+1 for each node dominating)
            D = 0
            for i in range(v):
                D += 1 - bin_val[i]
            total_cost += (T+D)*count
    total_cost = total_cost/total_count
    return total_cost
```

Traveling salesman

The TSP QAOA implementation is a much more demanding quantum algorithm and with the current implementation also uses different resources. The implementation is done according to the github: `https://lucaman99.github.io/new_blog/2020/mar16.html` [25] which is one of the few examples of a clear implementation. The author however indicates that the state preparation is sub optimal and is therefore altered to suit the algorithm better. The TSP algorithm, as other QAOA algorithm, consists of a state preparation, a mixer Hamiltonian and a cost Hamiltonian. Whereas the MCP and DSP QAOA need n to $2n$ qubits for n vertices, this implementation uses n^2 qubits. Other variations exist, such as presented by Ruan et al. [101], using m qubits, which is at most $n(n-1)/2$ qubits for a fully connected graph. This implementation will be considered a future improvement. This however means that the current implementation requires significantly more qubits compared to MCP and DSP.

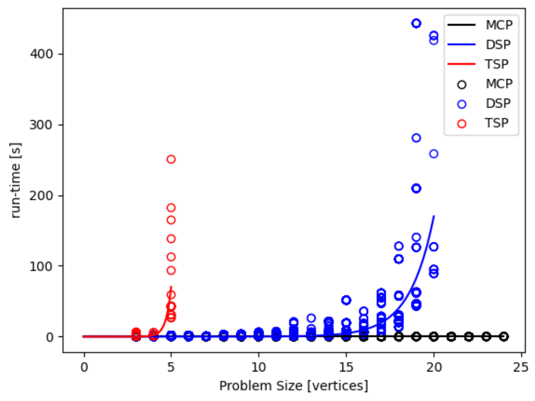


Figure 4.4: Runtime comparison for MCP, DSP and TSP QAOA applications, implemented in Qiskit, plotted in a single figure (100 QAOA runs were done for comparison).

For the state preparation n binary strings of length n are required with a Hamming weight of 2. The superposition of all possible strings with a set Hamming weight is called a Dicke state [35]. In the TSP implementation a Dicke state preparation is applied on each row of n qubits for Hamming weight 2, using the implementation by Bärtschi and Eidenbenz [23]. The cost and mixer Hamiltonian are left unchanged. Considering the significant cost of both the Dicke state preparation and the QAOA cycles, it worth mentioning the costs separately. The gate counts are provided in the following tables:

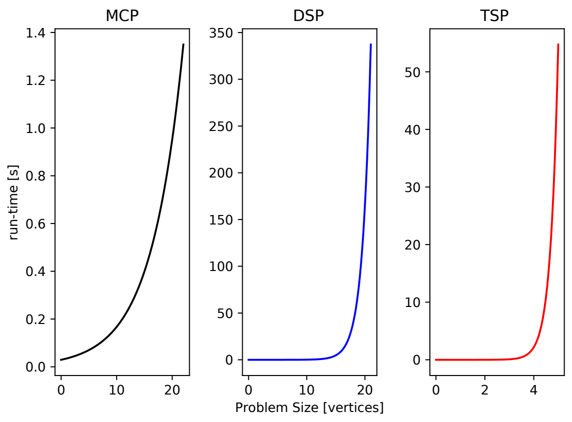


Figure 4.5: Runtime comparison for MCP, DSP and TSP QAOA applications, implemented in Qiskit plotted in separate figures for detail.

Dicke State Preparation Cost	
Gate type	# gates
Hadamard	$8n(n-2)$
$T/T^\dagger$	$49n(n-2)$
CNOT	$(31n-28)n$
RY	$(6n-10)n$
X	$2n$

TSP QAOA cycles Cost	
Gate type	# gates
RZ	$n^2 \cdot p$
RZZ	$\frac{n^2-n}{2} \cdot p$
RXX	$2n \cdot p$
RYY	$2n \cdot p$

This shows that while not only the circuit cost is significantly higher, many different gates are used. This will avoid fine tuning of quantum computers on solely Hademard, CNOT, RX and RZ Gates. To compare the impact of the different applications, a run-time comparison was done. The applications are implemented using the SHGO classical optimizer. The results in Figures 4.4 and 4.5 show that the run-time corresponds to the increase of circuit depth and required qubits as expected. Note that Figures 4.4 and 4.5 do not show large problem sizes for DSP and TSP as the larger problem sizes result in simulator memory issues. Despite the smaller data range, the difference in run-times can be clearly distinguished. The code implementation of the TSP circuit as well as the cost evaluation, are given below. The implementation is again in Python and qiskit and omitted gates are given in Appendix A

```
def tsp_circ(params, graph, p, rep, backend):
# QAOA circuit for TSP
    beta = params[0:p]
    gamma = params[p:2*p]
    v, A, D = graph

    vertice_list = list(range(0, v, 1))

    # initialize circuit
    n = v**2
    qc = QuantumCircuit(n, n)
    # initialization using the Dicke state preparation
    for q in range(v):
        q_range = range(q*v, (q+1)*v)
        DickeGate = dicke_init(v, 2)
        qc.append(DickeGate, q_range)
    # QAOA cycles
    # for 1 to p:

    # cost unitary
    for iteration in range(p):
        for i in range(n):
            qc.rz(gamma[p-1]*D[i]/(2*pi), i)

        for i in range(v):
            for j in range(i):
                if i != j:
                    qc.rzz(20*gamma[p-1]/pi, (j+i*v), i+j*v)   #RZZ on reflection over the diagonal

        # mixer unitary
        for i in range(0, v):
            qc.rxx(-beta[p-1], i*v, (i*v+1))
            qc.rxx(-beta[p-1], (i*v+1), (i*v+2))

            qc.ryy(-beta[p-1], i * v, (i * v + 1))
            qc.ryy(-beta[p-1], (i * v + 1), (i * v + 2))

        qc.measure(range(n), range(n))
        job = execute(qc, backend, shots=rep)
        result = job.result()
        out_state = result.get_counts()
    return out_state
```

```
# TSP cost evaluation
def cost_tsp(params, graph, p, rep, backend):
    v, A, D = graph
    # measured state formatting
    out_state = tsp_circ(params, graph, p, rep, backend)
    total_cost = 0
    coupling = []
    for i in range(v):
        for j in range(i):
            if i != j:
                coupling.append([i + j * v, j + i * v])
    for t in out_state:
        count = int(out_state.get(t))
        bin_len = "{0:0" + str(v) + "b}"  # string required for binary formatting
        bin_val = t.format(bin_len)
        bin_val = [int(i) for i in bin_val]
        # cost evaluation
        cost = 0
        for i in range(0, v):
            for j in range(0, v):
                cost += 0.5*D[i + v*j]*bin_val[i + v*j]
        for j in coupling:
            cost += -5*(1 - 2*bin_val[j[0]])*(1 - 2*bin_val[j[1]])
        total_cost += (cost*count)/rep
    return total_cost
```

4.4. Implementation

Our QPack benchmark provides reference implementations for the aforementioned problems, can be run for increasing problem size, until the quantum hardware is unable to find a solution. The QAOA runs themselves will need multiple runs to determine the average run-time and accuracy. The solutions found by the QAOA optimization, need to be compared to a pre-computed answer using a classical exact algorithm. If the hardware supports larger circuit depth, the QAOA algorithm can be adjusted to a maximum p. As theoretically the best solution is always larger or equal to solutions achieved with smaller QAOA depth p, a maximum value can be found when the found solution does not improve or diminish. Diminishing of the solution indicates that no larger circuit depth is possible. This search to a maximum p can be included as an additional step for the benchmark to find the best accuracy of the QAOA implementation. The maximum circuit depth and width of quantum hardware will translate to maximum resources per available memory for a quantum simulator.

As the benchmark needs to be reproducible, the configuration of the problems must be predetermined. Akshay et al. [5] show that the performance of QAOA in terms of configurations is mainly due to the density (e.g. in a graph: the number of edges of a graph relative to the number of vertices). As the benchmark needs to scale to any problem size, the problems configurations cannot be hard-coded. In the proposed benchmark, the density will remain the same, to make the performance predetermined.

In order to create a scalable problem set with continuous edge density, a 4-regular graph will be used. Specifically, the 4-regular graph will have a structure similar to Figure 4.6. This however is not necessarily representative of practical applications, but will ensure reproducability and scalability without requiring

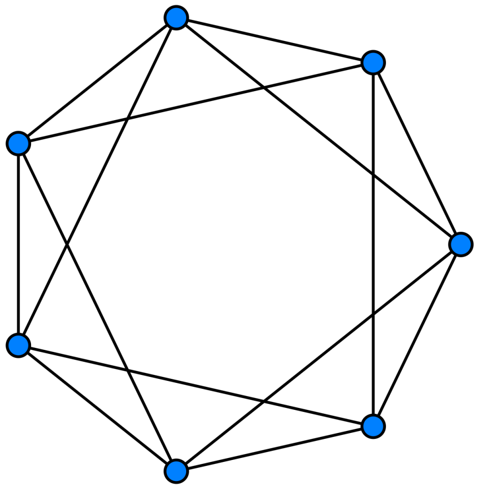


Figure 4.6: Structure of the circular 4-regular graph. Each node is connected to its neighbour, as well as the node next to the neighbours. The figure shows an example of a 7-node graph for such a structure.

score adjustments to random generated graphs.

The number of "shots" for a single QAOA iteration will range between 1000 and 10000, as these give the best results [119]. The number of iterations to achieve an estimation of the accuracy must be further analyzed. Expected is that the estimation will saturate towards a maximum reliability. A lower bound must be determined for this reliability. Determining the parameter p is also not trivial. There are arguments that the accuracy of QAOA will not grow for $p > 2$ [78], but others argue that e.g. a minimum of $p \geq 8$ is required for the MaxCut application to compete with classical approaches [29]. Determining p, possibly dependent on the application will require further numerical analysis.

The outline for QPack is given schematically in Figure 4.7. Here, the QAOA applications are run for increasing problem sizes. The benchmark will stop if for all applications the maximum problem size has been reached. The maximum is determined when not enough qubits are available to evaluate the problem. Some design choices can be made to make sure that the benchmark is as complete as possible. First of all, the generation of problems can be done every iteration of the QAOA optimization. It would be fairest to create a new configuration of the problem, as certain configurations might be easier to optimize and might give an unfair advantage. This however will require longer computation times for the benchmark itself. Including a random configuration for each run however, might cause benchmark result to become incomparable, and a significant amount of time will be required to validate each generated graph. The number of QAOA runs is also debatable. More runs will give a better average, but a limit must be set to limit the run-time. Another interesting addition would be adding both weighted and unweighted problem instances for all benchmarks. This can show if the hardware can adjust to weighted edges with e.g. using parameter "sensitive" edges [84]. This solution would preferably be implemented by the benchmark, but requires too much information about the hardware, which should be abstracted. A quantum instruction compiler would be tasked to implement the qubit SWAPs required for this method. A further concern is that the benchmark should allow for parallelism. With the current size of quantum hardware, it is unlikely that this will be fully exploited in practice. However, as quantum hardware will scale up, there should be support to apply the parallelism. For one of the viable optimizations algorithms, the Nelder-Mead classical optimization algorithm, a parallel variant has been developed [34, 72, 87]. This will allow the use of more quantum resources to compute smaller problems. The implementation of such a parallel implementation should be considered as quantum hardware can provide the required resources.

Finally, the implementation of the benchmark will require universality, in the sense that it should run on every quantum computer. The LibKet library [83] is suited for this task. It will, however, take more work to fully adjust the benchmark to this library. Another option would be the XACC library [80], which is conveniently targeted towards hybrid computing. This library provides both a Python and a C++ implementation. XACC supports IBM, Rigetti, IonQ and D-Wave QPUs (and several quantum simulators), whereas LibKet has larger support but is not fully developed for hybrid quantum computing yet. To demonstrate an universal benchmark, an implementation of QPack is presented at the **QPack XACC github repo** in collaboration with Huub Donkers. Using this implementation, several quantum simlators have been benchmarked which will

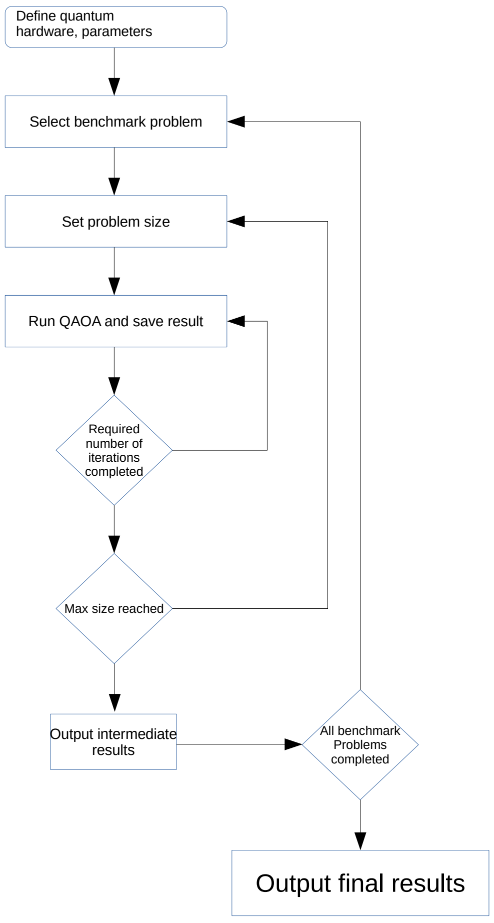

Define quantum hardware, parameters
Select benchmark problem
Set problem size
Run QAOA and save result
Required number of iterations completed
Max size reached
Output intermediate results
All benchmark Problems completed
Output final results

Figure 4.7: Flowchart for the proposed QPack benchmark.

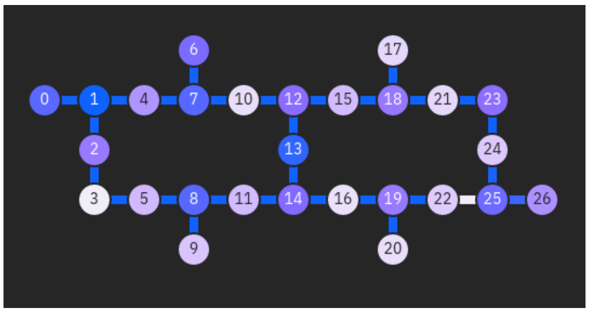
be presented in Chapter 5.

To give the reader an idea of using the qiskit implemented benchmark. a code fragment of the benchmark is given below. For this code fragment all account details should be filled by the user, as well as the backend they would like to use. The function *Benchmark* allows the entries *'mcp'*, *'dsp'* and *'tsp'* for initializing the different benchmarks. The function *set_lim(int)* sets the largest problem instance size the benchmark will run. The function *set_p(int)* will set the number of QAOA iterations p, which defaults to one. Finally, the function *run(void)* will run the benchmark for increasing problem instance sizes up to a (user defined) limit. The results are then presented in JSON format.

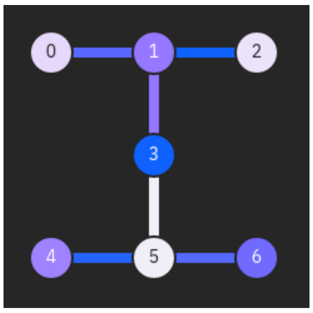
```
IBMQ.enable_account('...')
provider = IBMQ.get_provider(hub='ibm-q', group='open', project='main')
backend = provider.get_backend("ibmq_qasm_simulator")

#MCP benchmark
mcp = Benchmark('mcp')
mcp.set_backend(backend)
mcp.set_lim(10)
mcp.update_p(2)
mcp.run()
```

5 Results

In this chapter the results are presented for each benchmark discussed in this thesis. The results are shown for the remote IBM QASM simulator (32 simulated qubits), IBM Montreal (27 physical qubits, Figure 5.1a) and IBM Nairobi (7 physical qubits, Figure 5.1b). The IBM Montreal and Nairobi are not a publicly available quantum computer, but special access was given by the Strangeworks team. The fact that these systems are not publicly available, should greatly decrease the queuing time. For the IBM Nairobi, the IBM runtime environment is available, which supports the use of a local classical computer, in order to speed up communication for hybrid computing. Currently, IBM runtime environment is available for all their hardware, but at the time of testing it was only available for the IBM Nairobi. Both the score and runtime results are presented in their respective sections. The results have been obtained using 1000 shots per iteration, with the number of function evaluations of the classical optimizer limited to 1000. The results obtained are scaled up from problem size 5 up to a larger size that still allows a reasonable runtime.



(a) Qubit layout of the IBM Montreal, as presented on IBM experience.



(b) Qubit layout of the IBM Nairobi, as presented on IBM experience.

5.1. Run-time

The time measurements are made separately for each step of the quantum computing job execution process:

- Creating
- Validating
- Queued
- Running
- Connection/communication
- Walltime
- Other

"Creating" and "validating" are the overhead to submit the job, compile and validate the quantum circuit. The "queued" time presents the time the job needs to wait for the hardware to become available. The "running" time represents the time spent on the quantum hardware or simulator. The time in "connection" shows the time spent between the host sending the job and the framework running the job. The "walltime" is the time measured between starting the job and completing it on the host computer. The difference in time between "connection" and "walltime" is the time spent on the classical computer. Any time lost on other miscellaneous tasks is shown by "other".

MCP

This section presents the results of the MaxCut problem (MCP) benchmark. The times are shown cumulatively. The times are not normalized to the number of optimization steps. This is why the runtime for the simulator is significantly longer compared to the quantum hardware results, as the simulation has a higher limit on optimization steps. The results for the IBM QASM simulator are shown in Figure 5.2. Since the simulator is relatively fast for these small problem sizes, a significant portion is spent on the classical computation.

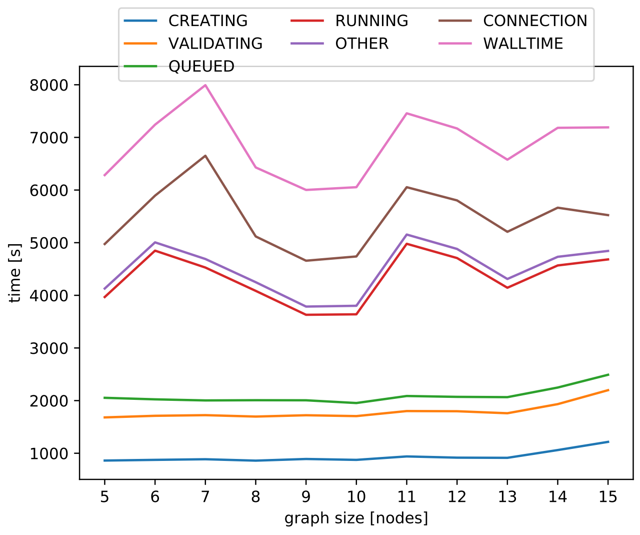
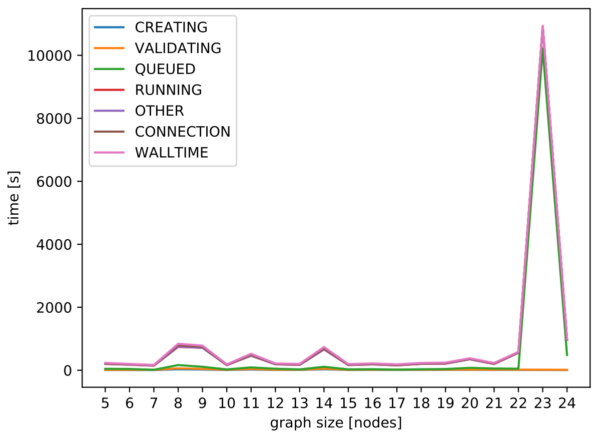


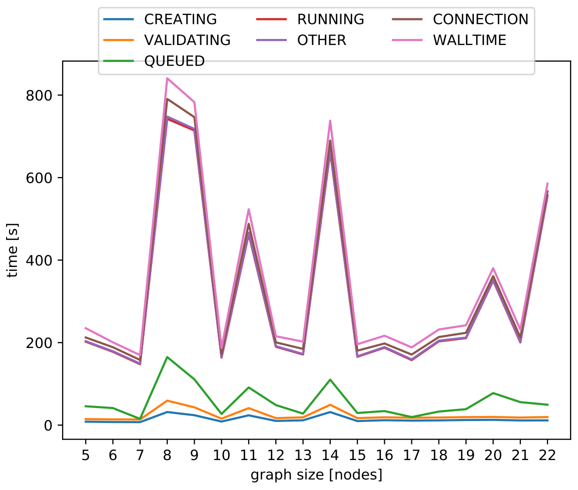
Figure 5.2: Results of the MCP benchmark for a problem size of 5 up to 15 nodes on the IBM QASM simulator.

The results for the MCP benchmark on the IBM Montreal are shown in Figure 5.3b and Figure 5.3a. For Figure 5.3b, the results for problem size 23 and 24 have been cut out, to focus on how the computation time is spent without queue spikes. These queue spike of about 10K s in Figure 5.3a for a graph size of 23 clearly shows the bottleneck for hybrid quantum computing without having a dedicated classical computer near the quantum computer or without exclusive access to a quantum computer. For most operations, the main part of the computation is running the quantum circuit as shown in Figure 5.3b. However, as shown by the results in Figure 5.3a, the computation time can be completely dominated by the queuing time.

The results for the MCP benchmark on the IBM Nairobi are presented in two parts: normal operation (Figure 5.4) and with the IBM runtime environment enabled (Figure 5.4b). The results of the IBM Nairobi can be compared to the Montreal hardware. The IBM Nairobi has been recently updated to support the runtime environment, and is expected to have increased in performance. The Nairobi hardware, however only support up to 7 qubits. The IBM Nairobi using the runtime environment (Figure 5.4b) unfortunately does not show significant speedups compared to the IBM Montreal (Figure 5.3a). By comparing the results of the Montreal results (Figure 5.3b) to the regular operation of the Nairobi hardware (Figure 5.4a), it can be observed that the Nairobi hardware is significantly slower compared to the Montreal hardware. Possibly, due to the limited number of qubits, more $SWAP$ gates are required compared to the Montreal hardware. Alternatively, the quantum chip for the IBM Nairobi might be optimized for fidelity, in return for slower execution time. Only when implementing the IBM runtime environment (Figure 5.4b), a performance similar to that of the Montreal hardware is achieved for graph size 5 to 7. By comparing the runtime environment (Figure 5.4b) to the regular operation on Nairobi (Figure 5.4a), a very significant speedup can be observed of 26.8 on average.

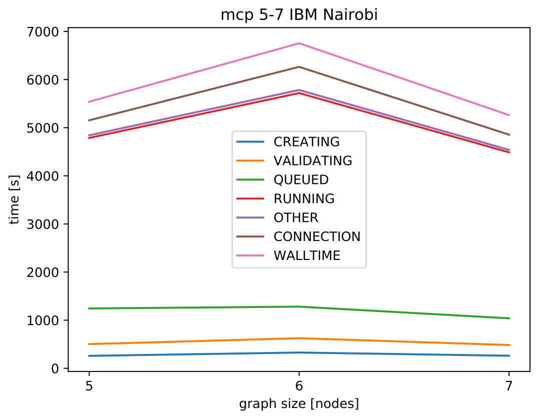


(a) Results of the MCP benchmark for 5 to 24 nodes on the IBM Montreal.

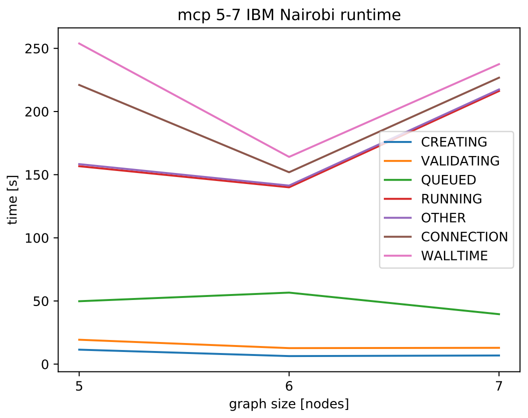


(b) Results of the MCP benchmark for 5 to 22 nodes on the IBM Montreal.

Figure 5.3: Run-times of the MCP benchmark on the IBM Montreal quantum computer



(a) Results of the MCP benchmark up to 7 nodes on the IBM Nairobi.

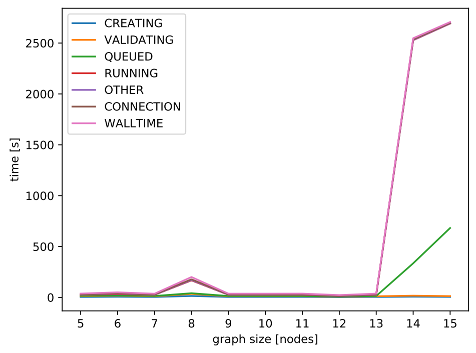


(b) Results of the MCP benchmark up to 7 nodes on the IBM Nairobi with runtime environment.

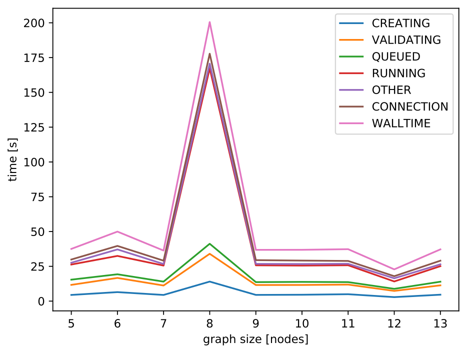
Figure 5.4: Comparison for the runtime environment on the IBM Nairobi using the MCP benchmark

DSP

The DSP results are presented for the IBM QASM simulator and the IBM Montreal. The results are not available for the IBM Nairobi hardware, as not enough qubits are present to run this benchmark. The runtimes for the DSP benchmark on the IBM QASM simulator are shown in Figure 5.5a. This figure shows large queue and run spikes at graph sizes 14 and 15. In order to show the results of graph sizes 5 to 13 in more detail, Figure 5.5b only shows the results for these sizes. The results for the IBM Montreal are presented in Figure 5.6a. This figure again shows large queue times for graph sizes 5 to 7 and 18 to 22. For this reason, Figure 5.6b only shows the results for graph sizes 8 to 17, for the same benchmark run. With the implementation of the dominating set problem (DSP), we can see that the runtimes are significantly higher on the quantum hardware compared to the MCP benchmark. While the results for the IBM Montreal hardware and IBM QASM simulator in Figure 5.5a and 5.6a are mainly dominated by queue times, the cut-out results in Figure 5.5b and 5.6b clearly show larger runtimes compared to the MCP benchmark on the (simulated) hardware. This result is to be expected, as the DSP QAOA circuit requires significantly more operations and qubits, as discussed before.

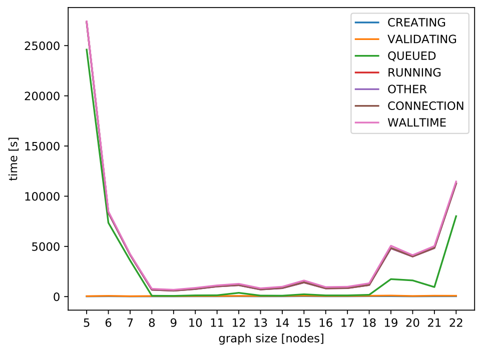


(a) Results of the DSP benchmark up to 15 nodes on the IBM QASM simulator.

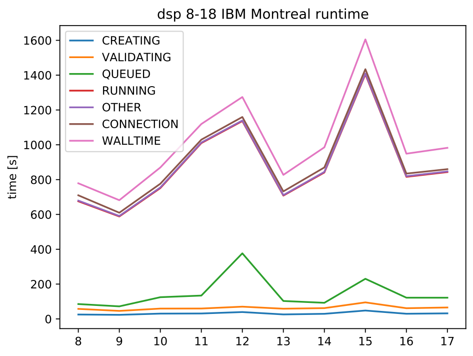


(b) Results of the DSP benchmark up to 13 nodes on the IBM QASM simulator.

Figure 5.5: Results of the DSP benchmark on the IBM QASM simulator



(a) Results of the DSP benchmark up to 22 nodes on the IBM Montreal.



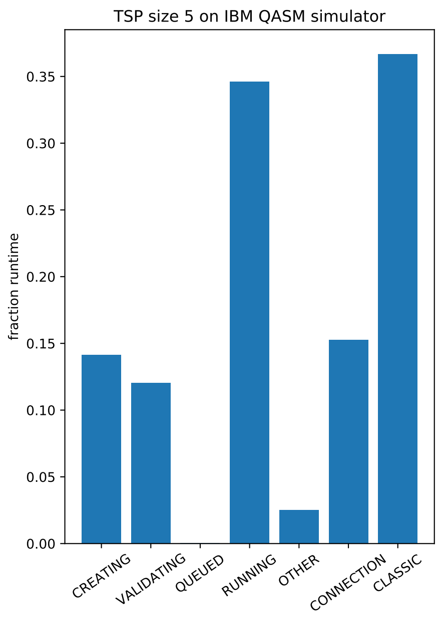
(b) Results of the DSP benchmark 8 to 18 nodes on the IBM Montreal.

Figure 5.6: Results of the DSP benchmark on the IBM Montreal

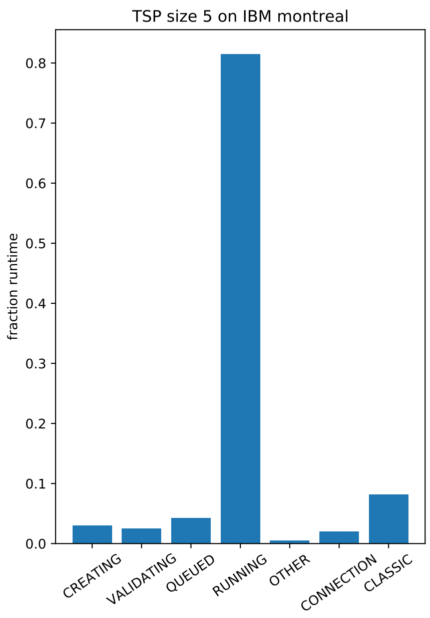
TSP

As the traveling salesman problem (TSP) benchmark requires significantly more qubits, it could only be run on the IBM QASM simulator and the Montreal hardware for 5 nodes (25 qubits). The results are presented as fractions of the total runtime and are shown in Figure 5.7a and 5.7b. This was done as only the results from graph size 5 were possible to measure. Therefore, the results cannot be presented as a line graph. The total runtime for the IBM QASM simulator was 21314.93s, while the total time spend on the IBM Montreal was 971.87s. Comparing these results, we see that on the hardware the main fraction of the computation (about 80% on the Montreal hardware) is spent on the quantum hardware. It can also be seen that this fraction is significantly more than the MCP (which has a running time percentage of 67.39% on the IBM Montreal) or DSP (which has a running time percentage of about 58.98% on the IBM Montreal) benchmarks, as the TSP QAOA circuit is not only much larger in terms of qubits, but also in terms of operations.

A notable difference between the run-time of the IBM QASM simulator and the IBM Montreal, is that significantly more time is spent on classical computations. This could indicate that the IBM QASM simulator takes proportionally less time to simulate the quantum circuit run compared to an actual hardware run.



(a) Results of the TSP benchmark for 5 nodes on the IBM QASM simulator.



(b) Results of the TSP benchmark for 5 nodes on the IBM Montreal.

Figure 5.7: Results of the TSP benchmark on the IBM QASM simulator

5.2. Outcome accuracy (score)

This section presents the second part of our benchmark measurement called outcome accuracy (or score) achieved by the benchmark. The optimal scores are obtained by a brute force search. All measured solutions are obtained with parameter $p = 1$.

MCP

This section presents the scores and corresponding outcome accuracy for the MCP benchmark. The score here refers to the cost of the MCP cost function i.e. the number of cut edges in a graph. This score is therefore calculated for edges e_{jk} and $-1, 1$ representation of the measured state $\mathbf{z}$:

$$c = \sum_{e} 0.5 * (1 - z_j * z_k) \tag{5.1}$$

As the problem size increases, so do the number of edges ($nr\_edges = 2 * nr\_nodes$ for the selected 4-regular circular graph). This means that in the MCP benchmark the nr of cuts should rise as well. The plots of the optimal score show that this increase is almost linear. The scores presented are both the optimal scores for a given graph size (brute force search), as well as the measured score through the QAOA implementation. In Figure 5.8, the measured MCP scores are given for the IBM QASM simulator for graph sizes 5 to 15. The score measured increases almost linearly with the graph size, as expected. The MCP scores for the IBM Montreal for graph sizes 5 to 25 are also presented in Figure 5.8. Similarly to the IBM QASM simulator results, these measured scores increase almost linearly as well. In Figure 5.8, the scores measured for the IBM QASM simulator and IBM Montreal are converted to the outcome accuracy.

In Figure 5.10, the MCP score results for the IBM Nairobi are presented for graph sizes 5 to 7. The scores for the IBM Nairobi, using the IBM runtime environment are also presented in Figure 5.10 for comparison.

The MCP scores presented in Figures 5.8 and 5.10 all show very similar results. The measured score is predictably lower than the optimal score, but all systems show similar results. The relatively low scores can be explained by the limit of $p = 1$ and the limited amount of function evaluations on the classical optimizer. Increasing either will significantly increase runtime, but will show higher scores. This is further discussed in Section 5.3.

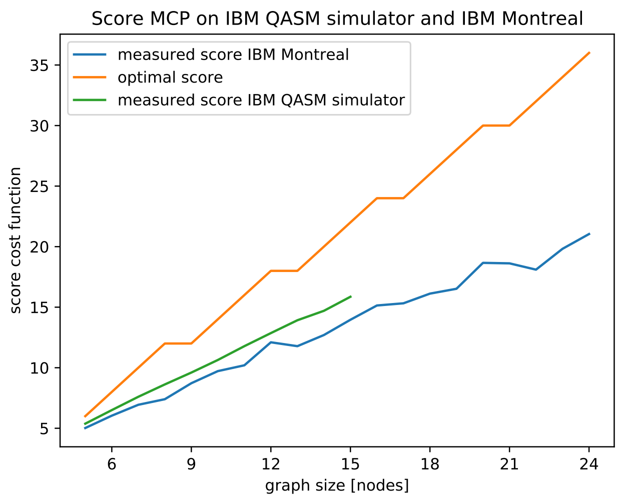


Figure 5.8: Comparison of the score results for the MCP benchmark on the IBM QASM simulator and the IBM Montreal

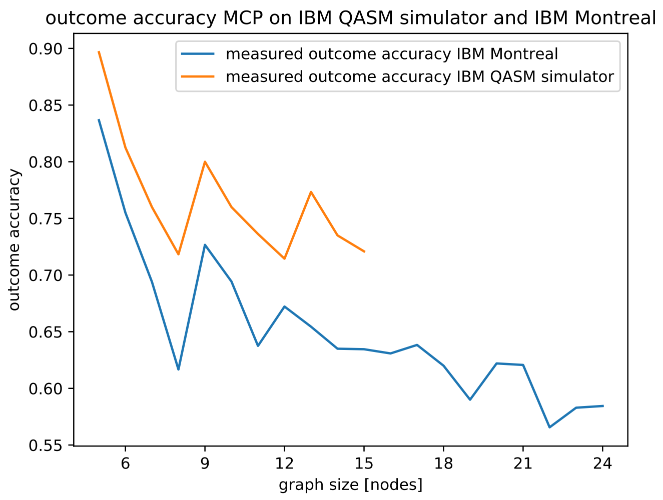


Figure 5.9: Comparison of the outcome accuracy results for the MCP benchmark on the QASM simulator and the IBM Montreal

In the results of the outcome accuracy of both the IBM QASM simulator and the IBM Montreal (Figure 5.9), an exponential decay of the outcome accuracy can be seen. Unsurprisingly, the outcome accuracy for the IBM QASM simulator lies higher than the outcome accuracy of the Montreal hardware. This is mainly due to noise, present in real quantum hardwares.

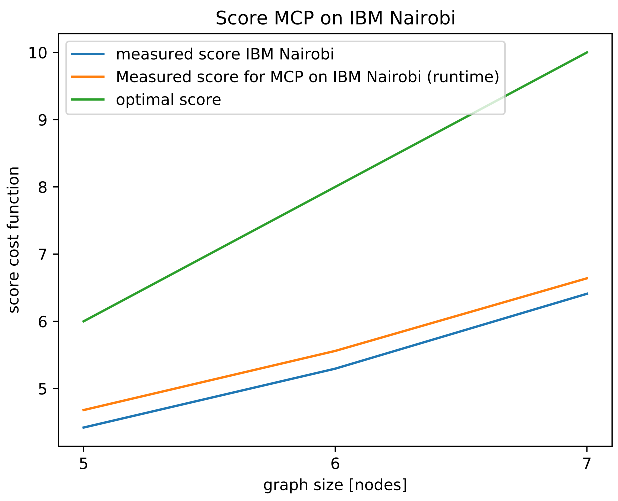


Figure 5.10: Comparison of the score results with and without the runtime environment on the IBM Nairobi

Despite the fact that the IBM Nairobi only supports up to 7 qubits, meaningful results are presented for the MCP benchmark for graph sizes of up to 7 nodes. While the scores for the regular run and the runtime accelerated implementation are very similar, the faster execution time on the runtime environment will enable larger p configurations and therefore better scores in comparable execution time.

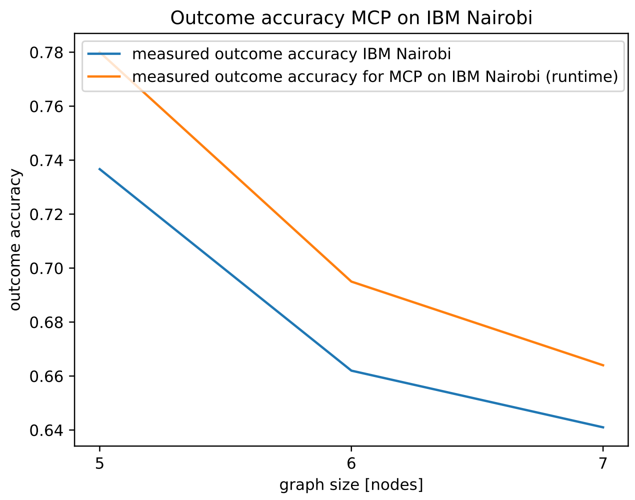


Figure 5.11: Comparison of the outcome accuracy results for the MCP benchmark on the IBM QASM simulator and the IBM Montreal.

The outcome accuracy results for the MCP benchmark on the IBM Nairobi without the runtime environment is presented in Figure 5.11. The results using the runtime environment on the same hardware are also presented in Figure 5.11. This figure again shows the decay of the outcome accuracy for larger graph sizes. By comparing the two Nairobi results, it can be seen that the runtime environment implementation slightly outperforms the regular implementation.

DSP

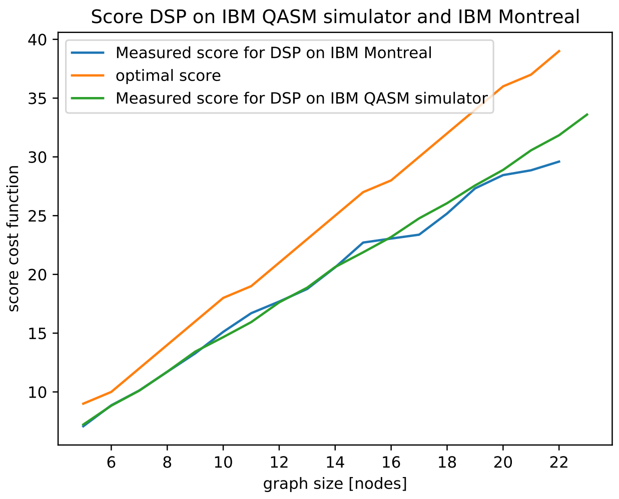


Figure 5.12: Comparison of the score results for DSP benchmark on the IBM QASM simulator and IBM Montreal

The results of the measured cost function score for the DSP benchmark on the IBM QASM simulator are shown in Figure 5.12. The scores are calculated according to the cost evaluation for DSP presented in Section 4.3.2. The scores are presented for graph sizes 5 to 23. The measured scores for the DSP benchmark on the IBM Montreal are also shown in Figure 5.12. The measured scores for the Montreal hardware are shown for graph sizes 5 to 22. Both measured scores are compared to the optimal score. The score needs to be maximized, so the measured score is predictably lower. Interestingly, while the long runtimes in the previous section make it seem like DSP is a worse fit for QAOA, the algorithm appears to perform better in terms of score.

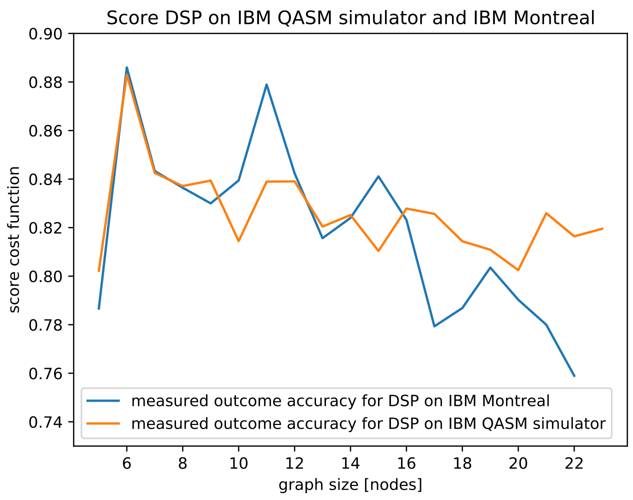


Figure 5.13: Comparison of the outcome accuracy results for DSP benchmark on the IBM QASM simulator and IBM Montreal

The scores of Figure 5.12, converted to the outcome accuracy, are presented in Figure 5.13 for the IBM QASM simulator and the IBM Montreal. The outcome accuracy of the DSP benchmark results appear to be in a very similar order compared to the MCP benchmark results. The DSP benchmark results however, decay much less and prove to be more accurate for larger graph sizes compared to the MCP benchmark. The IBM QASM simulator retains its accuracy for graph size 23 to around 82% for the DSP benchmark, while it drops to 72.5 % for the MCP benchmark. Similarly, on the IBM Montreal, the accuracy for the DSP benchmark is measured at around 76%, while the MCP benchmark results drop to around 58%. It could be suggested to convert large MCP problems to DSP problems to maintain accuracy, but a large drawback is that the number of qubits and runtime significantly increase by doing so. For graphs with low connectivity per node, the DSP implementation might still be favored, as the increase of qubits is determined by the most connections to one node.

TSP

The TSP score results are again limited, due to the fact that the only graph size currently implementable is 5 (25 qubits). The score found with the IBM QASM simulator for $p = 1$ was averaged on 269.7, compared to an optimal score of 200. The score was found as follows: the fully connected 5-node graph consists of edges length 50. Any incorrect implementations of edges (i.e. edge ij is connected but ji is not) are punished by increasing the cost by 5 (and -5 if properly connected). Any edges that should not exist (ii, jj), are given length 100. Therefore the optimal score for size 5 becomes $5 * 50 - 10 * 5 = 200$. Comparing the score of 269.7 results in an error of 34.85%. This error is far from optimal, but the author of this algorithm claims that large p is required for this algorithm to get acceptable results [25]. Deciding on a proper value for parameter p has no conclusive strategy, except that it is expected to grow linearly with the problem size. As such, the optimal value for p is not further explored. As increasing p increases the demand on the quantum hardware to support larger quantum circuit depth, it will be the challenge to the tester to increase p as far as possible, up to the maximum supported depth of the hardware. This of course, should still be in proportion with the problem size.

5.3. Scalability

One of the main metrics discussed in Chapter 4 is scalability. This includes how the runtime and the outcome accuracy scale for larger graph sizes. For the current quantum hardware, the challenge lies mainly in providing enough qubits. Especially in the context of QAOA, a system is desired to provide qubits as close as possible to the quantum supremacy threshold. This threshold was previously estimated to be in the few hundreds of qubits [53]. The current largest system the QPack benchmark was tested on, is the IBM Montreal with 27 qubits. This is still far from the desired threshold.

As quantum hardware progressively becomes larger in terms of qubits and circuit depth, other aspects become more interesting. The scalability of the runtime will become one of the major metrics, but with the flexibility of QAOA, interesting trade offs arise. By increasing the number of QAOA layers p, the runtime becomes larger, but the outcome accuracy theoretically increases. The challenge then lies to optimize the parameter to achieve the best results balanced for both metrics. Possibly, a focus can be placed on either of the metrics, in order to develop hardware specifically for fast runtimes or high accuracy. An interesting note on finding the optimal parameter p for QAOA, is that increasing p and therefore the quantum circuit depth, will also increase noise. This in turn will lower the outcome accuracy. This could mean that a different optimal p could be found for each quantum hardware instance, which optimizes the outcome accuracy.

In the results shown in the previous section, a decrease of outcome accuracy is observed as the graph size increases. This is to be expected as the problems gets increasingly difficult for the algorithm to optimize. In order to adjust for this decrease in outcome accuracy, the general practice is to increase p. As previously discussed, this parameter is not optimized within the benchmark for multiple reasons, including the possible hardware dependency of this parameter.

5.4. XACC implementation

This section covers the implementation of QPack using the XACC library [80]. The XACC library allows for a single piece of code to run on different backends, as opposed to writing a new code for every backend provider. The aim of using this library is to create a universal benchmark: the user should not need to rewrite the benchmark in order for the benchmark to run on their system. This universal implementation shows how the QPack benchmark can be used to evaluate different quantum backends. The XACC implementation can be found on the **github repo**. Credits go to Huub Donkers for rewriting the QPack benchmark to the XACC implementation. Using the XACC library, the QPack benchmark was tested on 5 different simulators:

- IBM QASM simulator (remote) [39]
- IonQ (remote) [61]
- aer (local) [39]
- qsim (local) [111]
- qpp (local) [48]

The IBM QASM simulator is the same remote simulator as presented in previous sections. This simulator runs remotely through the IBM quantum experience. The second backend is the IonQ simulator. This simulator is used remotely as well. The aer simulator is part of Qiskit and is run locally. The qsim simulator [111] is part of Google's quantum AI project and run locally as well. The qpp simulator [48] is a C++ quantum computing library that is able to simulate 25 pure state qubits or 12 mixed state qubits. This simulator is again run locally. The local simulators aer, qpp and qsim are all integrated in the XACC library.

MCP

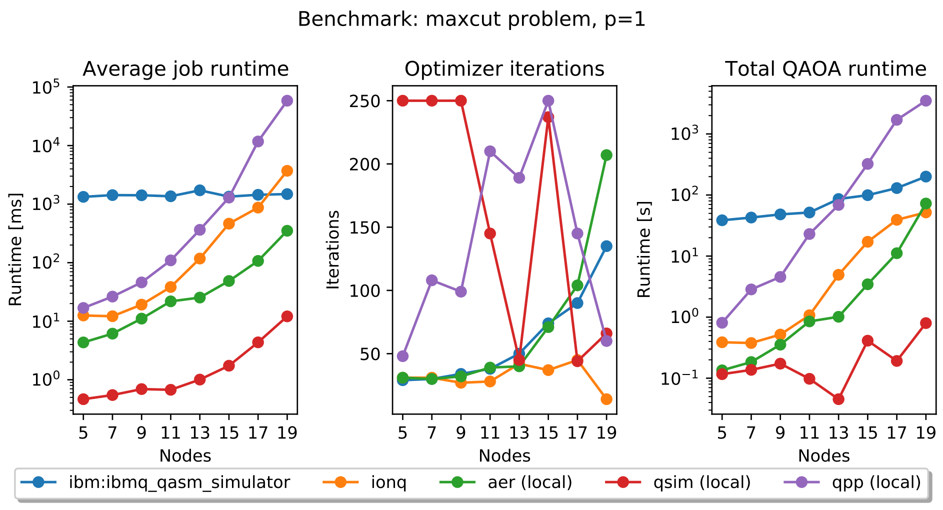


Figure 5.14: MCP benchmark runtime results for the QASM, IonQ, aer, qsim and qpp simulators for 5 to 19 nodes.

In Figure 5.14, the runtime results for all mentioned simulators are presented. The total runtime of the QAOA optimization is shown, as well as the average simulated quantum runtime for each iteration, as well as the number of iterations. All runtime results are presented on a logarithmic scale. From this figure can be seen that for most simulators the progression of both the average runtime and the number of iterations scale quadratic. One of the exceptions is that the average runtime of the IBM QASM simulator is much more constant (note that the logarithmic scale must be taken into account). The offset is however much larger. This means that for most of the 'smaller' graph sizes, the IBM QASM simulator is slower. This is only caused by the offset in simulated quantum run time, as the number of iterations is in the same order of size and has similar exponential progression as most of the other simulators. The logarithmic progression of the total simulated quantum runtime however suggests that the IBM QASM simulator should run faster for graph sizes much larger. The total runtime of the IBM QASM simulator already crosses the fastest increasing runtimes of the qpp simulator at graph size 13. The qsim simulator is however notably faster. The logarithmic progression of the IBM QASM simulator could be explained by the remote execution of the simulator, very likely providing much more computing resources. Interestingly, while the IonQ simulator is also run remotely, this simulator does not show the same linearity as the IBM QASM simulator, but increases exponentially, much like the local simulators. The offset of the simulated quantum runtime is still intriguing. One possible explanation could be that the IBM QASM simulator always simulates the maximum amount of qubits, also explaining the linearity.

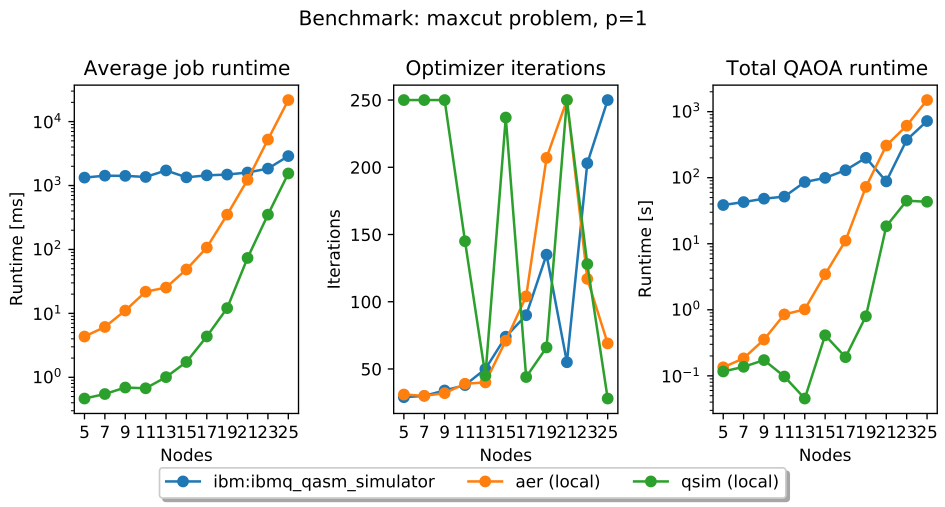


Figure 5.15: MCP benchmark runtime results for the QASM, aer, qsim simulators for 5 to 25 nodes.

Figure 5.15 shows the same results as Figure 5.14, extended to 25 qubits. In this figure, the results for the IonQ and qpp simulator are not included, as these show difficulties simulating over 18 qubits. In this figure the simulated runtime of qsim can be seen nearing the runtime of the IBM QASM simulator near 25 qubits. Due to a drop in iterations for graph sizes 23 and 25, the total runtime of qsim is still much lower for these sizes. At approximately graph size 18, the IBM QASM simulator performs faster than the local aer simulator.

In both Figures 5.14 and 5.15, the iterations of qpp and qsim show random progression. The number of iterations of the classical simulator remains unpredictable, as the algorithm might sometimes show much more difficulty finding the optimal points, compared to other similar runs, and vice versa. The number of iterations do not exceed 250, as a limit has been imposed to reduce outliers.

DSP

The results of the IBM QASM, IonQ, aer, qsim and qpp simulators for the DSP benchmark are shown in Figure 5.16. These show the average simulated quantum runtime per iteration, the number of classical optimizer iterations and the total optimization time. Similarly to the MCP results, the remote IBM QASM simulator shows a relatively logarithmic progression. In the DSP results, the qsim proves to be the fastest simulator for the tested sizes.

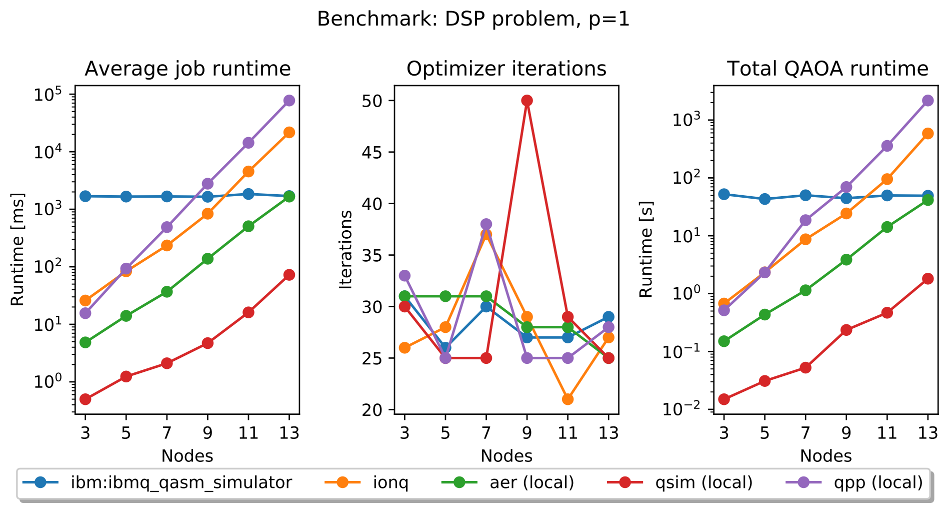


Figure 5.16: DSP results for the QASM, IonQ, aer, qsim and qpp simulators for 3 to 13 nodes.

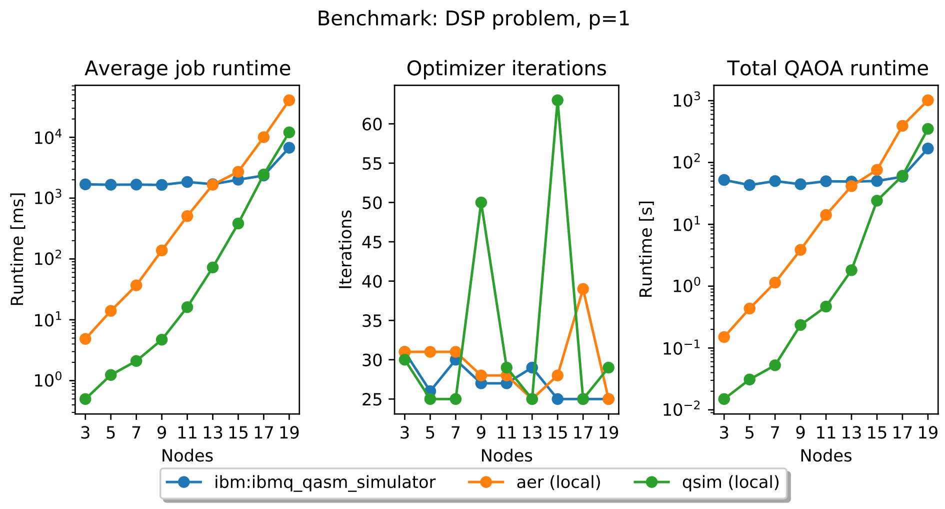


Figure 5.17: DSP results for the QASM, aer and qsim simulators for 3 to 19 nodes.

For the IBM QASM, aer and qsim simulators, sizes up to 19 nodes were possible, which are shown in Figure 5.17. These larger sizes were not possible for the IonQ and qpp simulators, as these have shown difficulties for such large qubit requirements. With the implemented benchmark graphs, the quantum circuit would require $n+5$ qubits for n nodes. The results interestingly show a much larger increase in runtime for the qsim simulator compared to the aer and IBM QASM simulator. For graph size 19, both the average quantum job runtime and the total runtime exceed the IBM QASM simulator runtimes. As this was something not seen in the MCP benchmark, this observation could mean that the qsim has much more difficulty simulating large circuit depths. This would be derived from the fact that the DSP benchmark has significantly larger ciruit depth compared to the MCP benchmark for comparable numbers of qubits. Another reason could be that incidentally, the qsim simulator found more difficulty finding an optimal value for larger number of nodes, as the number of iterations at graph sizes 9 and 15 for the qsim are notably higher. The average runtime however, shows that even disregarding the number of iterations, the runtime is already increasing much faster compared to other simulators.

TSP

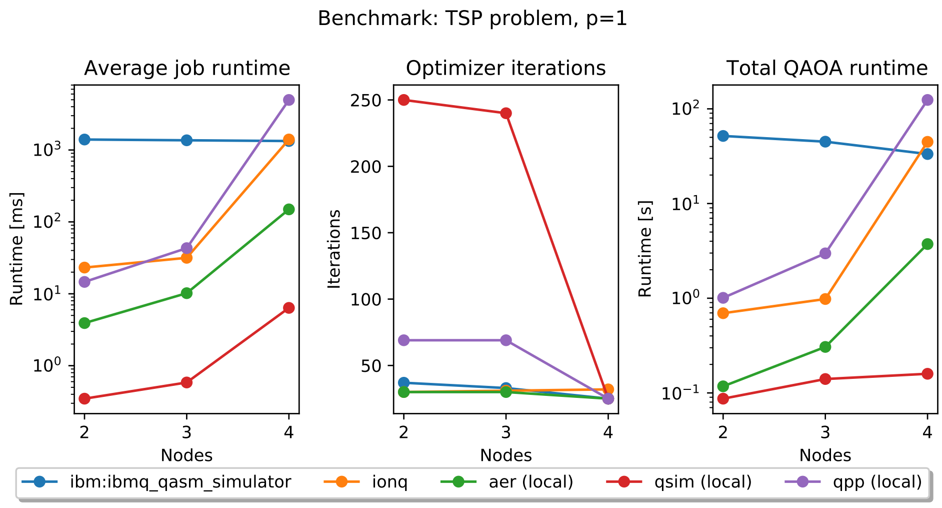


Figure 5.18: TSP results for the QASM, IonQ, aer, qsim and qpp simulators for 2 to 4 nodes.

The results of the TSP benchmark for the IBM QASM, IonQ, aer, qsim and qpp simulators for 2 to 4 nodes are shown in Figure 5.18. While these graph sizes appear to be trivial, the quantum requirement with the QAOA implementation are substantial. Interestingly, in these results, the runtime of the IonQ at 4 nodes (16 qubits) already exceeds the runtime of the IBM QASM simulator. This was not seen for the MCP benchmark. Most likely, the main contributing factor to this observation is the circuit length, similarly to earlier observations for the DSP benchmark.

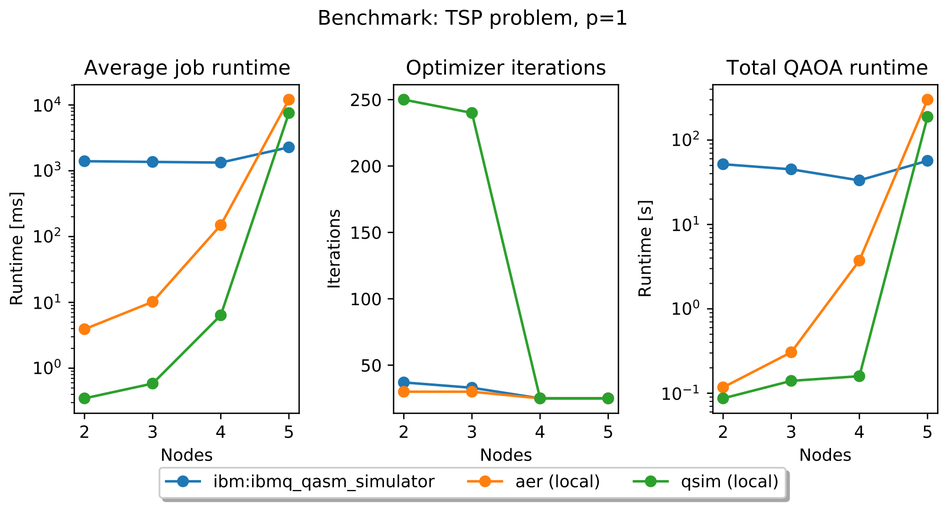


Figure 5.19: DSP results for the QASM, aer and qsim simulators for 2 to 5 nodes.

The extended results for graph size 5 (25 qubits) are shown in Figure 5.19. The IonQ and qpp results are not included for the same reasons in the extentions of the MCP and DSP benchmark. In these results, the IBM QASM simulator performs better at 5 nodes than both the aer and qsim simulator. The qsim simulator perform better than the ear simulator for the tested graph sizes, but its runtime rises quicker. From the results of all previous benchmarks, the remote IBM QASM simulator performs better for qubits sizes, starting around 13 qubits. For smaller qubits the qsim benchmark performs better, but its runtime rises much quicker compared to other simulators. This can be observed more significantly when the benchmark involves larger circuit depths.

5.5. Discussion

In the previous section, multiple instances of the benchmark have been run on different systems. From these measurements, certain observations can be clearly made. Firstly, the expected increase of the runtime with larger quantum circuits have been observed in both simulators, as well as on quantum hardware. This increase on quantum hardware is also the main significant contributor to the total runtime, suggesting that direct improvements in the field of quantum accelerated optimization should be in reducing the quantum circuit or reducing the calls to the quantum hardware from the classical optimizer.

Reducing the quantum circuit could be done through the compiler, creating more efficient compilations of the quantum circuit and reducing the circuit depth [4]. Another way of reducing the quantum resources is to move the quantum effort to the classical computation. While this is quite the opposite of what quantum acceleration aims to achieve, but in the context of QAOA with QUBO, this might be an efficient implementation. To the best of the author's knowledge, no publications to date have been presented concerning the performance comparison of QUBO and direct QAOA, a significant increase in runtime for both the DSP and TSP can be seen in direct QAOA implementation. QAOA using QUBO aims to standardize the quantum algorithm, likely retaining the complexity of the quantum circuit.

Other observations from the benchmark are the improvements that can be achieved through the quantum stack infrastructure. IBM has already published a detailed paper on their performance increase using the runtime environment [45] and with the implementation of the QPack benchmark the infrastructure improvements are validated. The queue peaks observed are of significant contribution to the measured total runtime. Using the runtime environment, such peaks contributions are much less prominent and the overall runtime improvement suggests that the overall reduction in communication to host is of significance as well.

In the case of the TSP benchmark, a percentage comparison has been made on the time spent on the different steps. The main take from these results, is that the quantum hardware takes significantly more time to process the circuit compared to the simulated quantum circuits. This indicates, that on small problem sizes, current hardware cannot outperform simulated circuits of the same qubits and depth. From these results, the second bottleneck in quantum acceleration can be seen: the classical computation. With hybrid optimization, the optimization still spends a significant portion on the classical (optimization) computations. It is therefore important to improve the required classical computations, possibly with dedicated hardware.

6 Conclusion and future developments

In this thesis, the theoretical background on QAOA has been explored to set a basic understanding of how the algorithm operates and what the trade-offs are. Increasing accuracy by increasing p requires longer run-time and is required for larger problems. The performance is also determined by the clause density and makes the performance problem dependent. State of the art development of the QAOA algorithm in practice is shown to be advancing, with a wide variety of algorithms shown for different, but similar problems. While these NP-hard problems are similar, it has been shown that QAOA is not well suited for every problem [106]. In some cases, such as the Ising problem, current classical approximations are expected to perform better and alternatives like the quantum adiabatic algorithm might be a better fit. For most of the problems discussed, the QAOA algorithm appears to be a good solution, with applications even reaching a full-stack application. The development of each implementation has been evaluated according to a TLR fitted for quantum algorithms. In this evaluation it is shown that most applications have at least been developed to the point that the algorithm has been simulated on a quantum simulator to show the capability of the implementations. In the case of the packing, MCP, DSP and TSP applications, a proper resource and performance analysis has been done. The implementation of TSP by Sarkar et al. [105] is currently the only implementation to have reached the level of a full stack development. This implementation also shows that current NISQ technology cannot properly support QAOA algorithms for practical implementation. Either advances on quantum hardware must be made, or more resilient implementations of QAOA must be developed.

Using this assessment, the MCP, DSP and TSP applications have been selected to be included in the QPack benchmark. Furthermore, a theoretical and numerical analysis has been presented on classical optimizers to find the implementation to best suit the QAOA algorithm. The results show that a clear trade-off exists between run-time and accuracy.

A reference implementation for QPack is presented with critical quantum benchmark requirements in mind [21, 97]. This benchmark will explore run-time and accuracy for available NISQ era quantum computers using various applications. The benchmark reflects the resources available, the implemented optimizations and the bottlenecks in the architectural implementation. The latter can specify the run-time on the quantum hardware, the classical hardware and the communication between both. The current implementation of the benchmark is available on the **github repo**.

Results of the benchmark have been presented for several IBM systems and give insight to current bottlenecks and hardware improvements. A clear advantage was seen in the usage of the IBM runtime environment, using a remote classical computer. A significant speedup of 24× was observed, but expected to increase for larger problem instances.

In future work, the benchmark can be further tailored for an improved implementation. The main pressing feature is to create a universal benchmark, using a quantum library. Libraries that currently explore this field are the LibKet [83] library and the XACC library [80].

For both libraries, the implementation in C++ will allow for better performance measurements for both run-time and memory consumption. The benchmark set presented by Qiskit uses the airspeed velocity (ASV) [40] tool to measure performance. This could be implemented to the current python implementation, but this effort is likely better spent on implementing such measurements on a C++ implementation. In the case of XACC, both could be done and a comparison between the python and C++ implementation could be presented.

An implementation using the XACC library is presented, showing progression towards a universal benchmark. With this implementation, a comparison is made between various simulators. The comparison highlights the advantages and disadvantages of the various simulators. While for example the QASM simulator performs much slower on smaller graph sizes, it outperforms other simulators at larger graph sizes (≥ 13). The qsim on the other hand excels at smaller graph sizes, but a much faster increase in runtime can be observed compared to other simulators. This shows that the benchmark is a viable tool for users to determine which simulator or hardware could be better tailored for their application.

Fine-tuning of the applications is also required, as a trade-off can be made between accuracy in run-time and precision when choosing the classical (hybrid) optimizer and the QAOA depth p. Finding the minimal requirements for the QAOA application to compete with classical approaches will need further examination. Currently, these parameters are left as user inputs.

A significant challenge in optimizing QAOA is parameter training. While studies have been performed using machine learning [8, 68, 114], other approaches have been presented as well. Several publications discuss the concentration of the β and γ parameters [6, 46, 110], which can be used to predict these parameters to speed up the parameter training. Unfortunately, for this benchmark implementation no similar results were observed. However, as multiple unaffiliated researchers have observed similar results, implementing this will certainly become a future improvement to the benchmark.

Furthermore, the QAOA applications currently implemented are not optimal in terms of resources and can be further improved upon as mentioned in Section 4.3.2. Finally, support for parallel processing is desired. As classical optimizers such as Nelder-Mead Simplex can be run in parallel [34, 72, 87], more objective functions need to run in parallel. If the quantum hardware can support a multiple of the qubits required by the objective function, such a parallel implementation can be supported. With current quantum hardware sizes, such implementations are not yet of importance, as practical problem sizes do not fit on the current hardware [53]. As the quantum hardware scales further, this parallelism can result in significant speedup.

The QPack benchmark aims to expand its repertoire of algorithms in the future, as applications based on Shor's and Grover's algorithm [104] will become implementable on quantum computers. Currently, the quantum hardware cannot support these algorithms as more qubits and larger circuit depths are required. As more applications become practical on quantum computers, the QPack benchmark is envisioned to grow correspondingly to set the standard as a practical quantum algorithm benchmark. Interesting near term benchmark expansions include $|0\rangle$ bias measurements, the BB84 protocol and VQE for Ising problems.

A Additional quantum gates

This appendix contains an overview of omitted gates, used in the implementation of the QAOA circuits for the dominating set and traveling salesman.

```
# controlled rotation Y
def cry(theta):
    qc = QuantumCircuit(2)
    qc.ry((pi/2)-theta/2, 0)
    qc.cnot(1, 0)
    qc.ry(-((pi/2)-theta/2), 0)
    return qc.to_gate()
```

```
# 2-qubit controlled rotation Y
def ccry(theta):
    qc = QuantumCircuit(3)
    qc.toffoli(0, 1, 2)
    qc.ry(-theta/2, 2)
    qc.toffoli(0, 1, 2)
    qc.ry(theta / 2, 2)
    return qc.to_gate()
```

```
# Split & Cyclic Shift unitary (SCS) gate
def scs(n, k):
    qc = QuantumCircuit(n)
    qc.cnot(n - 2, n-1)
    theta = 2 * (acos(sqrt(1 / n)))
    qc.append(cry(theta), [n-2, n - 1])
    qc.cnot(n - 2, n-1)

    for m in range(k - 1):
        l = 2+m
        control = n-2-m
        qc.cnot(control-1, n-1)
        theta = 2 * (acos(sqrt((n-control) / n)))
        qc.append(ccry(theta), [n-1, control, control-1])

        qc.cnot(control-1, n-1)
    return qc.to_gate()
```

```
#deterministic  Dicke state preparation (Bartschi & Eidenbenz, 2019)
def dicke_init(n, k):
    #unoptimized version
    qc = QuantumCircuit(n)
    qc.x(range(n-k, n))
    for i in range(n, k, -1):
        qc.append(scs(i, k), range(0, i))

    for i in range(k, 1, -1):
        qc.append(scs(i, i-1), range(0, i))
    return qc.to_gate()
```

```
# 2-qubit OR gate
def OR_2q():
    qc = QuantumCircuit(3)
    qc.toffoli(0, 1, 2)
    qc.cnot(0, 1)
    qc.cnot(0, 2)
    return qc.to_gate()
```

```
# n-qubit controlled rotation Z gate
def OR_nrz(n, gamma):
    ORGate = OR_2q()
    qc = QuantumCircuit(2*n)
    qc.append(ORGate, [0, 1, n])
    for i in range(2, n):
        qc.append(ORGate, [i, n+i-2, n+i-1])
    qc.crz(gamma, 2*n-2, 2*n-1)
    for i in range(n, 2, -1):
        qc.append(ORGate, [n-i-1, 2*n-2-i, 2*n-1-i])
    qc.append(ORGate, [0, 1, n])
    return qc.to_gate()
```

Bibliography

[1] Nlopt. https://readthedocs.org/projects/nlopt/. Accessed: 2021-02-01.

[2] Ibm aer quasm simulator. https://qiskit.org/documentation/stubs/qiskit.providers.aer.QasmSimulator.html.

[3] Zeineb Abdmouleh, Adel Gastli, L. Ben-Brahim, Mohamed Haouari, and Nasser Al-Emadi. Review of optimization techniques applied for the integration of distributed generation from renewable energy sources. *Renewable Energy*, 113, 05 2017. doi: 10.1016/j.renene.2017.05.087.

[4] Smaran Adarsh. Resource optimal executable quantum circuit generation using approximate computing. 2021.

[5] V. Akshay, H. Philathong, M.E.S. Morales, and J.D. Biamonte. Reachability deficits in quantum approximate optimization. *Physical Review Letters*, 124(9), Mar 2020. ISSN 1079-7114. doi: 10.1103/physrevlett.124.090504. URL http://dx.doi.org/10.1103/PhysRevLett.124.090504.

[6] V. Akshay, D. Rabinovich, E. Campos, and J. Biamonte. Parameter concentrations in quantum approximate optimization. *Phys. Rev. A*, 104:L010401, Jul 2021. doi: 10.1103/PhysRevA.104.L010401. URL https://link.aps.org/doi/10.1103/PhysRevA.104.L010401.

[7] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh. Analysis of quantum approximate optimization algorithm under realistic noise in superconducting qubits, 2019.

[8] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh. Accelerating quantum approximate optimization algorithm using machine learning, 2020.

[9] Juan Alonso and Charbel Farhat. Gradient-based optimization, 2012. URL http://adl.stanford.edu/aa222/Lecture_Notes_files/chapter3_gradient.pdf.

[10] Juan Alonso and Charbel Farhat. Gradient-free optimization, 2012. URL http://adl.stanford.edu/aa222/Lecture_Notes_files/chapter6_gradfree.pdf.

[11] M.H. Alsuwaiyel. *Algorithms: Design Techniques and Analysis*. Lecture notes series on computing. World Scientific, 1999. ISBN 9789812386397. URL https://books.google.nl/books?id=TbhgDQAAQBAJ.

[12] ATOS. Atos announces q-score, the only universal metrics to assess quantum performance and superiority. *ATOS*, Dec 2020. URL https://atos.net/en/2020/press-release_2020_12_04/atos-announces-q-score-the-only-universal-metrics-to-assess-quantum-performance-and-superiority.

[13] Sammie Bae. *Big-O Notation*, pages 1–11. Apress, Berkeley, CA, 2019. ISBN 978-1-4842-3988-9. doi: 10.1007/978-1-4842-3988-9_1. URL https://doi.org/10.1007/978-1-4842-3988-9_1.

[14] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36:493–513, 05 1988. doi: 10.1287/opre.36.3.493.

[15] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/170992.

[16] Thomas Bergamaschi. Quantum approximate optimization algorithms on the “traveling salesman problem”, 2020. URL https://medium.com/mit-6-s089-intro-to-quantum-computing/quantum-approximate-optimization-algorithms-on-the-traveling-salesman-problem-703b8aee6624.

[17] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5): 1411–1473, 1997.

[18] Lev S Bishop, Sergey Bravyi, Andrew Cross, Jay M Gambetta, and John Smolin. Quantum volume. *Quantum Volume. Technical Report*, 2017.

[19] Lennart Bittel and Martin Kliesch. Training variational quantum algorithms is np-hard. *Physical Review Letters*, 127(12):120502, 2021.

[20] Robin Blume-Kohout and Kevin C Young. A volumetric framework for quantum computer benchmarks. *Quantum*, 4:362, 2020.

[21] Robin J Blume-Kohout and Kevin Young. Metrics and benchmarks for quantum processors: State of play. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia, 2019.

[22] J. Brimberg. The fermat-weber location problem revisited. *Math. Program.*, 71(1):71–76, November 1995. ISSN 0025-5610. doi: 10.1007/BF01592245. URL `https://doi.org/10.1007/BF01592245`.

[23] Andreas Bärtschi and Stephan Eidenbenz. Deterministic preparation of dicke states. *Lecture Notes in Computer Science*, page 126–139, 2019. ISSN 1611-3349. doi: 10.1007/978-3-030-25027-0_9. URL `http://dx.doi.org/10.1007/978-3-030-25027-0_9`.

[24] J. Carlson, J.A.C.A.J.A. Wiles, J.A. Carlson, A. Jaffe, A. Wiles, Clay Mathematics Institute, and American Mathematical Society. *The Millennium Prize Problems*. Amsns AMS non-series Title Series. American Mathematical Society, 2006. ISBN 9780821836798. URL `https://books.google.nl/books?id=7wJIPJ80RdUC`.

[25] Jack Ceroni. Fun graphs with qaoa, 2016. URL `https://lucaman99.github.io/new_blog/2020/mar16.html`.

[26] Jaeho Choi, Seunghyeok Oh, Soohyun Park, Jong-Kook Kim, and Joongheon Kim. Proper cost hamiltonian design for combinatorial optimization problems: A boolean function approach. In *2021 International Conference on Information Networking (ICOIN)*, pages 469–472, 2021. doi: 10.1109/ICOIN50884.2021.9333931.

[27] Mark W. Coffey. Adiabatic quantum computing solution of the knapsack problem, 2017.

[28] Stephen A. Cook. The complexity of theorem-proving procedures. STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. ISBN 9781450374644. doi: 10.1145/800157.805047. URL `https://doi.org/10.1145/800157.805047`.

[29] Gavin E. Crooks. Performance of the quantum approximate optimization algorithm on the maximum cut problem, 2018.

[30] Elizabeth Crosson, Edward Farhi, Cedric Yen-Yu Lin, Han-Hsuan Lin, and Peter Shor. Different strategies for optimization using the quantum adiabatic algorithm, 2014.

[31] Sofia Josefina Lago Dudas Daniel Henry. Solving the traveling salesman problem using qaoa, June 2019. URL `https://cs269q.stanford.edu/projects2019`.

[32] Christoph Dankert, Richard Cleve, Joseph Emerson, and Etera Livine. Exact and approximate unitary 2-designs and their application to fidelity estimation. *Physical Review A*, 80(1):012304, 2009.

[33] Pierre Dupuy de la Grand'rive and Jean-Francois Hullo. Knapsack problem variants of qaoa for battery revenue optimisation, 2019.

[34] John E Dennis Jr and Virginia Torczon. Parallel implementations of the nelder-mead simplex algorithm for unconstrained optimization. In *High Speed Computing*, volume 880, pages 187–191. International Society for Optics and Photonics, 1988.

[35] R. H. Dicke. Coherence in spontaneous radiation processes. *Phys. Rev.*, 93:99–110, Jan 1954. doi: 10.1103/PhysRev.93.99. URL `https://link.aps.org/doi/10.1103/PhysRev.93.99`.

[36] Joseph Emerson, Robert Alicki, and Karol Życzkowski. Scalable noise estimation with random unitary operators. *Journal of Optics B: Quantum and Semiclassical Optics*, 7(10):S347, 2005.

[37] Stefan C Endres, Carl Sandrock, and Walter W Focke. A simplicial homology algorithm for lipschitz optimisation. *Journal of Global Optimization*, 72(2):181–217, 2018.

[38] Alexander Erhard, Joel J Wallman, Lukas Postler, Michael Meth, Roman Stricker, Esteban A Martinez, Philipp Schindler, Thomas Monz, Joseph Emerson, and Rainer Blatt. Characterizing large-scale quantum computers via cycle benchmarking. *Nature communications*, 10(1):1–7, 2019.

[39] MD SAJID ANIS et al. Qiskit: An open-source framework for quantum computing, 2021.

[40] Michael Droettboom et al. Airspeed velocity, 2018. URL `https://asv.readthedocs.io/en/stable/index.html`.

[41] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[42] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.

[43] Samuele Ferracin, Theodoros Kapourniotis, and Animesh Datta. Accrediting outputs of noisy intermediate-scale quantum computing devices. *New Journal of Physics*, 21(11):113038, 2019.

[44] Lance Fortnow and John Rogers. Complexity limitations on quantum computation. *Journal of Computer and System Sciences*, 59(2):240–252, 1999.

[45] Julien Gacon, Christa Zoufal, Giuseppe Carleo, and Stefan Woerner. Simultaneous perturbation stochastic approximation of the quantum fisher information, 2021.

[46] Alexey Galda, Xiaoyuan Liu, Danylo Lykov, Yuri Alexeev, and Ilya Safro. Transferability of optimal qaoa parameters between random graphs, 2021.

[47] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[48] Vlad Gheorghiu. Quantum++: A modern c++ quantum computing library. *PloS one*, 13(12):e0208073, 2018.

[49] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models, 2019.

[50] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995. ISSN 0004-5411. doi: 10.1145/227683.227684. URL `https://doi.org/10.1145/227683.227684`.

[51] Oded Goldreich. *In a World of P=BPP*, pages 191–232. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-22670-0. doi: 10.1007/978-3-642-22670-0_20. URL `https://doi.org/10.1007/978-3-642-22670-0_20`.

[52] Nicholas J Guerrero. Solving combinatorial optimization problems using the quantum approximation optimization algorithm. 2020.

[53] G. G. Guerreschi and A. Y. Matsuura. Qaoa for max-cut requires hundreds of qubits for quantum speed-up. *Scientific Reports*, 9(1), May 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-43176-9. URL `http://dx.doi.org/10.1038/s41598-019-43176-9`.

[54] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, Feb 2019. ISSN 1999-4893. doi: 10.3390/a12020034. URL `http://dx.doi.org/10.3390/a12020034`.

[55] Matthew P. Harrigan and et al. Quantum approximate optimization of non-planar graph problems on a planar superconducting processor. *Nature Physics*, Feb 2021. doi: 10.1038/s41567-020-01105-y. URL `https://doi.org/10.1038/s41567-020-01105-y`.

[56] M. Héder. From nasa to eu: the evolution of the trl scale in public sector innovation. *The Innovation Journal*, 22:1, 2017.

[57] Itay Hen and Federico M. Spedalieri. Quantum annealing for constrained optimization. *Physical Review Applied*, 5(3), Mar 2016. ISSN 2331-7019. doi: 10.1103/physrevapplied.5.034007. URL `http://dx.doi.org/10.1103/PhysRevApplied.5.034007`.

[58] Jason Hicken, Juan Alonso, and Charbel Farhat. Gradient-free optimization. URL `http://adl.stanford.edu/aa222/Home.html`.

[59] Jack Hidary. *Quantum Computing: An Applied Approach*. 01 2019. ISBN 978-3-030-23921-3. doi: 10.1007/978-3-030-23922-0.

[60] Russell Impagliazzo and Avi Wigderson. P= bpp if e requires exponential circuits: Derandomizing the xor lemma. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 220–229, 1997.

[61] IonQ. Ionq simulator reference. URL `https://ionq.com/docs/get-started-with-qiskit`.

[62] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM Journal on computing*, 22(5):1087–1116, 1993.

[63] Zhang Jiang, Eleanor G. Rieffel, and Zhihui Wang. Near-optimal quantum circuit for grover's unstructured search using a transverse field. *Physical Review A*, 95(6), Jun 2017. ISSN 2469-9934. doi: 10.1103/physreva.95.062317. URL `http://dx.doi.org/10.1103/PhysRevA.95.062317`.

[64] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of optimization Theory and Applications*, 79(1):157–181, 1993.

[65] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[66] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic methods for global optimization. *American Journal of Mathematical and Management Sciences*, 4(1-2):7–40, 1984. doi: 10.1080/01966324.1984.10737135. URL `https://doi.org/10.1080/01966324.1984.10737135`.

[67] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. ISBN 0-306-30707-3. URL `http://dblp.uni-trier.de/db/conf/coco/cocc1972.htmlKarp72`.

[68] Sami Khairy, Ruslan Shaydulin, Lukasz Cincio, Yuri Alexeev, and Prasanna Balaprakash. Reinforcement learning for quantum approximate optimization. volume 19, 2019.

[69] J. Kleinberg and É. Tardos. *Algorithm Design*. Alternative Etext Formats. Pearson/Addison-Wesley, 2006. ISBN 9780321295354. URL `https://books.google.nl/books?id=OiGhQgAACAAJ`.

[70] Emanuel Knill, Dietrich Leibfried, Rolf Reichle, Joe Britton, R Brad Blakestad, John D Jost, Chris Langer, Roee Ozeri, Signe Seidelin, and David J Wineland. Randomized benchmarking of quantum gates. *Physical Review A*, 77(1):012307, 2008.

[71] Donald E Knuth. Big omicron and big omega and big theta. *ACM Sigact News*, 8(2):18–24, 1976.

[72] Donghoon Lee and Matthew Wiswall. A parallel implementation of the simplex function minimization routine. *Computational Economics*, 30(2):171–187, 2007.

[73] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation, 2021.

[74] Cedric Yen-Yu Lin and Yechao Zhu. Performance of qaoa on typical instances of constraint satisfaction problems with bounded degree, 2016.

[75] LocalSolver. Localsolver: Max cut, 2020. URL `https://www.localsolver.com/docs/last/exampletour/maxcut.html`.

[76] Thuy Mai. Technology readiness level | nasa, oct 2018. URL `https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_accordion1.html`.

[77] Christophe Dutang Marie Laure Delignette Muller. Which optimization algorithm to choose?, May 2020. URL `https://cran.r-project.org/web/packages/fitdistrplus/vignettes/Optimalgo.html#results-of-the-numerical-investigation-1`.

[78] Jeffrey Marshall, Filip Wudarski, Stuart Hadfield, and Tad Hogg. Characterizing local noise in qaoa circuits. *IOP SciNotes*, 1(2):025208, Aug 2020. ISSN 2633-1357. doi: 10.1088/2633-1357/abb0d7. URL `http://dx.doi.org/10.1088/2633-1357/abb0d7`.

[79] Rajesh Matai, Surya Singh, and Murari Lal Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. In Donald Davendra, editor, *Traveling Salesman Problem*, chapter 1. IntechOpen, Rijeka, 2010. doi: 10.5772/12909. URL `https://doi.org/10.5772/12909`.

[80] Alexander McCaskey, Dmitry Lyakh, Eugene Dumitrescu, Sarah Powers, and Travis Humble. Xacc: a system-level software infrastructure for heterogeneous quantum-classical computing. *Quantum Science and Technology*, 5, 01 2020. doi: 10.1088/2058-9565/ab6bf6.

[81] Kristel Michielsen, Madita Nocon, Dennis Willsch, Fengping Jin, Thomas Lippert, and Hans De Raedt. Benchmarking gate-based quantum computers. *Computer Physics Communications*, 220:44–55, Nov 2017. ISSN 0010-4655. doi: 10.1016/j.cpc.2017.06.011. URL `http://dx.doi.org/10.1016/j.cpc.2017.06.011`.

[82] Yuichiro Minato. Solving the ising many-body problem of protein folding problem using blueqat and qaoa, May 2019. URL `https://medium.com/@minatoyuichiro/solving-the-ising-many-body-problem-of-protein-folding-problem-using-blueqat-and-qaoa-fd3afe571b3`

[83] Matthias Möller and Merel Schalkers. Libket: A cross-platform programming framework for quantum-accelerated scientific computing. In Valeria V. Krzhizhanovskaya, Gábor Závodszky, Michael H. Lees, Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira, editors, *Computational Science – ICCS 2020*, pages 451–464, Cham, 2020. Springer International Publishing. ISBN 978-3-030-50433-5.

[84] Ewan Munro. 3.2 application showcase: Entropica. URL `https://www.youtube.com/watch?v=NZ4OwvNaRI8&t=300s`.

[85] Murphy Yuezhen Niu, Sirui Lu, and Isaac L. Chuang. Optimizing qaoa: Success probability and runtime dependence on circuit depth, 2019.

[86] Tobias J Osborne. Hamiltonian complexity. *Reports on Progress in Physics*, 75(2):022001, Jan 2012. ISSN 1361-6633. doi: 10.1088/0034-4885/75/2/022001. URL `http://dx.doi.org/10.1088/0034-4885/75/2/022001`.

[87] Yoshihiko Ozaki, Shuhei Watanabe, and Masaki Onishi. Accelerating the nelder-mead method with predictive parallel evaluation. In *6th ICML Workshop on Automated Machine Learning*, 2019.

[88] G. Pagano, A. Bapat, P. Becker, K. S. Collins, A. De, P. W. Hess, H. B. Kaplan, A. Kyprianidis, W. L. Tan, C. Baldwin, L. T. Brady, A. Deshpande, F. Liu, S. Jordan, A. V. Gorshkov, and C. Monroe. Quantum approximate optimization of the long-range ising model with a trapped-ion quantum simulator, 2020.

[89] Hariphan Philathong, Vincent Nathan Akshay, Ksenia Samburskaya, and Jacob Biamonte. Computational phase transitions: Benchmarking ising machines and quantum optimisers, 2020.

[90] Sebastian Pinski. Adiabatic quantum computing. 08 2011.

[91] Jan Poland and Thomas Zeugmann. Clustering pairwise distances with missing data: Maximum cuts versus normalized cuts. In Ljupčo Todorovski, Nada Lavrač, and Klaus P. Jantke, editors, *Discovery Science*, pages 197–208, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46493-8.

[92] Michael JD Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*, pages 51–67. Springer, 1994.

[93] Michael JD Powell. The newuoa software for unconstrained optimization without derivatives. In *Large-scale nonlinear optimization*, pages 255–297. Springer, 2006.

[94] Michael JD Powell. The bobyqa algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pages 26–46, 2009.

[95] Miguel Paredes Quinones and Catarina Junqueira. Tailored variational forms for certain linear inequality constraints in quadratic binary optimization problems, 2020.

[96] Matthew Radzihovsky, J. Murphy, and Mason Swofford. A qaoa solution to the traveling salesman problem using pyquil. 2019.

[97] Salonik Resch and Ulya R Karpuzcu. Benchmarking quantum computers and the impact of quantum noise. *arXiv preprint arXiv:1912.00546*, 2019.

[98] Rigetti. Variational-quantum-eigensolver (vqe), 2016. URL `https://grove-docs.readthedocs.io/en/latest/vqe.html`.

[99] Christoph Roch, Alexander Impertro, Thomy Phan, Thomas Gabor, Sebastian Feld, and Claudia Linnhoff-Popien. Cross entropy hyperparameter optimization for constrained problem hamiltonians applied to qaoa, 2020.

[100] Yue Ruan, Samuel Marsh, Xilin Xue, Xi Li, Zhihao Liu, and Jingbo Wang. Quantum approximate algorithm for np optimization problems with constraints, 2020.

[101] Yue Ruan, Samuel Marsh, Xilin Xue, Zhihao Liu, and Jingbo Wang. The quantum approximate algorithm for solving traveling salesman problem. *Computers, Materials & Continua*, 63(3):1237–1247, 2020. ISSN 1546-2226. doi: 10.32604/cmc.2020.010001. URL `http://www.techscience.com/cmc/v63n3/38872`.

[102] Kenneth Michael Rudinger and Erik Nielsen. Quantum characterization verification & validation (qcvv) tutorial: Calibration-fee characterization. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2016.

[103] Thomas Philip Runarsson and Xin Yao. Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(2):233–243, 2005.

[104] Aritra Sarkar, Zaid Al-Ars, Carmen G Almudever, and Koen Bertels. An algorithm for dna read alignment on quantum accelerators. *arXiv preprint arXiv:1909.05563*, 2019.

[105] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. Quaser – quantum accelerated de novo dna sequence reconstruction, 2020.

[106] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. Estimating algorithmic information using quantum computing for genomics applications. 2021.

[107] Ruslan Shaydulin and Yuri Alexeev. Evaluating quantum approximate optimization algorithm: A case study. *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*, Oct 2019. doi: 10.1109/igsc48788.2019.8957201. URL `http://dx.doi.org/10.1109/IGSC48788.2019.8957201`.

[108] Steven Skiena. Stony brook algorithm repository, 2020. URL `https://algorist.com/problems/SetPacking.html`.

[109] Larry Stockmeyer. On approximation algorithms for# p. *SIAM Journal on Computing*, 14(4):849–861, 1985.

[110] Michael Streif and Martin Leib. Training the quantum approximate optimization algorithm without access to a quantum processing unit, 2019.

[111] Quantum AI team and collaborators. qsim, September 2020. URL https://doi.org/10.5281/zenodo.4023103.

[112] Pontus Vikstål, Mattias Grönkvist, Marika Svensson, Martin Andersson, Göran Johansson, and Giulia Ferrini. Applying the quantum approximate optimization algorithm to the tail-assignment problem. *Physical Review Applied*, 14(3):034009, 2020.

[113] Rui-Sheng Wang and Li-Min Wang. Maximum cut in fuzzy nature: Models and algorithms. *Journal of Computational and Applied Mathematics*, 234(1):240 – 252, 2010. ISSN 0377-0427. doi: https://doi.org/10.1016/j.cam.2009.12.022. URL http://www.sciencedirect.com/science/article/pii/S0377042709008309.

[114] Matteo M. Wauters, Emanuele Panizon, Glen B. Mbeng, and Giuseppe E. Santoro. Reinforcement-learning-assisted quantum optimization. *Physical Review Research*, 2(3), Sep 2020. ISSN 2643-1564. doi: 10.1103/physrevresearch.2.033446. URL http://dx.doi.org/10.1103/PhysRevResearch.2.033446.

[115] Madita Willsch, Dennis Willsch, Fengping Jin, Hans De Raedt, and Kristel Michielsen. Benchmarking the quantum approximate optimization algorithm. *Quantum Information Processing*, 19:197, 2020.

[116] Y Xiang, DY Sun, W Fan, and XG Gong. Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, 233(3):216–220, 1997.

[117] Kevin C. Young, Mohan Sarovar, and Robin Blume-Kohout. Error suppression and error correction in adiabatic quantum computation: Techniques and challenges. *Physical Review X*, 3(4), Nov 2013. ISSN 2160-3308. doi: 10.1103/physrevx.3.041013. URL http://dx.doi.org/10.1103/PhysRevX.3.041013.

[118] Serguei Zertchaninov and Kaj Madsen. *A C++ Programme for Global Optimization.* IMM, Institute of Mathematical Modelling, Technical University of Denmark, 1998.

[119] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D. Lukin. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *Physical Review X*, 10(2), Jun 2020. ISSN 2160-3308. doi: 10.1103/physrevx.10.021067. URL http://dx.doi.org/10.1103/PhysRevX.10.021067.

[120] A. A. Zhukov, E. O. Kiktenko, A. A. Elistratov, W. V. Pogosov, and Yu. E. Lozovik. Quantum communication protocols as a benchmark for programmable quantum computers. *Quantum Information Processing*, 18(1), Dec 2018. ISSN 1573-1332. doi: 10.1007/s11128-018-2144-y. URL http://dx.doi.org/10.1007/s11128-018-2144-y.

[121] Urban Škvorc, Tome Eftimov, and Peter Korošec. Cec real-parameter optimization competitions:progress from 2013 to 2018. 06 2019. doi: 10.1109/CEC.2019.8790158.